PROGRAMMING THE WEB

Subject Code: 10CS73	**I.A. Marks : 25**
Hours/Week : 04	**Exam Hours: 03**
Total Hours : 52	**Exam Marks: 100**

PART - A

UNIT – 1 **6 Hours**

Fundamentals of Web, XHTML – 1: Internet, WWW, Web Browsers and Web Servers, URLs, MIME, HTTP, Security, The Web Programmers Toolbox.
XHTML: Basic syntax, Standard structure, Basic text markup, Images, Hypertext Links.

UNIT – 2 **7 Hours**

XHTML – 2, CSS: XHTML (continued): Lists, Tables, Forms, Frames CSS: Introduction, Levels of style sheets, Style specification formats, Selector forms, Property value forms, Font properties, List properties, Color, Alignment of text, The box model, Background images, The <span> and <div> tags, Conflict resolution.

UNIT – 3 **6 Hours**

Javascript: Overview of Javascript, Object orientation and Javascript, Syntactic characteristics, Primitives, operations, and expressions, Screen output and keyboard input, Control statements, Object creation and modification, Arrays, Functions, Constructors, Pattern matching using regular expressions, Errors in scripts, Examples.

UNIT – 4 **7 Hours**

Javascript and HTML Documents, Dynamic Documents with Javascript: The Javascript execution environment, The Document Object Model, Element access in Javascript, Events and event handling, Handling events from the Body elements, Button elements, Text box and Password elements, The DOM 2 event model, The navigator object, DOM tree traversal and modification. Introduction to dynamic documents, Positioning elements, Moving elements, Element visibility, Changing colors and fonts, Dynamic content, Stacking elements, Locating the mouse cursor, Reacting to a mouse click, Slow movement of elements, Dragging and dropping elements.

PART - B

UNIT – 5 **6 Hours**

XML: Introduction, Syntax, Document structure, Document type definitions, Namespaces, XML schemas, Displaying raw XML documents, Displaying XML documents with CSS, XSLT style sheets, XML processors, Web services.

UNIT – 6 **7 Hours**

Perl, CGI Programming: Origins and uses of Perl, Scalars and their operations, Assignment statements and simple input and output, Control statements, Fundamentals of arrays, Hashes, References, Functions, Pattern matching, File input and output; Examples.

The Common Gateway Interface; CGI linkage; Query string format; CGI.pm module; A survey example; Cookies. Database access with Perl and MySQL

UNIT – 7 **6 Hours**

PHP: Origins and uses of PHP, Overview of PHP, General syntactic characteristics, Primitives, operations and expressions, Output, Control statements, Arrays, Functions, Pattern matching, Form handling, Files, Cookies, Session tracking, Database access with PHP and MySQL.

UNIT – 8 **7 Hours**

Ruby, Rails: Origins and uses of Ruby, Scalar types and their operations, Simple input and output, Control statements, Arrays, Hashes, Methods, Classes, Code blocks and iterators, Pattern matching.

Overview of Rails, Document requests, Processing forms, Rails applications with Databases, Layouts.

Text Books:

1. Robert W. Sebesta: Programming the World Wide Web, 4 Edition, Pearson Education, 2008. (Listed topics only from Chapters 1 to 9, 11 to 15)

Reference Books:

1. M. Deitel, P.J. Deitel, A. B. Goldberg: Internet & World Wide Web How to Program, 4th Edition, Pearson Education, 2004.
2. Chris Bates: Web Programming Building Internet Applications, 3rd Edition, Wiley India, 2007.
3. Xue Bai et al: The web Warrior Guide to Web Programming, Cengage Learning, 2003.

INDEX SHEET

UNIT - 1

Syllabus: UNIT - 1 Fundamentals of Web, XHTML – 1

Internet, WWW, Web Browsers and Web Servers; URLs; MIME;
HTTP; Security; The Web Programmers Toolbox.
XHTML: Origins and evolution of HTML and XHTML
Basic syntax
Standard XHTML document structure;
Basic text markup. Images, Hypertext Links.

Unit1 Fundamentals

1.1 A Brief Intro to the Internet

- Internet History
- Internet Protocols

Internet History

1.1.1 Origins

In the 1960s the U.S Department of Defense (DoD) became interested in developing a new large-scale computer network.

The purposes of this network were communications, program sharing and remote computer access. One fundamental requirement was that the network be sufficiently Robust so that even if some network nodes were lost due to damage or some more reason the network could continue to function.

The DoD's Advanced Research Projects Agency (ARPA) funded the construction of the first such network, and the network the first such network, and the network was named as ARPAnet in 1969.

The primary use of ARPAnet was simple text-based communications through e-mail.

A number of other networks were developed during the late 1970's and early 1980's with BITNET and CSNETT among them.

BITNET, which is an acronym for Because It's Time Network, developed at City University of NewYork. It was built initially to provide electronic mail and file transfers CSNET, which is an acronym for Computer Science Network, connected the university of Delware, Purdue

University, RAND corporation and many more universities with initial purpose was to provide Electronic mail.

For the variety of reasons, neither BITNET not CSNET became a dominant national network.

A new national network, NSFnet was created in 1986. It was funded by National Science Foundation (NSF). NSFnet initially connected NSF supercomputer centers.

By 1990, NSFnet had replaced ARPAnet for most nonmilitary uses. By 1992 NSFnet connected more than 1 million computers around the world.

In 1995 a small part of NSFnet returned to being a research network. The rest is known as the Internet.

As a Summary:

- ARPAnet - late 1960s and early 1970s
  - Network reliability
  - For ARPA-funded research organizations
- BITnet, CSnet - late 1970s & early 1980s
  - email and file transfer for other institutions
- NSFnet - 1986
  - Originally for non-DOD funded places
  - Initially connected five supercomputer centers
  - By 1990, it had replaced ARPAnet for non-military uses
  - Soon became the network for all (by the early 1990s)
- NSFnet eventually became known as the Internet

1.1.2 What the Internet is:

- Internet is a huge collection of computers connected in a communications network.
- It is a network of network rather than a network of computers.
- Using Internet many people can share resources and can communicate with each other.
- To have Internet service your computer must be connected to the Internet Service Providers (ISP) through cables modem, phone-line modem or DSL.
- The Internet employs a set of standardized protocols which allow for the sharing of resources. These standars are known by the Internet Protocol Suite.
- At the lowest level, since 1982, all connections use TCP/IP

1.1.3 Internet Protocols (IP) Addresses

■ Internet Protocol (IP) Addresses

  - ❑ Every node has a unique numeric address
  - ❑ Form: 32-bit binary number
- ■ IP address is divided into 2 main part:
  - ❑ Network number and
  - ❑ Host number
- ■ IP addresses usually are written as four 8-bit numbers separated by dots

NETWORK NUMBER	HOST NUMBER

- ■ Organizations are assigned groups of IPs for their computers
- ■ The are 5 classes of IP address

Sl.no.	IP address Class	Format	Pupose
1.	Class A	N.H.H.H	Few large organization use this class addressing
2.	Class B	N.N.H.H	Medium size organizations use this addressing
3.	Class C	N.N.N.H	Relatively small organizations use this class

Here N stands for Network number and H stands for Host number. For example, a small organization may be assigned 256 IP addresses, such as 191.28.121.0 to 191.28.121.255

- ■ Problem: By the mid-1980s, several different protocols had been invented and were being used on the Internet, all with different user interfaces (Telnet, FTP, Usenet, mailto

1.1.4 Domain names

- Form: host-name.domain-names
- First domain is the smallest; last is the largest
- Last domain specifies the type of organization
- Fully qualified domain name - the host name and all of the domain names
- DNS servers - convert fully qualified domain names to IPs
- Few domains are:
  - Edu –Extension for Educational institutions
  - Com – Specifies a Company
  - Gov – Specifies government
  - Org – Other kind of organization
- Even Domain specifies the country name
  - in – India
  - pk – Pakistan

- au – Australia
- us – United states

Domain Name Conversion

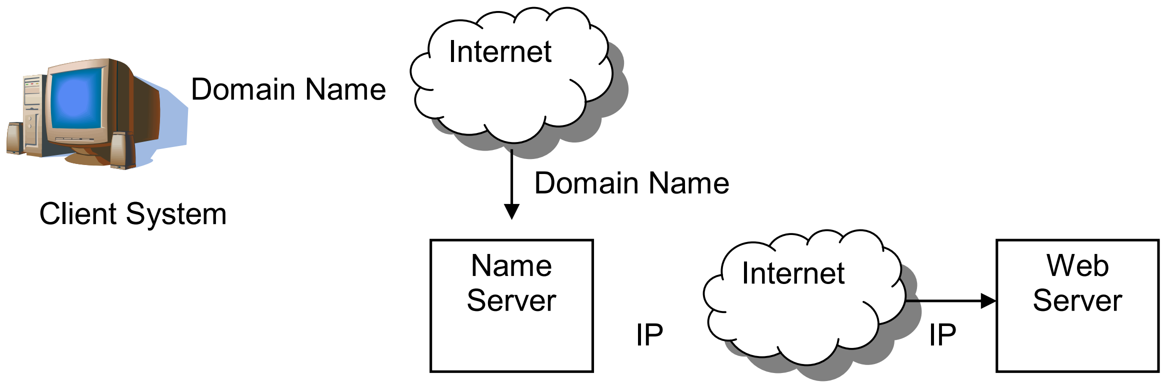


Fig. Domain Name Conversion

IP addresses are the address used internally by the Internet, the fully qualified domain name of the destination for a message, which is given browser, must be converted to an IP address before the message can be transmitted on the internet to the destination. These conversions are done by system software called Name Servers.

Name Servers server a collection of machines on the Internet and are operated by organizations that are responsible for the part of the Internet to which those machines are connected.

All documents requested from the browsers are routed to the nearest name server. If the name server can convert the fully qualified domain name to an IP address. If it cannot , the name server sends the fully qualified domain name to another name server for conversion.

The figure 1 shows how fully qualified domain names requested by a browser are translated into IPs before they are routed to the appropriate web server.

One way to determine the IP address of the website by using telnet.

If we want to know the IP address of www. Google.co.in, go to Dos prompt and type telnet **www.google.co.in**

PROTOCOLS

By the mid – 198s, a collection of different protocols that run on top of TCP/IP had been developed to support a variety of Internet users. Among those the most common were telnet, ftp, usenet, mailto

Uses:

- telnet – which was developed to allow a user on one computer on the Internet to log on to and use another computer on the Internet.[Remote Login]
- ftp[file transfer protocol] - which was developed to transfer file among computers on the Internet.
- usenet – Which was developed to serve as an electronic bulletin board.
- mailto – which was developed to allow messages to be sent from the user of one computer on the Internet to other users on other computer on the Internet.

Client and Server

- Clients and Servers are programs that communicate with each other over the Internet
- A Server runs continuously, waiting to be contacted by a Client
  - Each Server provides certain services
  - Services include providing web pages
- A Client will send a message to a Server requesting the service provided by that server
  - The client will usually provide some information, parameters, with the request

1.2 The World-Wide Web

- A possible solution to the proliferation of different protocols being used on the Internet

1.2.1 Origins

- Tim Berners-Lee at CERN proposed the Web in 1989
  - Purpose: to allow scientists to have access to many databases of scientific work through their own computers
- Document form: hypertext
- Pages? Documents? Resources?
  - We'll call them documents
- Hypermedia – more than just text – images, sound, etc.

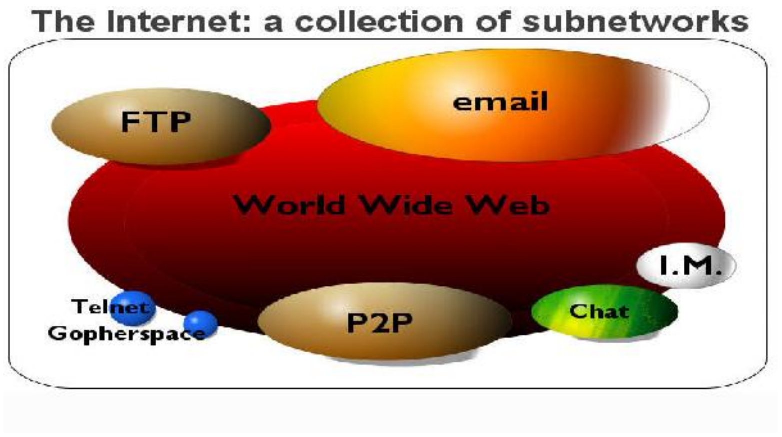
1.2.2 Web or Internet?

- The Web uses one of the protocols, http, that runs on the Internet--there are several others (telnet, mailto, etc.)
- The Internet is a massive network of networks, a networking infrastructure. It connects millions of computers together globally, forming a network in which any computer can communicate with any other computer as long as they are both

connected to the Internet. Information that travels over the Internet does so via a variety of languages known as protocols.

- The World Wide Web, or simply Web, is a way of accessing information over the medium of the Internet. The Web uses the HTTP protocol The Web also utilizes browsers, such as Internet Explorer or Firefox, to access Web documents called Web pages that are linked to each other via hyperlinks. Web documents also contain graphics, sounds, text and video.
- The Internet is the large container, and the Web is a part within the container.
- But to be technically precise, the Net is the restaurant, and the Web is the most popular dish on the menu.
- Browsers are used to connect to the www part of the internet.

Here is a conceptual diagram of the Internet and how it contains many forms of online communications



The Internet and the Web work together, but they are not the same thing. The Internet provides the underlying structure, and the Web utilizes that structure to offer content, documents, multimedia, etc.
The Internet is at its most basic definition an electronic communications network. It is the structure on which the World Wide Web is based.

1.3 Web Browsers

- Browsers are clients - always initiate, servers react (although sometimes servers require responses)
- Mosaic - NCSA (Univ. of Illinois), in early 1993
  - First to use a GUI, led to explosion of Web use
  - Initially for X-Windows, under UNIX, but was ported to other platforms by late 1993
- Most requests are for existing documents, using HyperText Transfer Protocol (HTTP)
  - But some requests are for program execution, with the output being returned as a document

1.4 Web Servers

- Provide responses to browser requests, either existing documents or dynamically built documents
- All communications between browsers and servers use Hypertext Transfer Protocol (HTTP)
- Apache, Microsoft internet information server (IIS)

1.4.1 Web Server Operation

- Web servers run as background processes in the operating system
  - Monitor a communications port on the host, accepting HTTP messages when they appear
- All current Web servers came from either
  1. The original from CERN
  2. The second one, from NCSA

1.4.2 Web Server Operation Details

- Web servers have two main directories:
  1. Document root (servable documents)
  2. Server root (server system software)
- Document root is accessed indirectly by clients
  - Its actual location is set by the server configuration file
  - Requests are mapped to the actual location
  - Path /admin/web/topdocs/xyz.html
- Server root – stores server and its support software
- Virtual document trees : many servers allow part of the servable document collection to be stored outside the directory of document root. The secondary areas from which document can be served are called virtual document trees.
- Proxy servers : Some servers can serve documents that are in the document root of other machines on the web and those servers are called proxy servers.

Difference between apache and IIS

Apache Web Server	IIS web Server
1. It is an open source product. 2. Apache web server is useful on both UNIX based systems and on windows platform. 3. The apache web server can be	1. It is a vendorspecific product and can be used on windows only. 2. IIS web server is used on windows platform. 3. The IIS server can be controlled by modifying the window based management programs called IIS span-in. We can access 115 span-in by

controlled by editing the configuration file http.conf	going to control panel → Administrative tools → IIS admin

1.5 URLs (uniform resource locators)

1.5.1 General form:

scheme: object-address

- ❑ The scheme is often a communications protocol, such as telnet or ftp
- ■ For the http protocol, the object-address is: fully qualified domain name/doc path
- ■ For the file protocol, only the doc path is needed
- ■ Host name may include a port number
- ■ URLs cannot include spaces or any of a collection of other special characters (semicolons, colons, ...)
- ■ The doc path may be abbreviated as a partial path
  - ❑ The rest is furnished by the server configuration
- ■ If the doc path ends with a slash, it means it is a directory

1.6 Multipurpose Internet Mail Extensions (MIME)

- ■ Originally developed for email
- ■ Used to specify the form of a file returned by the server
- ■ Type specifications
  - ❑ Form:

type/subtype

  - ❑ Examples: text/plain, text/html, image/gif, image/jpeg, video/mpeg, video/rm, video/quicktime

Browser gets the type explicitly from the server

1.7 The HyperText Transfer Protocol

- ■ The protocol used by ALL Web communications
- ■ Current version of HTTP is 1.1
- ■ Consists of 2 phases → request phase
  → response phase

http communication[request or response] between a browser and a web server consists of 2 parts → header-consists information about communication and
→ body – consists data of communication

1.7.1 Request Phase

- Form:

1. HTTP method domain part of URL HTTP ver.
2. Header fields
3. blank line semantics
4. Message body

1.7 The HyperText Transfer Protocol: Methods

- GET - Fetch a document
- POST - Execute the document, using the data in body
- HEAD - Fetch just the header of the document
- PUT - Store a new document on the server
- DELETE - Remove a document from the server
  - An example of the first line of a request:

GET /degrees.html HTTP/1.1

- Format of second line header field (optional)
  - Field name followed by a colon and the value of the field.

HTTP Headers

- Four categories of header fields:

General, request, response, & entity

- Common request fields:

Accept: text/plain
Accept: text/*
If-Modified_since: date

- Common response fields:

Content-length: 488
Content-type: text/html

- Can communicate with HTTP without a browser
  - telnet blanca.uccs.edu http
    - Creates connection to http port on …….. server

http command eg:
GET /respond.html HTTP/1.1
Host: blanca.uccs.edu

1.7.2 Response phase

- Form:

1. Status line
2. Response header fields
3. blank line
4. Response body

- Status line format:

HTTP version status code explanation

- Example: HTTP/1.1 200 OK

(Current version is 1.1)

- Status code is a three-digit number; first digit specifies the general status

1 => Informational
2 => Success
3 => Redirection
4 => Client error
5 => Server error

- The header field, Content-type, is required

HTTP Response Example

```
HTTP/1.1  200  OK
Date: Tues, 18 May 2004 16:45:13 GMT
Server: Apache (Red-Hat/Linux)
Last-modified: Tues, 18 May 2004 16:38:38 GMT
Etag: "841fb-4b-3d1a0179"
Accept-ranges: bytes
Content-length: 364
Connection: close
Content-type: text/html, charset=ISO-8859-1
```

- Both request headers and response headers must be followed by a blank line

1.8 Note on security?

1.9 The Web Programmer's Toolbox

- Document languages and programming languages that are the building blocks of the web and web programming
- XHTML
- Plug-ins
- Filters
- XML
- Javascript
- Java, Perl, Ruby, PHP

1.9.1 XHTML

- To describe the general form and layout of documents
- An XHTML document is a mix of content and controls
    - Controls are tags and their attributes
        - Tags often delimit content and specify something about how the content should be arranged in the document
        - Attributes provide additional information about the content of a tag

1.9.2 Creating XHTML documents

- XHTML editors - make document creation easier
    - Shortcuts to typing tag names, spell-checker,
- WYSIWYG XHTML editors
    - Need not know XHTML to create XHTML documents

1.9.3 Plugins and Filters

- Plug ins
    - Integrated into tools like word processors, effectively converting them to WYSIWYG XHTML editors
- Filters
    - Convert documents in other formats to XHTML

Plugins and Filters: Advantages and Disadvantages

- Advantages of both filters and plug-ins:
    - Existing documents produced with other tools can be converted to XHTML documents
    - Use a tool you already know to produce XHTML
- Disadvantages of both filters and plug-ins:
    - XHTML output of both is not perfect - must be fine tuned
    - XHTML may be non-standard
    - You have two versions of the document, which are difficult to synchronize

1.9.4 XML

- A meta-markup language
- Used to create a new markup language for a particular purpose or area
- Because the tags are designed for a specific area, they can be meaningful
- No presentation details
- A simple and universal way of representing data of any textual kind

1.9.5 JavaScript

- A client-side HTML-embedded scripting language
- Only related to Java through syntax
- Dynamically typed and not object-oriented
- Provides a way to access elements of HTML documents and dynamically change them

1.9.6 Java

- General purpose object-oriented programming language
- Based on C++, but simpler and safer
- Our focus is on applets, servlets, and JSP

1.9.7 Perl

- Provides server-side computation for HTML documents, through CGI
- Perl is good for CGI programming because:
    - Direct access to operating systems functions
    - Powerful character string pattern-matching operations
    - Access to database systems
- Perl is highly platform independent, and has been ported to all common platforms
- Perl is not just for CGI

1.9.8 PHP

- A server-side scripting language
- An alternative to CGI

- Similar to JavaScript
- Great for form processing and database access through the Web

1.10 Origins and Evolution of HTML

- HTML was defined with SGML
- Original intent of HTML: General layout of documents that could be displayed by a wide variety of computers
- Recent versions:
  - HTML 3.2 – 1997
    - Introduced many new features and deprecated many older features
  - HTML 4.01 - 1999 - A cleanup of 4.0
  - XHTML 1.0 - 2000
    - Just 4.01 defined using XML, instead of SGML
  - XHTML 1.1 – 2001
    - Modularized 1.0, and drops frames
    - We'll stick to 1.1, except for frames
- **Reasons to use XHTML, rather than HTML:**
  1. HTML has lax syntax rules, leading to sloppy and sometime ambiguous documents
     – XHTML syntax is much more strict, leading to clean and clear documents in a standard form
  2. HTML processors do not even enforce the few syntax rule that do exist in HTML
  3. The syntactic correctness of XHTML documents can be validated

1.11 Basic Syntax

- Elements are defined by tags (markers)
  - Tag format:
    - Opening tag: <name>
    - Closing tag: </name>
  - The opening tag and its closing tag together specify a container for the *content* they enclose
- Not all tags have content
  - If a tag has no content, its form is <name />
- The container and its content together are called an *element*
- If a tag has attributes, they appear between its name and the right bracket of the opening tag
- Comment form: <!- … ->
- Browsers ignore comments, unrecognizable tags, line breaks, multiple spaces, and tabs
- Tags are suggestions to the browser, even if they are recognized by the browser

1.12 HTML Document Structure

- <html>, <head>, <title>, and <body> are required in every document
- Every XHTML document must begin with:

```
<?xml version = ″1.0″?>
<!DOCTYPE html PUBLIC ″-//w3c//DTD XHTML 1.1//EN″
  http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd>
```

- The whole document must have <html> as its root
- html must have the xmlns attribute:

```
 <html xmlns = ″http://www.w3.org/1999/xhtml″
```

- A document consists of a head and a body
- The <title> tag is used to give the document a title, which is normally displayed in the browser's window title bar (at the top of the display)
- Prior to XHTML 1.1, a document could have either a body or a frameset

1.13 Basic Text Markup

- Text is normally placed in paragraph elements
- *Paragraph Elements*
  - The <p> tag breaks the current line and inserts a blank line - the new line gets the beginning of the content of the paragraph
  - The browser puts as many words of the paragraph's content as will fit in each line

```
<?xml version = ″1.0″?>
  <!DOCTYPE html PUBLIC ″-//w3c//DTD XHTML 1.1//EN″
    http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd>
<!-- greet.hmtl
    A trivial document
    -->
<html xmlns = ″http://www.w3.org/1999/xhtml″>
  <head> <title> Our first document </title>
  </head>
  <body>
    <p>
     Greetings from your Webmaster!
    </p>
  </body>
</html>
```

- W3C HTML Validation Service

  http://validator.w3.org/file-upload.html
- Line breaks
  - The effect of the <br /> tag is the same as that of <p>, except for the blank line

        - No closing tag!
- Example of paragraphs and line breaks

```
On the plains of hesitation <p> bleach the
bones of countless millions </p> <br />
who, at the dawn of victory <br /> sat down
to wait, and waiting, died.
```

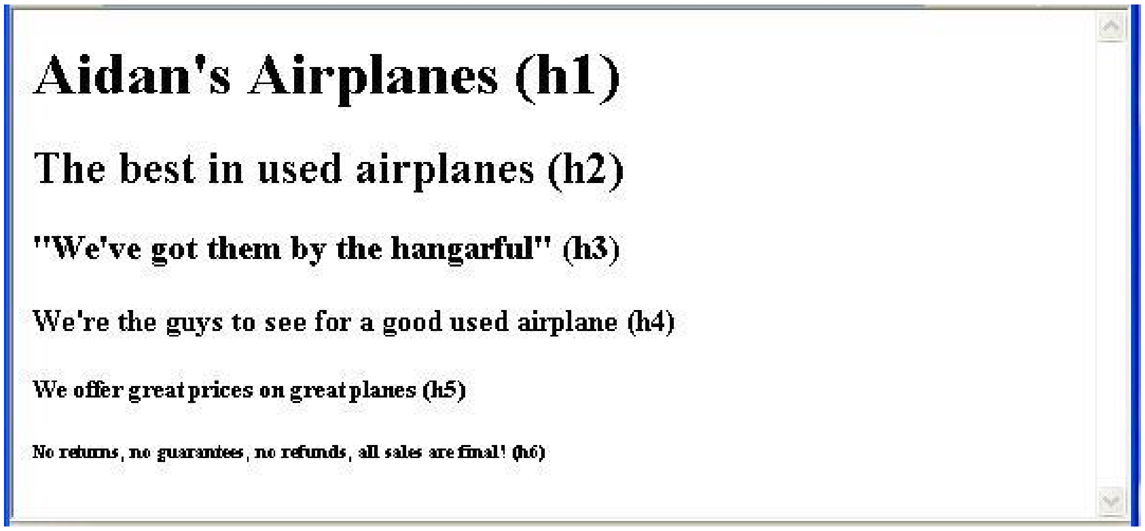
- Typical display of this text:

On the plains of hesitation

bleach the bones of countless millions
who, at the dawn of victory
sat down to wait, and waiting, died.

- *Headings*
    - ❑ Six sizes, 1 - 6, specified with <h1> to <h6>
    - ❑ 1, 2, and 3 use font sizes that are larger than the default font size
    - ❑ 4 uses the default size
    - ❑ 5 and 6 use smaller font sizes

```
<!-- headings.html
     An example to illustrate headings
     -->
<html xmlns = ″http://www.w3.org/1999/xhtml″>
  <head> <title> Headings </title>
  </head>
  <body>
    <h1> Aidan’s Airplanes (h1) </h1>
    <h2> The best in used airplanes (h2) </h2>
    <h3> "We’ve got them by the hangarful" (h3)
    </h3>
    <h4> We’re the guys to see for a good used
         airplane (h4) </h4>
    <h5> We offer great prices on great planes
         (h5) </h5>
    <h6> No returns, no guarantees, no refunds,
         all sales are final (h6) </h6>
  </body>
</html>
```



Aidan's Airplanes (h1)

The best in used airplanes (h2)

"We've got them by the hangarful" (h3)

We're the guys to see for a good used airplane (h4)

We offer great prices on great planes (h5)

No returns, no guarantees, no refunds, all sales are final! (h6)

- **Blockquotes**
  - **Content of <blockquote>**
  - **To set a block of text off from the normal flow and appearance of text**
  - **Browsers often indent, and sometimes italicize**
- ***Font Styles and Sizes (can be nested)***
  - **Boldface - <b>**
  - **Italics - <i>**
  - **Larger - <big>**
  - **Smaller - <small>**
  - **Monospace - <tt>**

The <big> sleet <big> in <big> <i> Crete
</i><br /> lies </big> completely </big>
in </big> the street
The sleet in *Crete*
lies completely in the street

- **These tags are not affected if they appear in the content of a <blockquote>, unless there is a conflict (e.g., italics)**
- ***Superscripts and subscripts***
  - **Subscripts with <sub>**
  - **Superscripts with <sup>**

Example: x<sub>2</sub><sup>3</sup>
Display: x23

- **Inline versus block elements**
- **All of this font size and font style stuff can be done with style sheets, but these tags are not yet deprecated**
- **Character Entities**

Char.	*Entity*	*Meaning*
&	**&amp;**	**Ampersand**

<	&lt;	Less than
>	&gt;	Greater than
”	&quot;	Double quote
’	&apos;	Single quote
¼	&frac14;	One quarter
½	&frac12;	One half
¾	&frac34;	Three quarters
°	&deg;	Degree
(space)	&nbsp;	Non-breaking space

- **Horizontal rules**
  - **<hr /> draws a line across the display, after a line break**
- **The meta element (for search engines) Used to provide additional information about a document, with attributes, not content**

Images

- **GIF (Graphic Interchange Format)**
  - **8-bit color (256 different colors)**
- **JPEG (Joint Photographic Experts Group)**
  - **24-bit color (16 million different colors)**
- **Both use compression, but JPEG compression is better**
- **Images are inserted into a document with the <img /> tag with the src attribute**
  - **The alt attribute is required by XHTML**
    - **Purposes:**
      1. **Non-graphical browsers**
      2. **Browsers with images turned off**

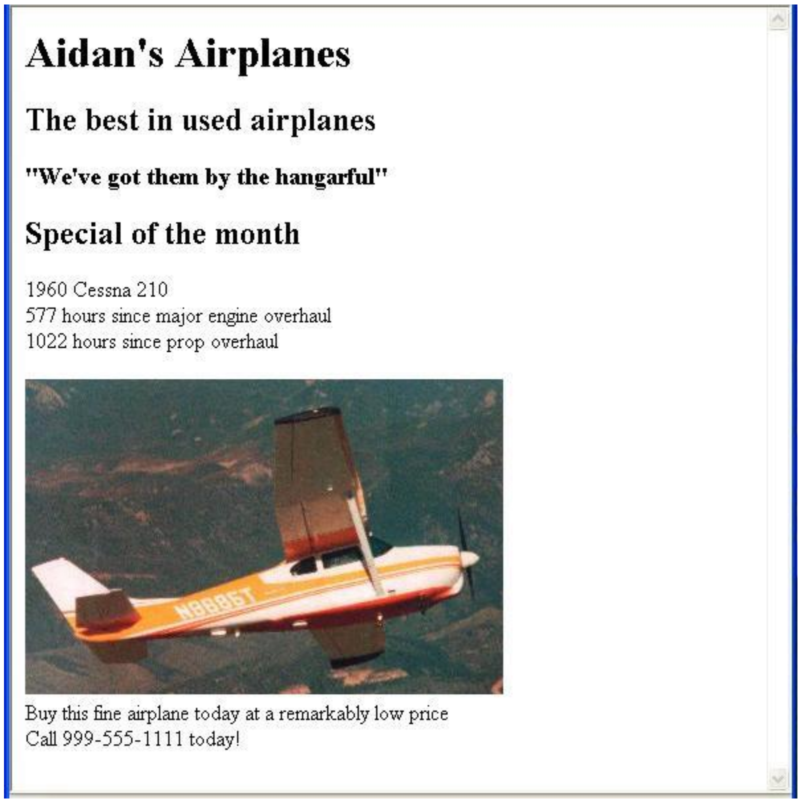
```
<img src = "comets.jpg"
     alt = "Picture of comets" />
```

- **The <img> tag has 30 different attributes, including width and height (in pixels)**
- **Portable Network Graphics (PNG)**
  - **Relatively new**
  - **Should eventually replace both gif and jpeg**

Eg:

```
<!-- image.html
    An example to illustrate an image
    -->
<html xmlns = ″http://www.w3.org/1999/xhtml″>
  <head> <title> Images </title>
  </head>
  <body>
```

```
    <h1> Aidan's Airplanes </h1>
    <h2> The best in used airplanes </h2>
    <h3> "We've got them by the hangarful"
    </h3>
    <h2> Special of the month </h2>
    <p>
      1960 Cessna 210 <br />
      577 hours since major engine overhaul
      <br />
      1022 hours since prop overhaul
      <br /><br />
      <img src = "c210new.jpg"
           alt = "Picture of a Cessna 210"/>
      <br />
      Buy this fine airplane today at a
      remarkably low price <br />
      Call 999-555-1111 today!
    </p>
  </body>
</html>
```



Aidan's Airplanes

The best in used airplanes

"We've got them by the hangarful"

Special of the month

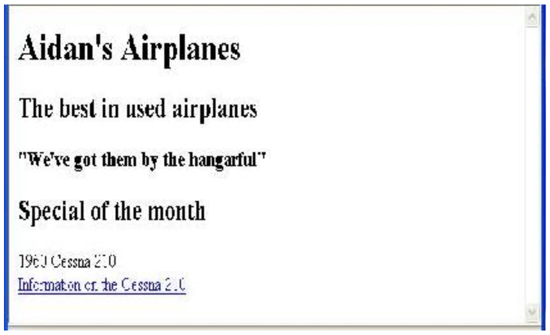
1960 Cessna 210
577 hours since major engine overhaul
1022 hours since prop overhaul

Buy this fine airplane today at a remarkably low price
Call 999-555-1111 today!

2.2 Hypertext Links

- **Hypertext is the essence of the Web!**
- **A link is specified with the href (*h*ypertext *ref*erence) attribute of <a> (the anchor tag)**
  - **The content of <a> is the visual link in the document**
  - **If the target is a whole document (not the one in which the link appears), the target need not be specified in the target document as being the target**
- **Note: Relative addressing of targets is easier to maintain and more portable than absolute addressing**

```
<!-- link.html
   An example to illustrate a link
   -->
<html xmlns = "http://www.w3.org/1999/xhtml">
 <head> <title> Links </title>
 </head>
 <body>
  <h1> Aidan's Airplanes </h1>
  <h2> The best in used airplanes </h2>
  <h3> "We've got them by the hangarful"
  </h3>
  <h2> Special of the month </h2>
  <p>
   1960 Cessna 210 <br />
   <a href = "C210data.html">
    Information on the Cessna 210 </a>
  </p>
 </body>
</html>
```



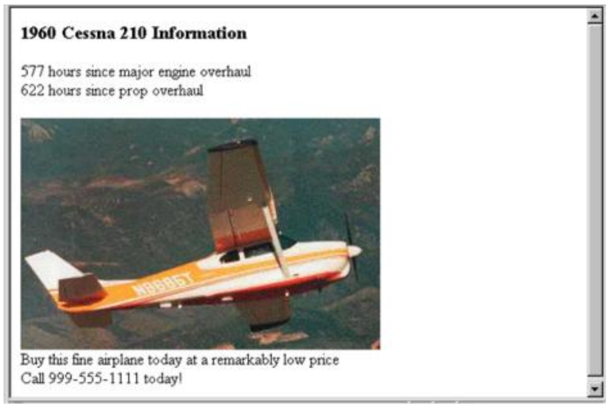
Aidan's Airplanes

The best in used airplanes

"We've got them by the hangarful"

Special of the month

1960 Cessna 210
Information on the Cessna 210



1960 Cessna 210 Information

577 hours since major engine overhaul
622 hours since prop overhaul

Buy this fine airplane today at a remarkably low price
Call 999-555-1111 today!

- **If the target is not at the beginning of the document, the target spot must be marked**
- **Target labels can be defined in many different tags with the id attribute, as in**

<h1 id = "baskets"> Baskets </h1>

- **The link to an id must be preceded by a pound sign (#); If the id is in the same document, this target could be**

```
<a href = "#baskets">
            What about baskets? </a>
```

- **If the target is in a different document, the document reference must be included**

<a href = "myAd.html#baskets”> Baskets </a>

- **Style note: a link should blend in with the surrounding text, so reading it without taking the link should not be made less pleasant**
- **Links can have images:**

```
  <a href = "c210data.html“>
  <img src = "smallplane.jpg"
    alt = "Small picture of an airplane " />
      Info on C210 </a>
```

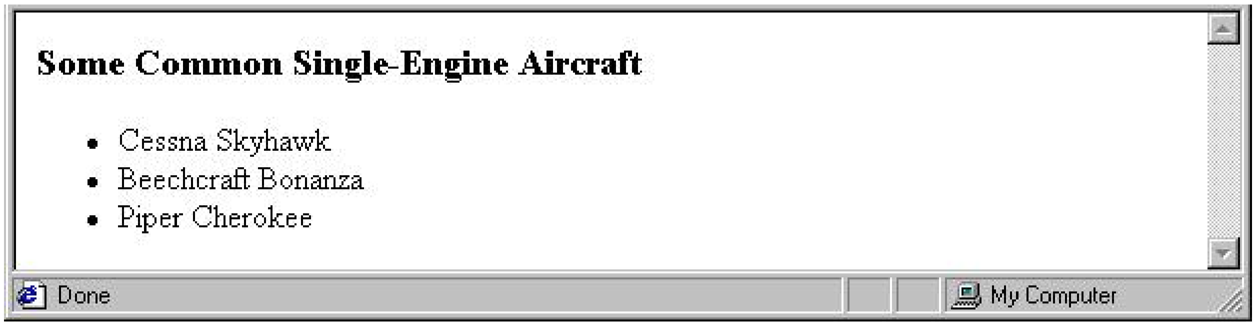
UNIT – 2: XHTML – 2: CSS: XHTML (continued..)

Lists; Tables;
Forms; Frames;
CSS: Introduction; Levels of style sheets; Property value forms;
Style specification formats; Selector forms;
Font properties; List properties;
Color; Alignment of text;
The Box model; Background images; The and tags; Conflict resolution.

2.1 Lists

- *Unordered lists*
- The list is the content of the <ul> tag
- List elements are the content of the <li> tag

```
<h3> Some Common Single-Engine Aircraft </h3>
 <ul>
   <li> Cessna Skyhawk </li>
   <li> Beechcraft Bonanza </li>
   <li> Piper Cherokee </li>
 </ul>
```

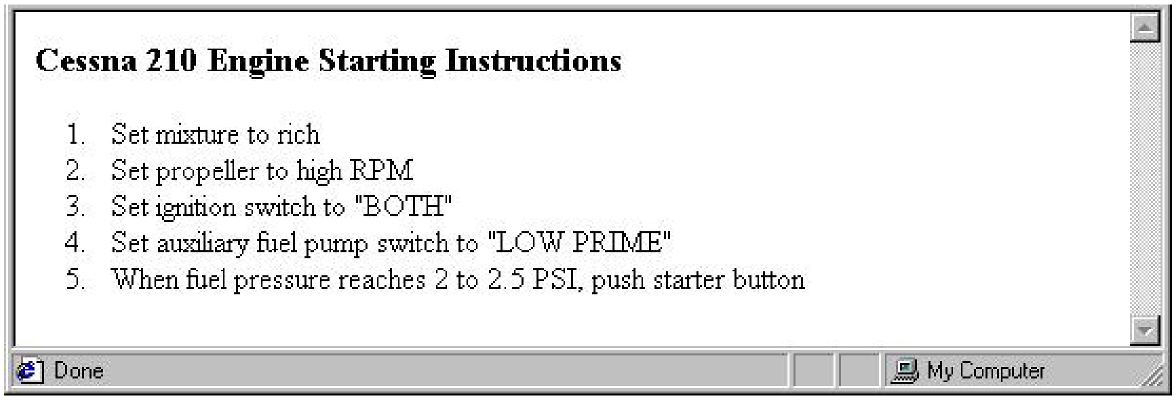


Some Common Single-Engine Aircraft

- Cessna Skyhawk
- Beechcraft Bonanza
- Piper Cherokee

- *Ordered lists*
    - The list is the content of the <ol> tag
    - Each item in the display is preceded by a sequence value

```
<h3> Cessna 210 Engine Starting Instructions
</h3>
<ol>
  <li> Set mixture to rich </li>
  <li> Set propeller to high RPM </li>
  <li> Set ignition switch to "BOTH" </li>
  <li> Set auxiliary fuel pump switch to
      "LOW PRIME" </li>
  <li> When fuel pressure reaches 2 to 2.5
      PSI, push starter button </li>
</ol>
```

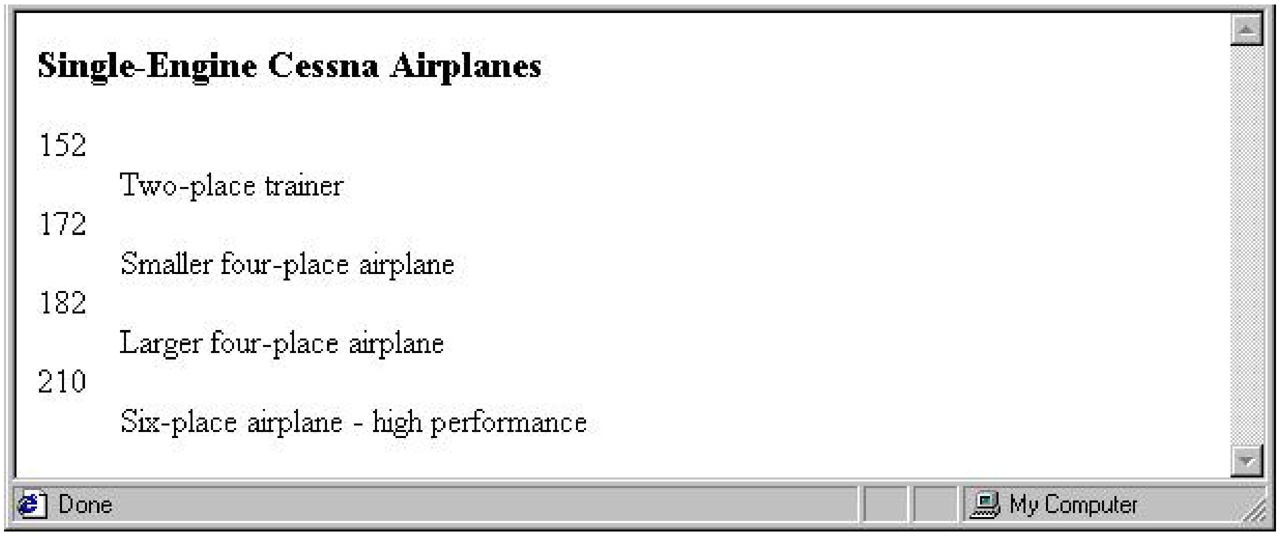


Cessna 210 Engine Starting Instructions

1. Set mixture to rich
2. Set propeller to high RPM
3. Set ignition switch to "BOTH"
4. Set auxiliary fuel pump switch to "LOW PRIME"
5. When fuel pressure reaches 2 to 2.5 PSI, push starter button

- *Definition lists (for glossaries, etc.)*
    - List is the content of the <dl> tag
    - Terms being defined are the content of the <dt> tag
    - The definitions themselves are the content of the <dd> tag

```
<h3> Single-Engine Cessna Airplanes </h3>
<dl >
  <dt> 152 </dt>
  <dd> Two-place trainer </dd>
  <dt> 172 </dt>
  <dd> Smaller four-place airplane </dd>
  <dt> 182 </dt>
  <dd> Larger four-place airplane </dd>
  <dt> 210 </dt>
  <dd> Six-place airplane - high performance
  </dd>
</dl>
```



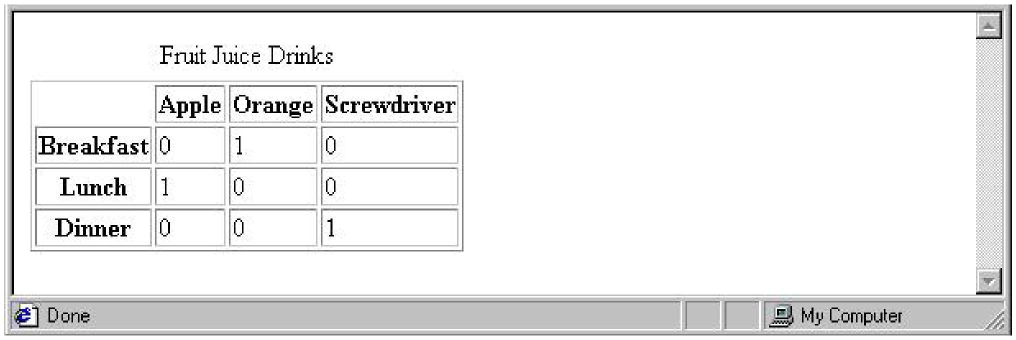
Single-Engine Cessna Airplanes

```
152
      Two-place trainer
172
      Smaller four-place airplane
182
      Larger four-place airplane
210
      Six-place airplane - high performance
```

2.2 Tables

- A table is a matrix of cells, each possibly having content
- The cells can include almost any element
- Some cells have row or column labels and some have data
- A table is specified as the content of a <table> tag
- A border attribute in the <table> tag specifies a border between the cells
- If border is set to "border", the browser's default width border is used
- The border attribute can be set to a number, which will be the border width
- Without the border attribute, the table will have no lines!
- Tables are given titles with the <caption> tag, which can immediately follow <table>
- Each row of a table is specified as the content of a <tr> tag
- The row headings are specified as the content of a <th> tag
- The contents of a data cell is specified as the content of a <td> tag

```
<table border = "border">
  <caption> Fruit Juice Drinks </caption>
    <tr>
      <th> </th>
      <th> Apple </th>
      <th> Orange </th>
      <th> Screwdriver </th>
    </tr>
    <tr>
      <th> Breakfast </th>
      <td> 0 </td>
      <td> 1 </td>
      <td> 0 </td>
    </tr>
    <tr>
      <th> Lunch </th>
      <td> 1 </td>
      <td> 0 </td>
      <td> 0 </td>
    </tr>
  </table>
```

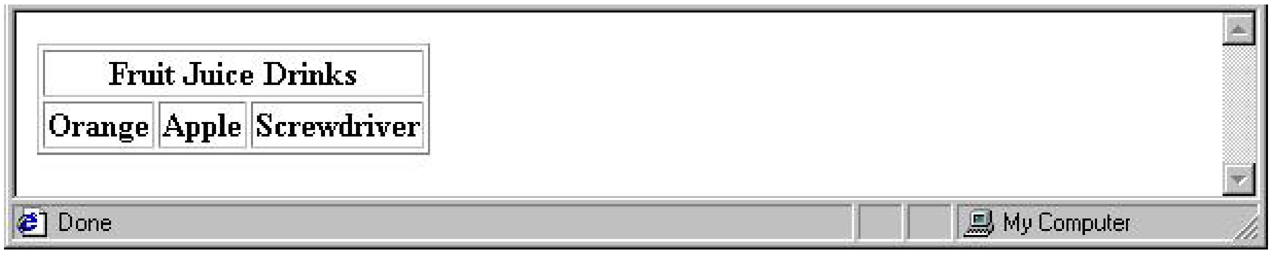


Fruit Juice Drinks

	Apple	Orange	Screwdriver
Breakfast	0	1	0
Lunch	1	0	0
Dinner	0	0	1

- A table can have two levels of column labels
  - If so, the colspan attribute must be set in the <th> tag to specify that the label must span some number of columns

```
<tr>
  <th colspan = "3"> Fruit Juice Drinks </th>
</tr>
<tr>
  <th> Orange </th>
  <th> Apple </th>
  <th> Screwdriver </th>
</tr>
```



Fruit Juice Drinks		
Orange	**Apple**	**Screwdriver**

- If the rows have labels and there is a spanning column label, the upper left corner must be made larger, using rowspan

```
<table border = "border">
  <tr>
```

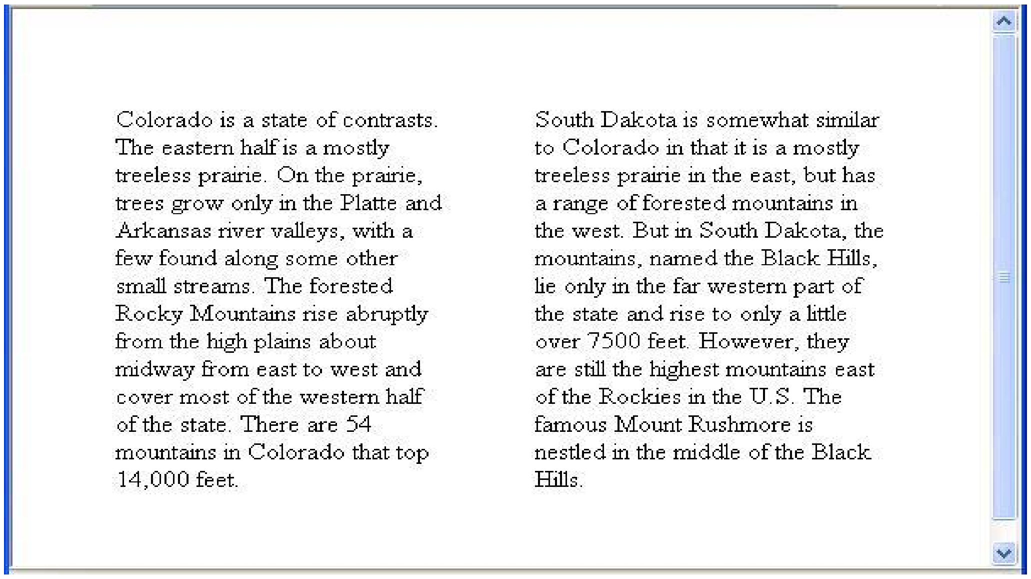
```
   <td rowspan = "2"> </td>
   <th colspan = "3"> Fruit Juice Drinks
   </th>
  </tr>
  <tr>
   <th> Apple </th>
   <th> Orange </th>
   <th> Screwdriver </th>
  </tr>
      …
</table>
```

- The align attribute controls the horizontal placement of the contents in a table cell
  - Values are left, right, and center (default)
  - align is an attribute of <tr>, <th>, and <td> elements
- The valign attribute controls the vertical placement of the contents of a table cell
  - Values are top, bottom, and center (default)
  - valign is an attribute of <th> and <td> elements

→ SHOW cell_align.html and display it

- The cellspacing attribute of <table> is used to specify the distance between cells in a table
- The cellpadding attribute of <table> is used to specify the spacing between the content of a cell and the inner walls of the cell

```
<table cellspacing = "50">
   <tr>
     <td> Colorado is a state of …
     </td>
     <td> South Dakota is somewhat…
     </td>
   </tr>
 </table>
```



Colorado is a state of contrasts. The eastern half is a mostly treeless prairie. On the prairie, trees grow only in the Platte and Arkansas river valleys, with a few found along some other small streams. The forested Rocky Mountains rise abruptly from the high plains about midway from east to west and cover most of the western half of the state. There are 54 mountains in Colorado that top 14,000 feet.

South Dakota is somewhat similar to Colorado in that it is a mostly treeless prairie in the east, but has a range of forested mountains in the west. But in South Dakota, the mountains, named the Black Hills, lie only in the far western part of the state and rise to only a little over 7500 feet. However, they are still the highest mountains east of the Rockies in the U.S. The famous Mount Rushmore is nestled in the middle of the Black Hills.

- *Table Sections*
  - Header, body, and footer, which are the elements: thead, tbody, and tfoot
  - If a document has multiple tbody elements, they are separated by thicker horizontal lines

2.3 Forms

- A form is the usual way information is gotten from a browser to a server
- HTML has tags to create a collection of objects that implement this information gathering
  - The objects are called *widgets* (e.g., radio buttons and checkboxes)
- When the Submit button of a form is clicked, the form's values are sent to the server
- All of the widgets, or components of a form are defined in the content of a <form> tag
  - The only required attribute of <form> is action, which specifies the URL of the application that is to be called when the Submit button is clicked

action =
"http://www.cs.ucp.edu/cgi-bin/survey.pl"

    - If the form has no action, the value of action is the empty string
- The method attribute of <form> specifies one of the two possible techniques of transferring the form data to the server, get and post
  - get and post are discussed in Chapter 10
- *Widgets*
  - Many are created with the <input> tag
    - The type attribute of <input> specifies the kind of widget being created
    - Text

- Creates a horizontal box for text input
- Default size is 20; it can be changed with the size attribute
- If more characters are entered than will fit, the box is scrolled (shifted) left

- If you don't want to allow the user to type more characters than will fit, set maxlength, which causes excess input to be ignored

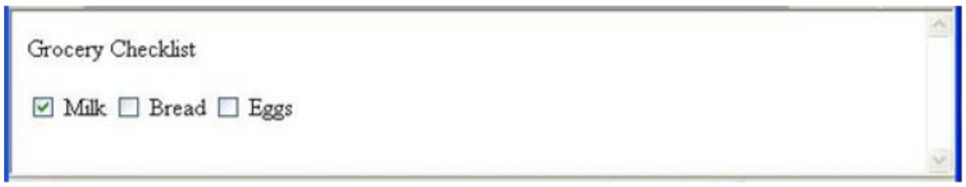
```
<input type = "text" name = "Phone"
    size = "12" >
```

2. Checkboxes - to collect multiple choice input

- Every checkbox requires a value attribute, which is the widget's value in the form data when the checkbox is 'checked'
  - A checkbox that is not 'checked' contributes no value to the form data
- By default, no checkbox is initially 'checked'
- To initialize a checkbox to 'checked', the checked attribute must be set to "checked"

- *Widgets* (continued)

```
Grocery Checklist
<form action = "">
 <p>
 <input type = "checkbox"  name ="groceries"
      value = "milk"  checked = "checked">
  Milk
 <input type = "checkbox"  name ="groceries"
      value = "bread">
  Bread
 <input type = "checkbox"  name = "groceries"
      value= "eggs">
  Eggs
 </p>
</form>
```


Grocery Checklist

☑ Milk ☐ Bread ☐ Eggs

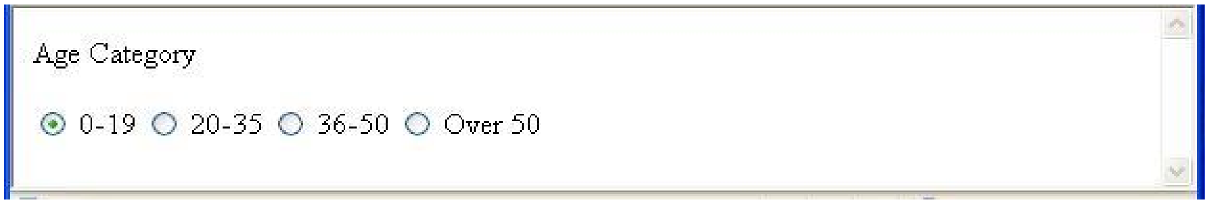
3. Radio Buttons - collections of checkboxes in which only one button can be 'checked' at a time

- Every button in a radio button group MUST have the same name

- *Widgets* (continued)

3. Radio Buttons (continued)

  - If no button in a radio button group is 'pressed', the browser often 'presses' the first one

Age Category
```
<form action = "">
 <p>
 <input type = "radio"  name = "age"
  value = "under20" checked = "checked"> 0-19
 <input type = "radio"  name = "age"
       value = "20-35"> 20-35
 <input type = "radio"  name = "age"
       value = "36-50"> 36-50
 <input type = "radio"  name = "age"
       value = "over50"> Over 50
 </p>
</form>
```

Age Category

● 0-19 ○ 20-35 ○ 36-50 ○ Over 50

4. Menus - created with <select> tags

- There are two kinds of menus, those that behave like checkboxes and those that behave like radio buttons (the default)
  - Menus that behave like checkboxes are specified by including the multiple attribute, which must be set to "multiple"
- The name attribute of <select> is required
- The size attribute of <select> can be included to specify the number of menu items to be displayed (the default is 1)
  - If size is set to > 1 or if multiple is specified, the menu is displayed as a pop-up menu

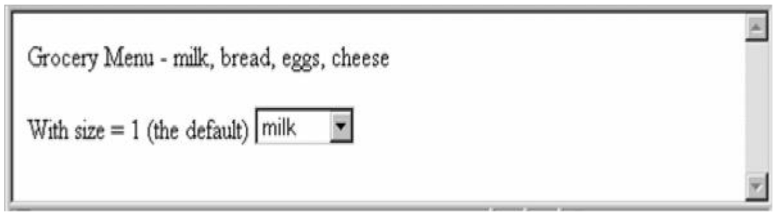
Menus (continued)

  - Each item of a menu is specified with an <option> tag, whose pure text content (no tags) is the value of the item
  - An <option> tag can include the selected attribute, which when assigned "selected" specifies that the item is preselected

Grocery Menu - milk, bread, eggs, cheese
```
<form action = "">
 <p>
```

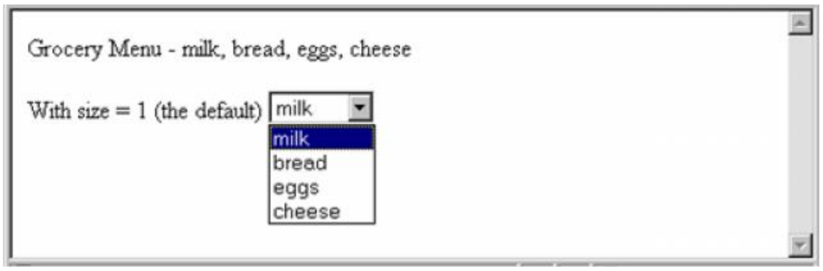
```
   With size = 1 (the default)
   <select name = "groceries">
     <option> milk </option>
     <option> bread </option>
     <option> eggs </option>
     <option> cheese </option>
   </select>
  </p>
</form>
```

- ***Widgets* (continued)**


Grocery Menu - milk, bread, eggs, cheese

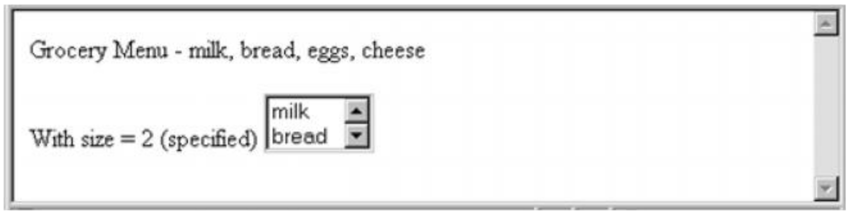
With size = 1 (the default) milk

- **After clicking the menu:**


Grocery Menu - milk, bread, eggs, cheese

With size = 1 (the default) milk

- **After changing `size` to 2:**


Grocery Menu - milk, bread, eggs, cheese

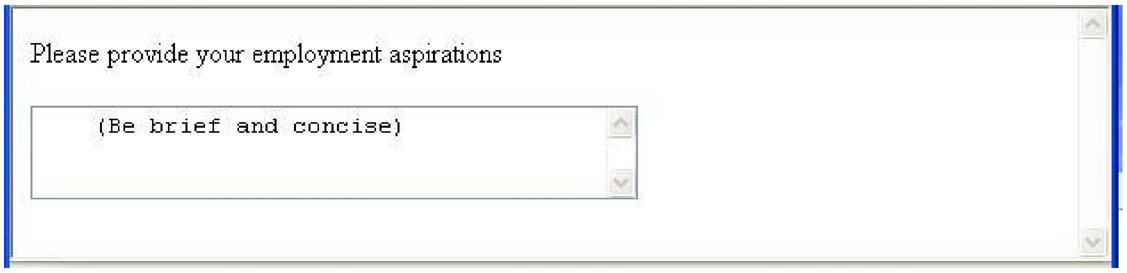
With size = 2 (specified)

- *Widgets* (continued)

5. Text areas - created with <textarea>
   - ❑ Usually include the rows and cols attributes to specify the size of the text area
   - ❑ Default text can be included as the content of <textarea>
   - ❑ Scrolling is implicit if the area is overfilled

Please provide your employment aspirations
<form action = "">

```
  <p>
  <textarea name = "aspirations"  rows = "3"
          cols = "40">
  (Be brief and concise)
  </textarea>
  </p>
</form>
```



Please provide your employment aspirations

(Be brief and concise)

- *Widgets* (continued)

6. Reset and Submit buttons

- Both are created with <input>

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

- Submit has two actions:
  1. Encode the data of the form
  2. Request that the server execute the server-resident program specified as the value of the action attribute of <form>
  3. A Submit button is required in every form

--> SHOW popcorn.html and display it

2.4 Frames

- Frames are rectangular sections of the display window, each of which can display a different document
- Because frames are no longer part of XHTML, you cannot validate a document that includes frames
- The <frameset> tag specifies the number of frames and their layout in the window
  - <frameset> takes the place of <body>
  - Cannot have both!
  - <frameset> must have either a rows attribute or a cols attribute, or both (usually the case)
  - Default is 1
  - The possible values for rows and cols are numbers, percentages, and asterisks
    - A number value specifies the row height in pixels - Not terribly useful!

  - A percentage specifies the percentage of total window height for the row - Very useful!
- ❑ An asterisk after some other specification gives the remainder of the height of the window
- ❑ Examples:

```
<frameset rows = "150, 200, 300">

<frameset rows = "25%, 50%, 25%">

<frameset rows = "50%, 20%, *" >

<frameset rows = "50%, 25%, 25%"
        cols = "40%, *">
```

- ■ The <frame> tag specifies the content of a frame
- ■ The first <frame> tag in a <frameset> specifies the content of the first frame, etc.
  - ❑ Row-major order is used
  - ❑ Frame content is specified with the src attribute
  - ❑ Without a src attribute, the frame will be empty (such a frame CANNOT be filled later)
- ■ If <frameset> has fewer <frame> tags than frames, the extra frames are empty
- ■ Scrollbars are implicitly included if needed (they are needed if the specified document will not fit)
- ■ If a name attribute is included, the content of the frame can be changed later (by selection of a link in some other frame)

→SHOW frames.html

- ■ Note: the Frameset standard must be specified in the DOCTYPE declaration

Eg:

```
<!-- contents.html
   The contents of the first frame of
   frames.html, which is the table of
   contents for the second frame
   -->
<html xmlns = ″http://www.w3.org/1999/xhtml″>
 <head> <title> Table of Contents Frame
   </title>
 </head>
 <body>
   <h4> Fruits </h4>
   <ul>
     <li> <a href = "apples.html"
```

```
          target = "descriptions">
          apples </a>
    <li> <a href = "bananas.html"
          target = "descriptions">
          bananas </a>
    <li> <a href = "oranges.html"
          target = "descriptions">
          oranges </a>
  </ul>
 </body>
</html>
```

- Nested frames - to divide the screen in more interesting ways

→ SHOW nested_frames.html

2.5 Introduction

- The CSS1 specification was developed in 1996
- CSS2 was released in 1998
- CSS3 is on its way
- CSSs provide the means to control and change presentation of HTML documents
- CSS is not technically HTML, but can be embedded in HTML documents
- Style sheets allow you to impose a standard style on a whole document, or even a whole collection of documents
- Style is specified for a tag by the values of its properties

2.6 Levels of Style Sheets

- There are three levels of style sheets
    - Inline - specified for a specific occurrence of a tag and apply only to that tag
      – This is fine-grain style, which defeats the purpose of style sheets - uniform style
    - Document-level style sheets - apply to the whole document in which they appear
    - External style sheets - can be applied to any number of documents
- When more than one style sheet applies to a specific tag in a document, the lowest level style sheet has precedence
    - In a sense, the browser searches for a style property spec, starting with inline, until it finds one (or there isn't one)

- Inline style sheets appear in the tag itself

- Document-level style sheets appear in the head of the document
- External style sheets are in separate files, potentially on any server on the Internet
    - Written as text files with the MIME type text/css

2.7 Linking an External Stylesheet

- A <link> tag is used to specify that the browser is to fetch and use an external style sheet file

```
<link rel = "stylesheet"  type = "text/css"
 href = "http://www.wherever.org/termpaper.css">
</link>
```

- External style sheets can be validated

http://jigsaw.w3.org/css-validator/
validator-upload.html

2.8 Style Specification Formats

- Format depends on the level of the style sheet
- Inline:
    - Style sheet appears as the value of the style attribute
    - General form:

```
style = "property_1: value_1;
      property_2: value_2;
      …
      property_n: value_n"
```

2.9 Format for Document-level

- Style sheet appears as a list of rules that are the content of a <style> tag
- The <style> tag must include the type attribute, set to "text/css"
- The list of rules must be placed in an HTML comment, because it is not HTML
- Comments in the rule list must have a different form - use C comments (/*…*/)

2.10 General Form, Document Level

- General form:

```
<style type = "text/css">
 <!--
rule list
 -->
</style>
```

- Form of the rules:
  selector {list of property/values}
    - Each property/value pair has the form:
      property: value
    - Pairs are separated by semicolons, just as in the value of a <style> tag

General Form, External style sheets

- Form is a list of style rules, as in the content of a <style> tag for document-level style sheets

Selector Forms: Simple

- The selector is a tag name or a list of tag names, separated by commas
  - h1, h3
  - p
- *Contextual selectors*
  - ol ol li

Class Selectors

- Used to allow different occurrences of the same tag to use different style specifications
- A style class has a name, which is attached to a tag name
  - p.narrow {property/value list}
  - p.wide {property/value list}
- The class you want on a particular occurrence of a tag is specified with the class attribute of the tag
- For example,

```
<p class = "narrow">
...
</p>
...
<p class = "wide">
...
</p>
```

Generic Selectors

- A generic class can be defined if you want a style to apply to more than one kind of tag
- A generic class must be named, and the name must begin with a period
- Example,

```
.really-big { … }
```

- Use it as if it were a normal style class

```
<h1 class = "really-big"> … </h1>
...
<p class = "really-big"> … </p>
```

id *Selectors*

- An id selector allow the application of a style to one specific element
- General form:

```
#specific-id {property-value list}
```

- Example:

```
#section14 {font-size: 20}
```

Pseudo Classes

- Pseudo classes are styles that apply when something happens, rather than because the target element simply exists
- Names begin with colons
- hover classes apply when the mouse cursor is over the element
- focus classes apply when an element has focus

Pseudo Class Example

```
<!-- pseudo.html -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> Checkboxes </title>
    <style type = "text/css">
      input:hover {color: red;}
      input:focus {color: green;}
    </style>
  </head>
  <body>
    <form action = "">
      <p>
        Your name:
        <input type = "text" />
      </p>
    </form>
  </body>
 </html>
```

Properties

- There are 60 different properties in 7 categories:
  - Fonts
  - Lists
  - Alignment of text
  - Margins
  - Colors
  - Backgrounds
  - Borders

Property Values

- Keywords - left, small, …
  - Not case sensitive
- Length - numbers, maybe with decimal points

- Units:
  - px - pixels
  - in - inches
  - cm - centimeters
  - mm - millimeters
  - pt - points
  - pc - picas (12 points)
  - em - height of the letter ‘m’
  - ex-height - height of the letter ‘x’
  - No space is allowed between the number and the unit specification e.g., 1.5 in is illegal!
- Percentage - just a number followed immediately by a percent sign
- URL values
  - url(protocol://server/pathname)
- Colors
  - Color name
  - rgb(n1, n2, n3)
    - Numbers can be decimal or percentages
  - Hex form: #XXXXXX
- Property values are inherited by all nested tags, unless overridden

Font Properties

- font-family
  - Value is a list of font names - browser uses the first in the list it has
  - font-family: Arial, Helvetica, Courier
  - Generic fonts: serif, sans-serif, cursive, fantasy, and monospace (defined in CSS)
    - Browser has a specific font for each
  - If a font name has more than one word, it should be single-quoted
- font-size
  - Possible values: a length number or a name, such as smaller, xx-large, etc.
- font-style
  - italic, oblique (useless), normal
- font-weight - degrees of boldness
  - bolder, lighter, bold, normal
    - Could specify as a multiple of 100 (100 – 900)
- font
  - For specifying a list of font properties

font: bolder 14pt Arial Helvetica

  - Order must be: style, weight, size, name(s)
- > SHOW fonts.html and display

- > SHOW fonts2.html and display
- The text-decoration property
    - line-through, overline, underline, none
    - letter-spacing – value is any length property value

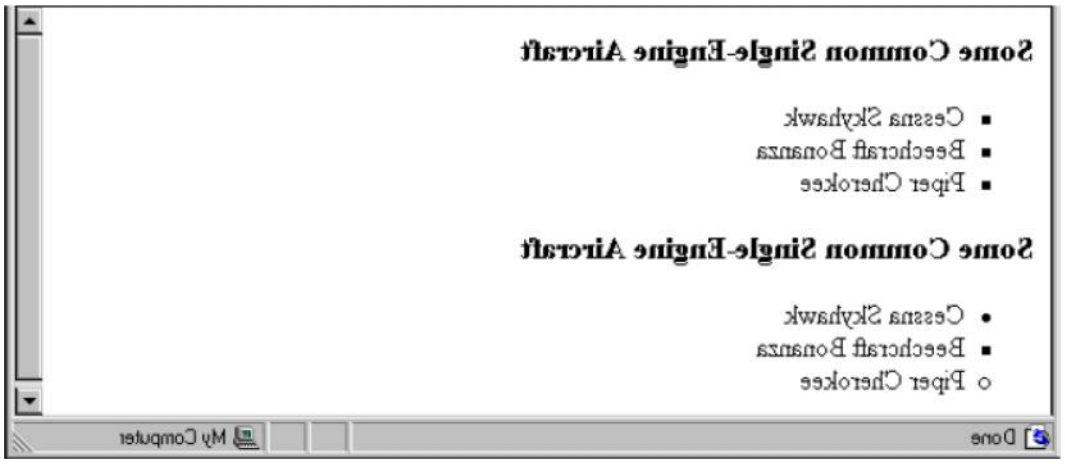
List properties

- list-style-type
- *Unordered lists*
    - Bullet can be a disc (default), a square, or a circle
    - Set it on either the <ul> or <li> tag
        - On <ul>, it applies to list items

```
<h3> Some Common Single-Engine Aircraft </h3>
 <ul style = "list-style-type: square">
    <li> Cessna Skyhawk </li>
    <li> Beechcraft Bonanza </li>
    <li> Piper Cherokee </li>
 </ul>
```

- On <li>, list-style-type applies to just that item

```
<h3> Some Common Single-Engine Aircraft </h3>
 <ul>
    <li style = "list-style-type: disc">
        Cessna Skyhawk </li>
    <li style = "list-style-type: square">
        Beechcraft Bonanza </li>
    <li style = "list-style-type: circle">
        Piper Cherokee </li>
 </ul>
```


Some Common Single-Engine Aircraft

- Cessna Skyhawk
- Beechcraft Bonanza
- Piper Cherokee

Some Common Single-Engine Aircraft

- Cessna Skyhawk
- Beechcraft Bonanza
- Piper Cherokee

- Could use an image for the bullets in an unordered list

- Example:

```
<li style = "list-style-image:
      url(bird.jpg)">
```

- *On ordered lists* - list-style-type can be used to change the sequence values

Property value	*Sequence type*	*First four*
Decimal	Arabic numerals	1, 2, 3, 4
upper-alpha	Uc letters	A, B, C, D
lower-alpha	Lc letters	a, b, c, d
upper-roman	Uc Roman	I, II, III, IV
lower-roman	Lc Roman	i, ii, iii, iv

→ SHOW sequence_types.html and display

- CSS2 has more, like lower-greek and hebrew

Colors

- *Color is a problem for the Web for two reasons*:
  1. Monitors vary widely
  2. Browsers vary widely

- There are three color collections
  1. There is a set of 16 colors that are guaranteed to be displayable by all graphical browsers on all color monitors

black	000000	green	008000
silver	C0C0C0	lime	00FF00
gray	808080	olive	808000
white	FFFFFF	yellow	FFFF00
maroon	800000	navy	000080
red	FF0000	blue	0000FF
purple	800080	teal	008080
fuchia	FF00FF	aqua	00FFFF

2. There is a much larger set, the Web Palette
   - 216 colors
   - Use hex color values of 00, 33, 66, 99, CC, and FF
   - Inside back cover of this book has them!

3. Any one of 16 million different colors

- The color property specifies the foreground color of elements

```
 <style type = “text/css”>
   th.red {color: red}
   th.orange {color: orange}
```

```
</style>
…
<table border = "5">
  <tr>
    <th class = "red"> Apple </th>
    <th class = "orange"> Orange </th>
    <th class = "orange"> Screwdriver </th>
  </tr>
</table>
```

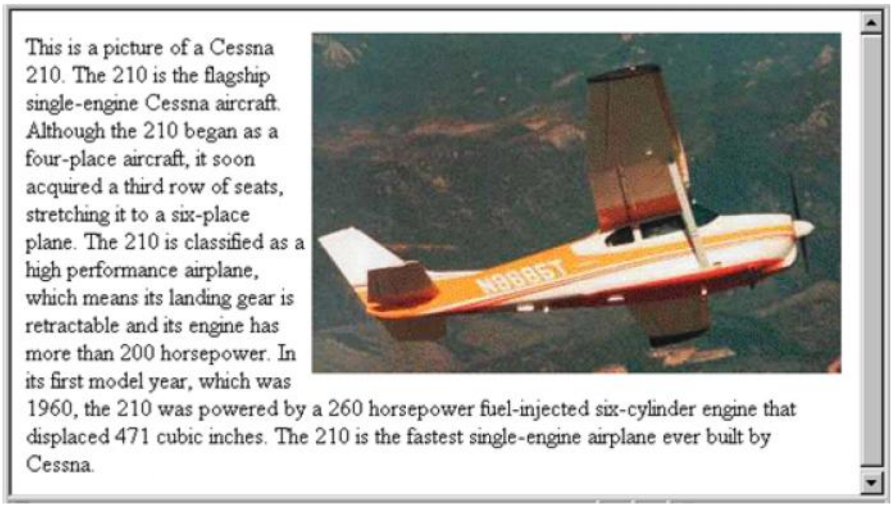
- ■ The background-color property specifies the background color of elements
- → SHOW back_color.html and display

Alignment of Text

- ■ The text-indent property allows indentation
  - ❑ Takes either a length or a % value
- ■ The text-align property has the possible values, left (the default), center, right, or justify
- ■ Sometimes we want text to flow around another element - the float property
  - ❑ The float property has the possible values, left, right, and none (the default)
  - ❑ If we have an element we want on the right, with text flowing on its left, we use the default text-align value (left) for the text and the right value for float on the element we want on the right

```
<img src = "c210.jpg"
      style = "float: right" />
```

- ❑ Some text with the default alignment - left



This is a picture of a Cessna 210. The 210 is the flagship single-engine Cessna aircraft. Although the 210 began as a four-place aircraft, it soon acquired a third row of seats, stretching it to a six-place plane. The 210 is classified as a high performance airplane, which means its landing gear is retractable and its engine has more than 200 horsepower. In its first model year, which was 1960, the 210 was powered by a 260 horsepower fuel-injected six-cylinder engine that displaced 471 cubic inches. The 210 is the fastest single-engine airplane ever built by Cessna.

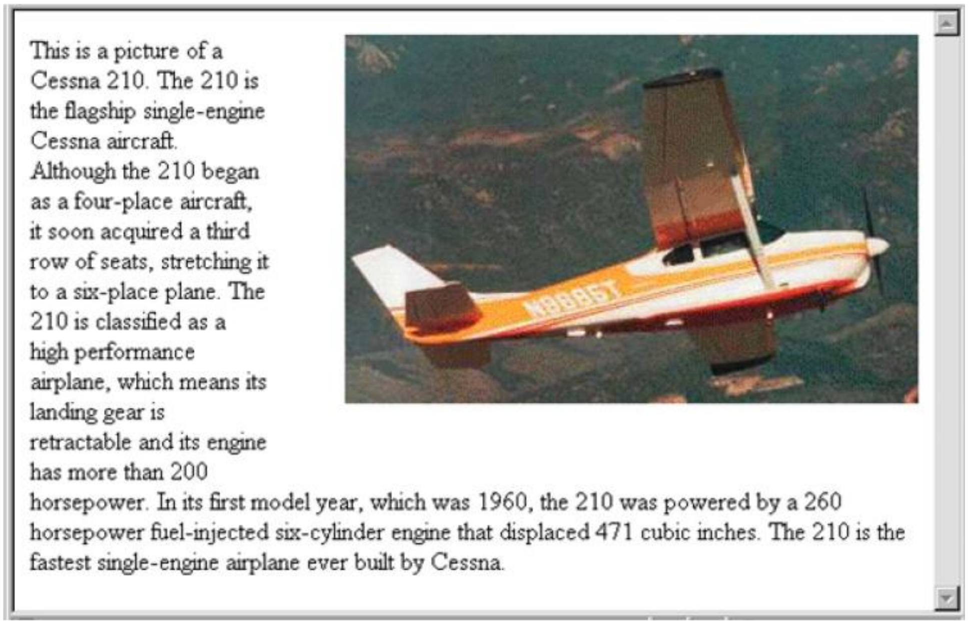
The Box Model

- Borders – every element has a border-style property
    - ❑ Controls whether the element has a border and if so, the style of the border
    - ❑ border-style values: none, dotted, dashed, and double
    - ❑ border-width – thin, medium (default), thick, or a length value in pixels
    - ❑ Border width can be specified for any of the four borders (e.g., border-top-width)
    - ❑ border-color – any color
    - ❑ Border color can be specified for any of the four borders (e.g., border-top-color)

→ SHOW borders.html and display

- Margin – the space between the border of an element and its neighbor element
- The margins around an element can be set with margin-left, etc. - just assign them a length value

```
<img src = "c210.jpg " style = "float: right;
  margin-left: 0.35in;
  margin-bottom: 0.35in" />
```


This is a picture of a Cessna 210. The 210 is the flagship single-engine Cessna aircraft. Although the 210 began as a four-place aircraft, it soon acquired a third row of seats, stretching it to a six-place plane. The 210 is classified as a high performance airplane, which means its landing gear is retractable and its engine has more than 200 horsepower. In its first model year, which was 1960, the 210 was powered by a 260 horsepower fuel-injected six-cylinder engine that displaced 471 cubic inches. The 210 is the fastest single-engine airplane ever built by Cessna.

- Padding – the distance between the content of an element and its border
    - Controlled by padding, padding-left, etc.

→ SHOW marpads.html and display

Background Images

- The background-image property

→ SHOW back_image.html and display

- Repetition can be controlled
    - background-repeat property
    - Possible values: repeat (default), no-repeat, repeat-x, or repeat-y
    - background-position property
        - Possible values: top, center, bottom, left, or right

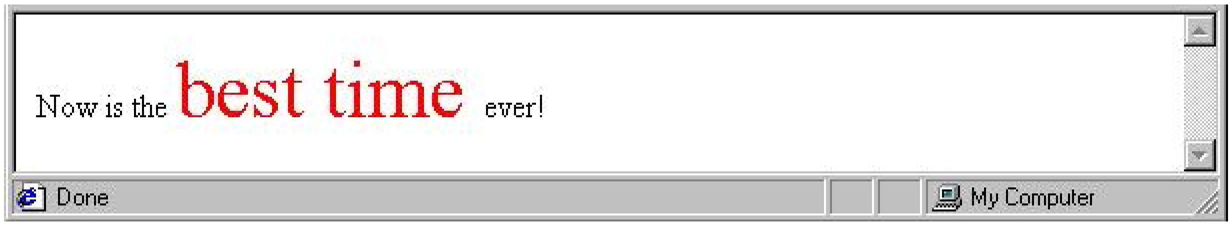
The <span> and <div> tags

- One problem with the font properties is that they apply to whole elements, which are often too large
    - Solution: a new tag to define an element in the content of a larger element - <span>
    - The default meaning of <span> is to leave the content as it is

```
<p>
Now is the <span> best time </span> ever!
</p>
```

- Use <span> to apply a document style sheet to its content

```
<style type = "text/css">?
    bigred {font-size: 24pt;
          font-family: Ariel; color: red}
  </style>
  <p>
    Now is the
      <span class = "bigred">
    best time </span> ever!
  </p>
```


Now is the best time ever!

- The <span> tag is similar to other HTML tags, they can be nested and
- they have id and class attributes
- Another tag that is useful for style specifications: <div>
    - Used to create document sections (or divisions) for which style can be specified
        - e.g., A section of five paragraphs for which you want some particular style

Conflict Resolution

- When two or more rules apply to the same tag there are rules for deciding which rule applies
- Document level
    - In-line style sheets have precedence over document style sheets
    - Document style sheets have precedence over external style sheets
- Within the same level there can be conflicts
    - A tag may be used twice as a selector
    - A tag may inherit a property and also be used as a selector
- Style sheets can have different sources
    - The author of a document may specify styles
    - The user, through browser settings, may specify styles
- Individual properties can be specified as important

Precedence Rules

- From highest to lowest

1. Important declarations with user origin
2. Important declarations with author origin
3. Normal declarations with author origin
4. Normal declarations with user origin
5. Any declarations with browser (or other user agent) origin

Tie-Breakers

- Specificity
    1. id selectors
    2. Class and pseudo-class selectors
    3. Contextual selectors
    4. General selectors
- Position
    1. Essentially, later has precedence over earlier

UNIT 3: JAVASCRIPT:

Syllabus:

Overview of Javascript; Object orientation and Javascript
General syntactic characteristics; Primitives,
operations, and expressions; Screen output and keyboard input;
Control statements; Object creation and modification; Arrays; Functions; Constructor;
Pattern matching using regular expressions; Errors in scripts;
Examples.

Basics of JavaScript

4.1 Overview of JavaScript: Origins

- Livescript
- Originally developed by Netscape
- Joint Development with Sun Microsystems in 1995
- Supported by Netscape, Mozilla, Internet Exploer

4.1 JavaScript Components

- Core
  - The heart of the language
- Client-side
  - Library of objects supporting browser control and user interaction EG: mouse clicks
- Server-side
  - Library of objects that support use in web servers
  - Eg: commun. With database management system

4.1 Java and JavaScript

- Differences
  - JavaScript has a different object model from Java
  - JavaScript is not strongly typed
- Java 1.6 has support for scripting
  - http://java.sun.com/javase/6/docs/technotes/guides/scripting/index.html
- Mozilla Rhino is an implementation of JavaScript in Java
  - http://www.mozilla.org/rhino/

4.1 Uses of JavaScript

- Provide alternative to server-side programming
  - Servers are often overloaded
  - Client processing has quicker reaction time
- JavaScript can work with forms

- JavaScript can interact with the internal model of the web page (Document Object Model)
- JavaScript is used to provide more complex user interface than plain forms with HTML/CSS can provide
    - http://www.protopage.com/ is an interesting example
    - A number of toolkits are available. Dojo, found at http://dojotoolkit.org/, is one example

4.1 Event-Driven Computation

- Users actions, such as mouse clicks and key presses, are referred to as *events*
- The main task of most JavaScript programs is to respond to events
- For example, a JavaScript program could validate data in a form before it is submitted to a server
    - *Caution:* It is important that crucial validation be done by the server. It is relatively easy to bypass client-side controls
    - For example, a user might create a copy of a web page but remove all the validation code.

4.1 XHTML/JavaScript Documents

- When JavaScript is embedded in an XHTML document, the browser must interpret it
- Two locations for JavaScript server different purposes
    - JavaScript in the head element will react to user input and be called from other locations
    - JavaScript in the body element will be executed once as the page is loaded
- Various strategies must be used to 'protect' the JavaScript from the browser
    - For example, comparisons present a problem since < and > are used to mark tags in XHTML
    - JavaScript code can be enclosed in XHTML comments
    - JavaScript code can be enclosed in a CDATA section

4.2 Object Orientation and JavaScript

- JavaScript is *object-based*
    - JavaScript defines objects that encapsulate both data and processing
    - However, JavaScript does not have true inheritance nor subtyping
- JavaScript provides *prototype-based inheritance*
    - See, for example this Wikipedia article for a discussion:

http://en.wikipedia.org/wiki/Prototype-based_languages

4.2 JavaScript Objects

- Objects are collections of *properties*
- Properties are either *data properties* or *method properties*
- Data properties are either primitive values or references to other objects
- Primitive values are often implemented directly in hardware
- The Object object is the ancestor of all objects in a JavaScript program

- ❑ Object has no data properties, but several method properties

4.3 JavaScript in XHTML

- ■ Directly embedded

<script type="text/javascript">
```
          <!--
                    …Javascript here…
          -->
</script>
```
- ❑ However, note that **a--** will not be allowed here!
- ■ Indirect reference

<script type="text/javascript" src="tst_number.js"/>

- ❑ This is the preferred approach

4.3 JavaScript in XHTML: CDATA

- ■ The **<![CDATA[…]]>** block is intended to hold data that should not be interpreted as XHTML
- ■ Using this should allow any data (including special symbols and --) to be included in the script
- ■ This, however does not work, at least in Firefox:

```
<script type="text/javascript">
          <![CDATA[
                    …JavaScript here…
          ]]>
</script>
```
- ■ The problem seems to be that the CDATA tag causes an internal JavaScript error
- ■ This does work in Firefox

```
<script type="text/javascript">
          /*<![CDATA[ */
                    …JavaScript here…
          /*]]> */
</script>
```
- ■ The comment symbols do not bother the XML parser (only /* and */ are 'visible' to it)
- ■ The comment symbols protect the CDATA markers from the JavaScript parser

4.3 General Syntactic Characteristics

- ■ Identifiers
  - ❑ Start with $, _, letter
  - ❑ Continue with $, _, letter or digit
  - ❑ Case sensitive
- ■ Reserved words
- ■ Comments
  - ❑ //

- ❑ /* … */

4.3 Statement Syntax

- Statements can be terminated with a semicolon
- However, the interpreter will insert the semicolon if missing at the end of a line and the statement seems to be complete
- Can be a problem:
  return
  x;
- If a statement must be continued to a new line, make sure that the first line does not make a complete statement by itself
- Example hello.html

4.4 Primitive Types

- Five primitive types
  - ❑ Number
  - ❑ String
  - ❑ Boolean
  - ❑ Undefined
  - ❑ Null
- There are five classes corresponding to the five primitive types
  - ❑ Wrapper objects for primitive values
  - ❑ Place for methods and properties relevant to the primitive types
  - ❑ Primitive values are *coerced* to the wrapper class as necessary, and vice-versa

4.4 Primitive and Object Storage

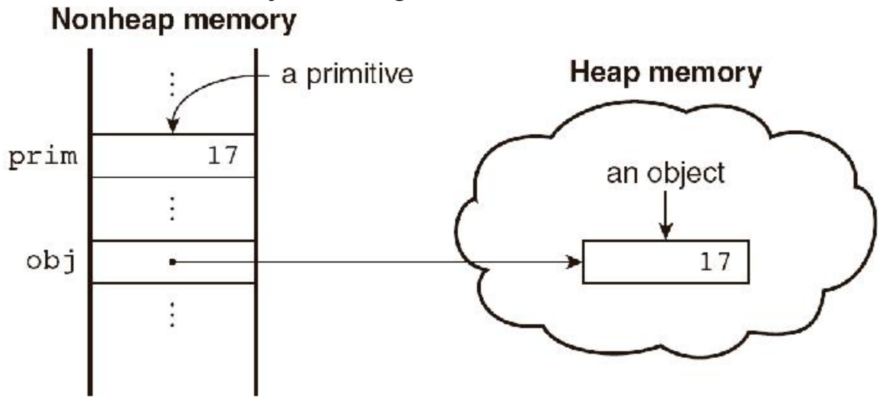


Figure 4.1 Primitives and objects

4.4 Numeric and String Literals

- Number values are represented internally as double-precision floating-point values
  - ❑ Number literals can be either integer or float
  - ❑ Float values may have a decimal and/or and exponent

- A String literal is delimited by either single or double quotes
  - There is no difference between single and double quotes
  - Certain characters may be *escaped* in strings
    - \' or \" to use a quote in a string delimited by the same quotes
    - \\\\ to use a literal backspace
  - The empty string '' or "" has no characters

4.4 Other Primitive Types

- Null
  - A single value, null
  - null is a reserved word
  - A variable that is used but has not been declared nor been assigned a value has a null value
  - Using a null value usually causes an error
- Undefined
  - A single value, undefined
  - However, undefined is not, itself, a reserved word
  - The value of a variable that is declared but not assigned a value
- Boolean
  - Two values: true and false

4.4 Declaring Variables

- JavaScript is *dynamically typed*, that is, variables do not have declared types
  - A variable can hold different types of values at different times during program execution
- A variable is declared using the keyword var

var counter,
index,
pi = 3.14159265,
quarterback = "Elway",
stop_flag = true;

4.4 Numeric Operators

- Standard arithmetic
  - + * - / %
- Increment and decrement
  - -- ++
  - Increment and decrement differ in effect when used before and after a variable
  - Assume that a has the value 3, initially
  - (++a) * 3 has the value 24
  - (a++) * 3 has the value 27

a has the final value 8 in either case

4.4 Precedence of Operators

Operators	Associativity
++, --, unary -	**Right**
***, /, %**	**Left**
+, -	**Left**
>, <, >= ,<=	**Left**
==, !=	**Left**
===,!==	**Left**
&&	**Left**
||	**Left**
=, +=, -=, *=, /=, &&=, ||=, %=	**Right**

4.4 Example of Precedence
var a = 2,
b = 4,
c,
d;
c = 3 + a * b;
// * is first, so c is now 11 (not 24)
d = b / a / 2;
// / associates left, so d is now 1 (not 4)

4.4 The Math Object
- Provides a collection of properties and methods useful for Number values
- This includes the trigonometric functions such as sin and cos
- When used, the methods must be qualified, as in Math.sin(x)

4.4 The Number Object
- Properties
  - MAX_VALUE
  - MIN_VALUE
  - NaN
  - POSITIVE_INFINITY
  - NEGATIVE_INFINITY
  - PI
- Operations resulting in errors return NaN
  - Use isNaN(a) to test if a is NaN
- toString method converts a number to string

4.4 String Catenation
- The operation + is the string catenation operation

- In many cases, other types are automatically converted to string

4.4 Implicit Type Conversion

- JavaScript attempts to convert values in order to be able to perform operations
- "August " + 1977 causes the number to be converted to string and a concatenation to be performed
- 7 * "3" causes the string to be converted to a number and a multiplication to be performed
- null is converted to 0 in a numeric context, undefined to NaN
- 0 is interpreted as a Boolean false, all other numbers are interpreted a true
- The empty string is interpreted as a Boolean false, all other strings (including "0"!) as Boolean true
- undefined, Nan and null are all interpreted as Boolean false

4.4 Explicit Type Conversion

- Explicit conversion of string to number
    - Number(aString)
    - aString – 0
    - Number must begin the string and be followed by space or end of string
- parseInt and parseFloat convert the beginning of a string but do not cause an error if a non-space follows the numeric part

4.4 String Properties and Methods

- One property: length
    - Note to Java programmers, this is not a method!
- Character positions in strings begin at index 0

4.4.11 String Methods

Method	**Parameters**	**Result**
charAt	**A number**	**Returns the character in the String object that is at the specified position**
indexOf	**One-character string**	**Returns the position in the String object of the parameter**

substring	**Two numbers**	**Returns the substring of the String object from the first parameter position to the second**
toLowerCase	**None**	**Converts any uppercase letters in the string to lowercase**
toUpperCase	**None**	**Converts any lowercase letters in the string to uppercase**

4.4 The typeof Operator

- Returns "number" or "string" or "boolean" for primitive types
- Returns "object" for an object or null
- Two syntactic forms
  - typeof x
  - typeof(x)

4.4 Assignment Statements

- Plain assignment indicated by =
- Compound assignment with
  - += -= /= *= %= …
- a += 7 means the same as
- a = a + 7

4.4 The Date Object

- A Date object represents a *time stamp*, that is, a point in time
- A Date object is created with the new operator
  - var now= new Date();
  - This creates a Date object for the time at which it was created

4.4 The Date Object: Methods

toLocaleString	**A string of the Date information**
getDate	**The day of the month**
getMonth	**The month of the year, as a number in the range of 0 to 11**
getDay	**The day of the week, as a number in the range of 0 to 6**
getFullYear	**The year**

getTime	**The number of milliseconds since January 1, 1970**
getHours	**The number of the hour, as a number in the range of 0 to 23**
getMinutes	**The number of the minute, as a number in the range of 0 to 59**
getSeconds	**The number of the second, as a number in the range of 0 to 59**
getMilliseconds	**The number of the millisecond, as a number in the range of 0 to 999**

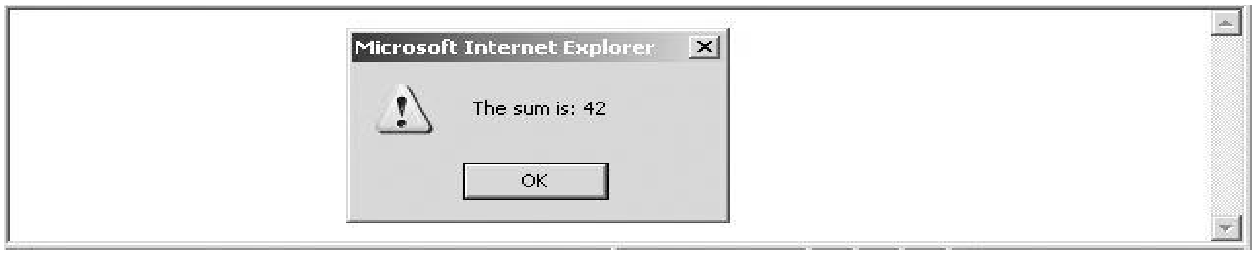
4.5 Window and Document

- The Window object represents the window in which the document containing the script is being displayed
- The Document object represents the document being displayed using DOM
- Window has two properties
  - window refers to the Window object itself
  - document refers to the Document object
- The Window object is the default object for JavaScript, so properties and methods of the Window object may be used without qualifying with the class name

4.5 Screen Output and Keyboard Input

- Standard output for JavaScript embedded in a browser is the window displaying the page in which the JavaScript is embedded
- The write method of the Document object write its parameters to the browser window
- The output is interpreted as HTML by the browser
- If a line break is needed in the output, interpolate <br/> into the output

  4.5 The alert Method
- The alert method opens a dialog box with a message
- The output of the alert is *not* XHTML, so use new lines rather than <br/>
- alert("The sum is:" + sum + "\n");

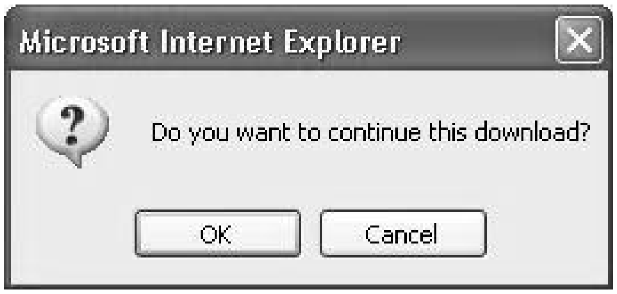


```
The sum is: 42
```

4.5 The confirm Method

- The confirm methods displays a message provided as a parameter
    - The confirm dialog has two buttons: OK and Cancel
- If the user presses OK, true is returned by the method
- If the user presses Cancel, false is returned

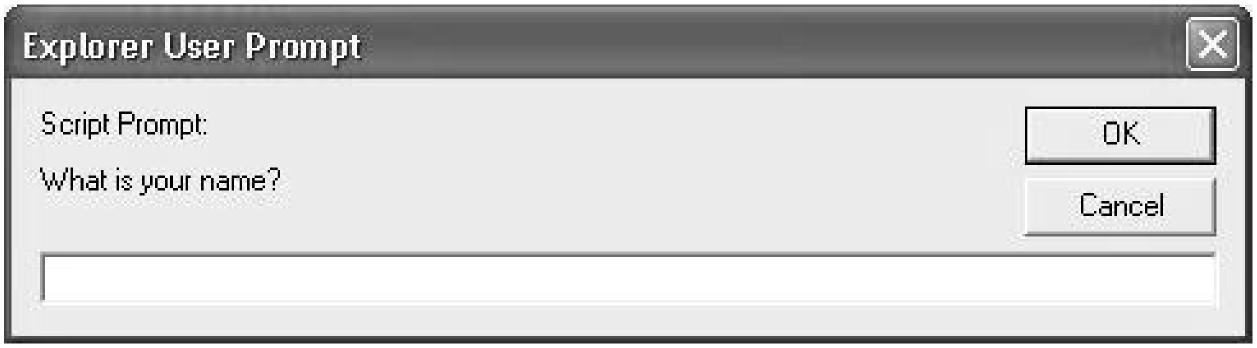
  var question =
    confirm("Do you want to continue this download?");



Do you want to continue this download?

4.5 The prompt Method

- This method displays its string argument in a dialog box
    - A second argument provides a default content for the user entry area
- The dialog box has an area for the user to enter text
- The method returns a String with the text entered by the user
- name = prompt("What is your name?", "");



Script Prompt:
What is your name?

4.5 Example of Input and Output

- roots.html

4.6 Control Statements

- A *compound statement* in JavaScript is a sequence of 0 or more statements enclosed in curly braces
    - Compound statements can be used as components of control statements allowing multiple statements to be used where, syntactically, a single statement is specified
- A *control construct* is a control statement including the statements or compound statements that it contains

4.6 Control Expressions

- A control expression has a Boolean value
    - An expression with a non-Boolean value used in a control statement will have its value converted to Boolean automatically
- Comparison operators
    - == != < <= > >=
    - === compares identity of values or objects
    - 3 == '3' is true due to automatic conversion
    - 3 === '3' is false
- Boolean operators
    - && || !
- Warning! A Boolean object evaluates as true
    - Unless the object is null or undefined

4.6 Selection Statements

- The if-then and if-then-else are similar to that in other programming languages, especially C/C++/Java

4.6 switch Statement Syntax

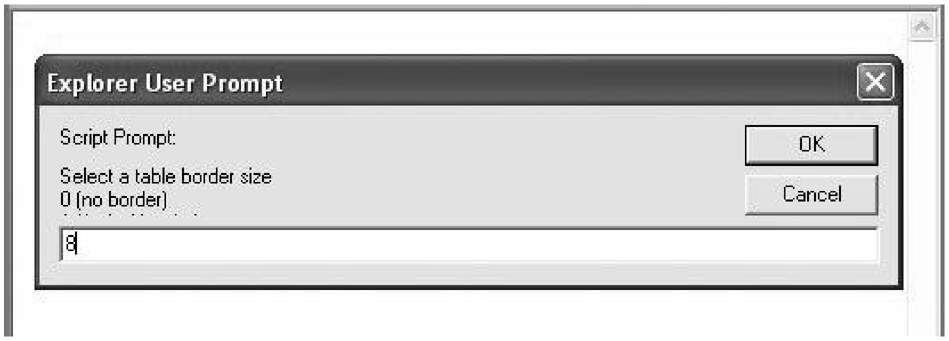
```
switch (expression) {
case value_1:
        // statement(s)
case value_2:
        // statement(s)
...
[default:
        // statement(s)]
}
```

4.6 switch Statement Semantics

- The expression is evaluated
- The value of the expressions is compared to the value in each case in turn
- If no case matches, execution begins at the default case
- Otherwise, execution continues with the statement following the case
- Execution continues until either the end of the switch is encountered or a break statement is executed

4.6 Example borders2.js

User Input Prompt



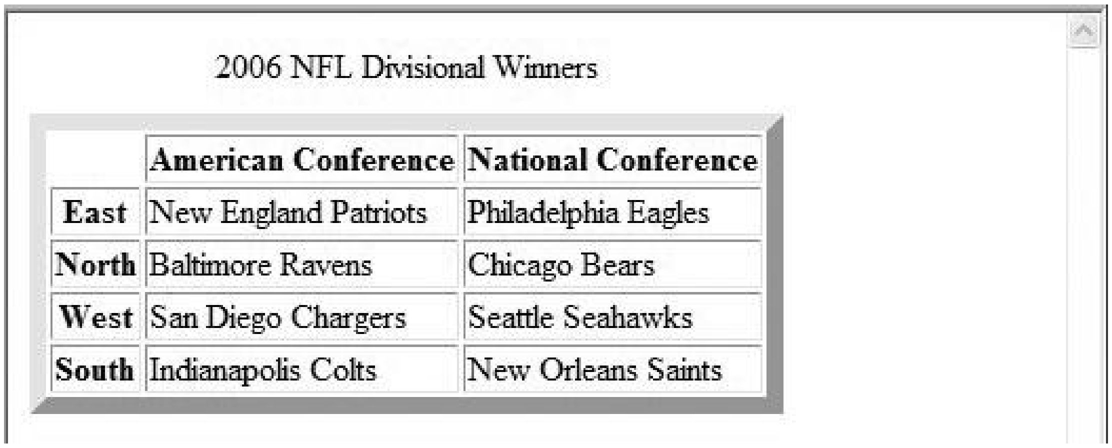
Explorer User Prompt

Script Prompt:

Select a table border size
0 (no border)

8

Results



2006 NFL Divisional Winners

	American Conference	National Conference
East	New England Patriots	Philadelphia Eagles
North	Baltimore Ravens	Chicago Bears
West	San Diego Chargers	Seattle Seahawks
South	Indianapolis Colts	New Orleans Saints

4.6 Loop Statements

- Loop statements in JavaScript are similar to those in C/C++/Java
- While

while (*control expression*)

  statement or compound statement

- For

for (*initial expression; control expression; increment expression*)

  statement or compound statement

- do/while

do *statement or compound statement*

while (*control expression*)

4.6 date.js Example

- Uses Date objects to time a calculation
- Displays the components of a Date object
- Illustrates a for loop

4.6 while Statement Semantics

- The control expression is evaluated
- If the control expression is true, then the statement is executed

- These two steps are repeated until the control expression becomes false
- At that point the while statement is finished

4.6 for Statement Semantics

- The initial expression is evaluated
- The control expression is evaluated
- If the control expression is true, the statement is executed
- Then the increment expression is evaluated
- The previous three steps are repeated as long as the control expression remains true
- When the control expression becomes false, the statement is finished executing

4.6 do/while Statement Semantics

- The statement is executed
- The control expression is evaluated
- If the control expression is true, the previous steps are repeated
- This continues until the control expression becomes false
- At that point, the statement execution is finished

4.7 Object Creation and Modification

- The new expression is used to create an object
    - This includes a call to a *constructor*
    - The new operator creates a blank object, the constructor creates and initializes all properties of the object
- Properties of an object are accessed using a dot notation: *object.property*
- Properties are not variables, so they are not declared
    - An object may be thought of as a Map/Dictionary/Associative-Storage
- The number of properties of an object may vary dynamically in JavaScript

4.7 Dynamic Properties

- Create my_car and add some properties

```
// Create an Object object
var my_car = new Object();
// Create and initialize the make property
my_car.make = "Ford";
// Create and initialize model
my_car.model = "Contour SVT";
```

- **The delete operator can be used to delete a property from an object**

```
delete my_car.model
```

4.7 The for-in Loop

- Syntax

for (*identifier* in *object*)
statement or compound statement

- **The loop lets the identifier take on each property in turn in the object**
- **Printing the properties in my_car:**

```
for (var prop in my_car)
        document.write("Name: ", prop, "; Value: ",
                my_car[prop], "<br />");
```

- **Result:**

Name: make; Value: Ford
Name: model; Value: Contour SVT

4.8 Arrays

- Arrays are lists of elements indexed by a numerical value
- Array indexes in JavaScript begin at 0
- Arrays can be modified in size even after they have been created

4.8 Array Object Creation

- Arrays can be created using the new Array method
    - new Array with one parameter creates an empty array of the specified number of elements
        - new Array(10)
    - new Array with two or more parameters creates an array with the specified parameters as elements
        - new Array(10, 20)
- Literal arrays can be specified using square brackets to include a list of elements
    - var alist = [1, “ii”, “gamma”, “4”];
- Elements of an array do not have to be of the same type

4.8 Characteristics of Array Objects

- The length of an array is one more than the highest index to which a value has been assigned or the initial size (using Array with one argument), whichever is larger
- Assignment to an index greater than or equal to the current length simply increases the length of the array
- Only assigned elements of an array occupy space
    - Suppose an array were created using new Array(200)
    - Suppose only elements 150 through 174 were assigned values
    - Only the 25 assigned elements would be allocated storage, the other 175 would not be allocated storage

4.8 Example insert_names.js

- This example shows the dynamic nature of arrays in JavaScript

4.8 Array Methods

- join
- reverse
- sort
- concat
- slice

4.8 Dynamic List Operations

- push
  - Add to the end
- pop
  - Remove from the end
- shift
  - Remove from the front
- unshift
  - Add to the front

4.8 Two-dimensional Arrays

- A two-dimensional array in JavaScript is an array of arrays
  - This need not even be rectangular shaped: different rows could have different length
- Example nested_arrays.js illustrates two-dimensional arrays

4.9 Function Fundamentals

- Function definition syntax
  - A function definition consist of a header followed by a compound statement
  - A function header:
    - function *function-name*(*optional-formal-parameters*)
- return statements
  - A return statement causes a function to cease execution and control to pass to the caller
  - A return statement may include a value which is sent back to the caller
    - This value may be used in an expression by the caller
  - A return statement without a value implicitly returns undefined
- Function call syntax
  - Function name followed by parentheses and any actual parameters
  - Function call may be used as an expression or part of an expression
- Functions must defined before use in the page header

4.9 Functions are Objects

- Functions are objects in JavaScript
- Functions may, therefore, be assigned to variables and to object properties
  - Object properties that have function values are methods of the object
- Example

```
function fun() {
        document.write(       "This surely is fun! <br/>");
}
        ref_fun = fun; // Now, ref_fun refers to the fun object
        fun(); // A call to fun
        ref_fun(); // Also a call to fun
```

4.9 Local Variables

- "The *scope* of a variable is the range of statements over which it is visible"
- A variable not declared using var has global scope, visible throughout the page, even if used inside a function definition
- A variable declared with var outside a function definition has global scope
- A variable declared with var inside a function definition has local scope, visible only inside the function definition
    - If a global variable has the same name, it is hidden inside the function definition

4.9 Parameters

- Parameters named in a function header are called *formal parameters*
- Parameters used in a function call are called *actual parameters*
- Parameters are passed by value
    - For an object parameter, the reference is passed, so the function body can actually change the object
    - However, an assignment to the formal parameter will not change the actual parameter

4.9 Parameter Passing Example

```
function fun1(my_list) {
        var list2 = new Array(1, 3, 5);
        my_list[3] = 14;
        ...
        my_list = list2;
        ...
}
...
var list = new Array(2, 4, 6, 8)
fun1(list);
```

- The first assignment changes list in the caller
- The second assignment has no effect on the list object in the caller
- Pass by reference can be simulated by passing an array containing the value

4.9 Parameter Checking

- JavaScript checks neither the type nor number of parameters in a function call
    - Formal parameters have no type specified
    - Extra actual parameters are ignored (however, see below)
    - If there are fewer actual parameters than formal parameters, the extra formal parameters remain undefined
- This is typical of scripting languages
- A property array named arguments holds all of the actual parameters, whether or not there are more of them than there are formal parameters
    - Example params.js illustrates this

4.9 The sort Method, Revisited

- A parameter can be passed to the sort method to specify how to sort elements in an array
    - The parameter is a function that takes two parameters
    - The function returns a negative value to indicate the first parameter should come before the second
    - The function returns a positive value to indicate the first parameter should come after the second
    - The function returns 0 to indicate the first parameter and the second parameter are equivalent as far as the ordering is concerned

Example median.js illustrates the sort method

4.11 Constructors

- Constructors are functions that create an initialize properties for new objects
- A constructor uses the keyword this in the body to reference the object being initialized
- Object methods are properties that refer to functions
    - A function to be used as a method may use the keyword this to refer to the object for which it is acting
- Example car_constructor.html

4.12 Using Regular Expressions

- Regular expressions are used to specify patterns in strings
- JavaScript provides two methods to use regular expressions in pattern matching
    - String methods
    - RegExp objects (not covered in the text)
- A literal regular expression pattern is indicated by enclosing the pattern in slashes
- The search method returns the position of a match, if found, or -1 if no match was found

4.12 Example Using search

```
var str = "Rabbits are furry";
var position = str.search(/bits/);
if (position > 0)
        document.write("'bits' appears in position",
                position, "<br />");
else
        document.write(
                "'bits' does not appear in str <br />");
```

- This uses a pattern that matches the string ‘bits’
- The output of this code is as follows:

```
'bits' appears in position 3
```

4.12 Characters and Character-Classes

- *Metacharacters* have special meaning in regular expressions
    - \ | ( ) [ ] { } ^ $ * + ? .

    - These characters may be used literally by escaping them with \
- Other characters represent themselves
- A period matches any single character
    - /f.r/ matches for and far and fir but not fr
- A character class matches one of a specified set of characters
    - [*character set*]
    - List characters individually: [abcdef]
    - Give a range of characters: [a-z]
    - Beware of [A-z]
    - ^ at the beginning negates the class

4.12 Predefined character classes

Name	Equivalent Pattern	Matches
\d	**[0-9]**	**A digit**
\D	**[^0-9]**	**Not a digit**
\w	**[A-Za-z_0-9]**	**A word character (alphanumeric)**
\W	**[^A-Za-z_0-9]**	**Not a word character**
\s	**[\r\t\n\f]**	**A whitespace character**
\S	**[^ \r\t\n\f]**	**Not a whitespace character**

4.12 Repeated Matches

- A pattern can be repeated a fixed number of times by following it with a pair of curly braces enclosing a count
- A pattern can be repeated by following it with one of the following special characters
    - * indicates zero or more repetitions of the previous pattern
    - + indicates one or more of the previous pattern
    - ? indicates zero or one of the previous pattern
- Examples
    - /\(\d{3}\)\d{3}-\d{4}/ might represent a telephone number

    - /[$_a-zA-Z][$_a-zA-Z0-9]*/ matches identifiers

4.12 Anchors

- Anchors in regular expressions match positions rather than characters
    - Anchors are 0 width and may not take multiplicity modifiers
- Anchoring to the end of a string
    - ^ at the beginning of a pattern matches the beginning of a string
    - $ at the end of a pattern matches the end of a string
        - The $ in /a$b/ matches a $ character
- Anchoring at a word boundary
    - \b matches the position between a word character and a non-word character or the beginning or the end of a string
    - /\bthe\b/ will match 'the' but not 'theatre' and will also match 'the' in the string 'one of the best'

4.12 Pattern Modifiers

- Pattern modifiers are specified by characters that follow the closing / of a pattern
- Modifiers modify the way a pattern is interpreted or used
- The x modifier causes whitespace in the pattern to be ignored
    - This allows better formatting of the pattern
    - \s still retains its meaning
- The g modifier is explained in the following

4.12 Other Pattern Matching Methods

- The replace method takes a pattern parameter and a string parameter
    - The method replaces a match of the pattern in the target string with the second parameter
    - A g modifier on the pattern causes multiple replacements
- Parentheses can be used in patterns to mark sub-patterns
    - The pattern matching machinery will remember the parts of a matched string that correspond to sub-patterns
- The match method takes one pattern parameter
    - Without a g modifier, the return is an array of the match and parameterized sub-matches
    - With a g modifier, the return is an array of all matches
- The split method splits the object string using the pattern to specify the split points

4.13 An Example

- forms_check.js
- Using javascript to check the validity of input data
- Note, a server program may need to check the data sent to it since the validation can be bypassed in a number of ways

4.14 Errors in Scripts

- JavaScript errors are detected by the browser

- Different browsers report this differently
    - Firefox uses a special console
- Support for debugging is provided
    - In IE 7, the debugger is part of the browser
    - For Firefox 2, plug-ins are available
        - These include Venkman and Firebug

UNIT 4: JAVASCRIPT AND HTML DOCUMENTS SYLLABUS

The Javascript execution environment; The Document Object Model; Element access in Javascript;
Events and event handling; Handling events from the Body elements,
Button elements, Text box and Password elements; The DOM 2 event model
The navigator object; DOM tree traversal and modification.
Introduction to dynamic documents, Positioning elements, Moving elements, Element visibility, Changing colors and fonts,
Dynamic content, Stacking elements, Locating the mouse cursor,
Reacting to a mouse click, Slow movement of elements, Dragging and dropping elements.

Why is JavaScript Important?

It is simple and lots of scripts available in public domain and easy to use. It is used for client-side scripting. It is important for web because it can promptly validate user input. It can update the web page without post back to server. It allows page to react to user actions other than pushing a “submit” button more interactively.

It increases Server efficiency. Since the client processes the script, which saves *internet time* and *server time*. Web pages can contain JavaScript programs executed inside the browser.

It is supported by all major browsers. User may disable JavaScript due to security reasons.

What is JavaScript?

An *interpreted* scripting language for web. It was introduced by N*etscape* with Netscape 2.0 in 1995. Microsoft version is called Jscript . Language is casesensitive.

It can be embedded within HTML tags or separately. Tightly integrated with browser. It can handle many types of events generated by the normal interaction between user and browser.

When not to use JavaScript?

When you need to access other resources such as:

- Files
- Programs
- Databases.

When you are using sensitive or copyrighted data or algorithms.

Java versus JavaScript

Java supports OOP. JavaScript does not support OO software development paradigm
Java is strongly-typed. JavaScript is dynamically-typed.
JavaScript syntax is same as java. Constructs like expressions, assign statements and control statements are same as in Java.

Where to place JavaScript code?

1. Inline JavaScript
   It can be placed on the event handler attributes of input controls of a form.

2. Internal / Embedded JavaScript
It can be enclosed in script tags either in the header or the document body.
3. External JavaScript
   It can be placed in a separate file. Enclosed in script tags in the header with 'src' attribute. This method is good for maintenance due to modularization.

How is JavaScript executed?
It is almost always executed at client-side. Script tags in the header
are executed in
the order in which they appear, before any of the body is processed.

Script tags in the body are executed as the body is rendered.

Script in the event handlers are executed whenever the appropriate event occurs.
Script in functions executed triggered by browser events.
- built-in java script
- user-defined functions (declared in the header)

All properties are visible to all scripts (no information hiding). Global variables are properties of the Window object. There can be more than one Window object. Global variables depend on which Window is the context. The Document object represents the document displayed.

Why DOM?

Portability is the major issue while using the JavaScript. The standard DOM has to provide a specification that would make Java programs and JavaScript scripts that deal with XHTML documents portable among various browsers.

DOM Levels

• DOM 0: informal, early browsers
• DOM 1: XHTML/XML structure
• DOM 2: event model, style interface, traversal
• DOM 3: content model, validation

DOM specifications describe an abstract model of a document.

• Interfaces describe methods and properties
• The interfaces describe a tree structure
• Different languages will *bind* the interfaces to specific implementations. The internal representation may not be tree-like. In JavaScript, data are represented as properties and operations as methods. Nodes of the tree will be JavaScript objects.

Attributes of elements become named properties of element node objects.

• <input type="text" name="address">

The object representing this node will have two properties. Type property will have value "text" and name property will have value "address".

The following XHTML document and its corresponding DOM tree illustrate the relationship between them.

```
 <html xmlns = http://www.w3.org/1999/xhtml>
 <head> <title> A simple document</title>
 </head>
 <body>
 <table>
 <tr>
<th> Breakfast</th>
<td>0 </td>
<td>1 </td>
</tr>
<tr>
<th> Lunch</th>
<td>1 </td >
<td> 0 </td>
</tr>
```

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

Elements in XHTML document correspond to objects in JavaScript. Objects can be addressed in several ways:
1. forms and elements array defined in DOM 0.
• Individual elements are specified by index.
• The index may change when the form changes.
2. Using the name attributes for form and form elements
• A name on the form element causes validation problems.
• Names are required on form elements providing data to the server.
3. Using getElementById with id attributes.
• id attribute value must be unique for an element.
Consider this simple form:

```
<form action = "">
 <input type ="button“ name = “pushMe">
  </form>
```

The input element can be referenced as document.forms[0].element[0].

• All elements from the reference element up to, but not including, the body must have a name attribute.
• This violates XHTML standards in some cases.
Example:

```
<form name = "myForm"  action = "">
<input type ="button“ name="pushMe">
</form>
```

• Referencing the input
document.myForm.pushMe

• Set the id attribute of the input element

```
<form action = "">
  <input type="button"  id=“turnItOn">
</form>
```

• Then use getElementById
document.getElementById(“turnItOn")

Event-driven Programming involves execution based on user's action . (ex: User's selection in a web page.)

Event-driven Programming: Instead of user synchronizing with the program, the program synchronizes with, or reacts to, the user. All communication from user occurs via events. An event is an action that happens in the system. For example,

- A mouse button pressed or released
- A keyboard key is hit
- A window is moved, resized, closed, etc.

Event-driven Programming: Typically two different classes of events.

1. User-initiated events
   Events that result directly from a user action.
   e.g., mouse click, button press, move mouse.
2. System-initiated events
   Events created by the system, as it responds to a user action

e.g., scrolling text, re-drawing a window.

There is no top-down flow of control, i.e., Main program defining the sequential. Flow. Code fragments are associated with events and invoked when events occur.
The order of execution is decoupled with the code. Don't have to deal with order of events. This is especially helpful when the order is unknown!
Event-driven programming is a style of programming in which pieces of code, *event handlers*, are written to be activated when certain *events* occur.

Events represent activity in the environment including, especially, user actions such as moving the mouse or typing on the keyboard.

An *event handler* is a program segment designed to execute when a certain event occurs. Events are represented by JavaScript objects. *Registration* is the activity of connecting a script to a type of event. Assign an event attribute an event handler.
Assign a DOM node an event handler.

Event	*Tag Attribute*
blur	onblur
change	onchange
click	onclick
focus	onfocus
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout
mouseover	onmouseover

mouseup	onmouseup
select	onselect
submit	onsubmit
unload	onunload

Particular events are associated to certain attributes. The attribute for one kind of event may appear on different tags allowing the program to react to events affecting different components. A text element gets focus in three ways:

• When the user puts the mouse cursor over it and presses the left button
• When the user tabs to the element
• By executing the focus method

Losing the focus is *blur*ring.

Using an attribute, a JavaScript command can be specified:

```
<input type="button" name="myButton"
 onclick= "alert('You clicked the button!')"/>
```

A function call can be used if the handler is longer than a single statement.

```
<input type="button" name="myButton"
 onclick="myHandler()"/>
```

An event can be registered for this tag in two ways:

```
<input type="button" name="freeOffer"
id="freeButton"/>
```

• Using an event attribute

```
<input type="button" name="freeOffer"
id="freeButton"
onclick="freeButtonHandler()"/>
```

• Assigning to a property of the element node

```
document.getElementById("freeButton").onclick =  freeButtonHandler
```

– Note that the function name, a reference to the function, is assigned.
– Writing freeButtonHandler() would assign the return value of the function call as the handler (possible, but unlikely).

Note that the no parameters are passed to the function when called by the JavaScript System. The handler code must identify the element that caused the call.

The handler call can be enclosed in an anonymous function
– dom.elements[0].onclick = function() {planeChoice(152)};

Assigning to an attribute is more flexible, allowing passing parameters without having to create an anonymous function. Assigning to a node property helps separate HTML and code. Assigning to a node property allows reassignment later if the handler needs to be changed.

By manipulating the focus event the user can be prevented from changing the amount in a text input field.
This is possible to work around:
– Copy the page but leave out the validation code.
– Simulate an HTTP request directly with socket-level programming.
– If the validity of data is important, the server needs to check it.

Validating data using JavaScript provides quicker interaction for the user. Validity checking on the server requires a round-trip for the server to check the data and then to respond with an appropriate error page. Handling a data validity error:
– Put the focus in the field in question
– Highlight the text for easier editing
If an event handler returns false, default actions are not taken by the browser. This can be used in a Submit button event handler to check validity and not submit if there are problems.
The name is First, Last, Middle-Initial, each part capitalized.
– /^[A-Z][a-z]+, ?[A-Z][a-z]+, ?[A-Z]\.?$/
The phone is ddd-ddd-dddd where d is a digit.
– /^\d{3}-\d{3}-\d{4}$/
Each pattern uses the ^ and $ anchors to make sure the entire string matches.

DOM 2 is defined in modules.
The Events module defines several sub modules.
– HTML Events and Mouse Events are common.
An event object is passed as a parameter to an event handler.
– Properties of this object provide information about the event.
– Some event types will extend the interface to include information relevant to the subtype. For example, a mouse event will include the location of the mouse at the time of the event.

DOM 2 defines a process for determining which handlers to execute for a particular event.
The process has three phases:
The event object representing the event is created at a particular node called the *target node.*

In the *capturing phase* each node from the document root to the target node, in order, is examined.

– If the node is not the target node and there is a handler for that event at the node and the handler is enabled for capture for the node, the handler is executed.
Then all handlers registered for the target node, if any, are executed.
In the *bubbling phase* each node from the parent of the target node to the root node, in order, is examined.
– If there is a handler for that event at the node and the handler is *not* enabled for capture for the node, the handler is executed.
– Some event types are not allowed to bubble: load, unload, blur and focus among the HTML event types.

As each handler is executed, properties of the event provide context.
– The currentTarget property is the node to which the handler is registered.
– The target property is the node to which the event was originally directed.
One major advantage of this scheme over DOM 0 is that event handling can be centralized in an ancestor node.
For example, a calculator keyboard will have a number of digit buttons.
– In some GUI frameworks, a handler must be added to each button separately.
– In DOM 2, the buttons could be organized under a single node and the handler placed on the node.

As each handler is executed, properties of the event provide context.
– The currentTarget property is the node to which the handler is registered
– The target property is the node to which the event was originally directed Handlers are called *listeners* in DOM 2. addEventListener is used to register a handler, it takes three parameters.
– A string naming the event type,
– The handler,
– A Boolean specifying whether the handler is enabled for the capture phase or not.

The navigator object indicates which browser is being used to view the XHTML document. The appName property of the navigator object gives the name of the browser. The appVersion property of the navigator object gives the browser version.
Each element in an XHTML document has a corresponding ELEMENT object in the DOM representation. The ELEMENT object has methods to support.
– Traversing the document, that is, visiting each of the document nodes.
– Modifying the document.

For example, removing and inserting child nodes. Various properties of Element objects are related nodes. parentNode references the parent node of the Element. previousSibling and nextSibling connect the children of a node into a list. firstChild and lastChild reference children of an Element. These would be text nodes or element nodes contained in the Element.

The insertBefore(newChild, refChild) method places the newChild node before the refChild node.
The replaceChild(newChild, oldChild)method places the oldChild node with the newChild node.

The removeChild(oldChild) method removes oldChild node from the DOM structure.
The appendChild(newChild) method adds the given node to the end of the list of the siblings of the node through which it is called.

The highest levels of the execution environment of client-side JavaScript are represented with the *Window* and *Document* objects.

The Document object includes a forms array property, which includes references to all forms in the document. Each element of the forms array has an elements array, which includes references to all elements in the form.

The DOM is an abstract interface whose purpose is to provide language independent way to access the elements of an XHTML document.

XHTML tags are represented in JavaScript as objects, tag attributes are represented as properties.

Introdcution

Dynamic Documents wih JavaScript

Dynamic XHTML is not a technology in and of itself, but rather is a combination of three echnologies: XHTML, Cascading Style Sheets (CSS) and Scripting. It usually involves using scripting like JavaScript to change tag attribute, tag contents or CSS property of an XHTML element . In modern browsers, most CSS properties can be modified dynamically ie changes can be made after the document has been and is still being displayed . This can be done by changing an individual style of element (using the style property of the element) or by changing the class name assigned to the element (using the className property).

Advantages of Dynamic XHTML

(1) Dynamic XHTML makes documents dynamic. Dynamic documents :

o Allow the designer to control how the HTML displays Web pages' content.

o React and change with the actions of the visitor.

o Can exactly position any element in the window, and change that position after the document has loaded.

o Can hide and show content as needed.

(2) Dynamic XHTML allows any HTML element (any object on the screen that can be controlled independently using JavaScript) to be manipulated at any time, turning plain HTML into dynamic HTML

(3) With DHTML, changes occur entirely on the client-side

(4) Using DHTML gives the author more control over how the page is formatted and how content is positioned on the page..

Positioning Elements

Cascading Style Sheets (CSS) Positioning defines the placement of elements on a page and is an extension of cascading style sheets as specified in the W3C on Positioning HTML with CSS. By default, elements flow one after another in the same order as they appear in the HTML source, with each element having a size and position that depends on the type of element, the contents of the element, and the display context for the element as it will render on the page. This default flow model for HTML layout doesn't allow a high level of control over the placement of elements on the page. By applying a small set of CSS attributes to the elements that are defined for the page, CSS can control the precise position of elements by giving exact coordinates. It is also possible to specify placement relative to the position of other objects on the page.

Just like any other HTML or CSS attribute, the CSS attributes used to control an element's position are available for scripting. The position of these elements on the page can thus be dynamically changed with script. As a result, the position of these elements can be recalculated and redrawn after the document is loaded without reloading the page from the server. It usually involves using JavaScript to change a positioning style properties of an HTML elements. position, top, left are the three properties that dictate the position of the elements. position specifies the reference point for the placement of the elements. top *and* left specify the distance from top and left of reference point where element is to appear. absolute, relative **and** static are the three possible values for the position property.

Absolute Positioning

A element can be placed at specific position in the document using absolute value for the position styling property. Absolute positioning defines the x and y coordinates of an element with reference to the top left corner of the browser page or the containing block and the position attribute is set to absolute. With absolute positioning elements are placed without regard to the positions of other elements. For example, if you want place an image 100 pixels from the top and 100 pixels from the left of the document display window, it can be placed as following statements:

```
 <img  src="earth.jpg" style="position:absolute; left:100px; top:100px" />
```

Use of Absolute positioning

• Places elements at specific position in the document display.

• Can be used superimpose text over the ordinary text to create effect similar that watermark on page.

The following example illustrates the usage of absolute positioning to position five elements at specific different positions in the document display.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<! - - abpos.html - - >
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Absolute positioning </title>
    <style type = "text/css">
     .s1 {position: absolute; top: 45px; left: 50px; }
     .s2 {position: absolute; top: 45px; left: 300px; }
     .s3 {position: absolute; top: 175px; left: 50px; }
     .s4 {position: absolute; top: 175px; left: 300px; }
     .s5 {position: absolute; top: 100px; left: 175px; }
    </style>
  </head>
  <body>
  <p>Positioning 5 instances same image at 5 different positions
   <img class="s1" src="smiley.gif" />
   <img class="s2" src="smiley.gif" />
   <img class="s3" src="smiley.gif" />
   <img class="s4" src="smiley.gif" />
   <img class="s5" src="smiley.gif" />
  </p>
  </body>  </html>
```

Output

When an element is absolutely positioned inside another positioned element, the top and left property values are measured from the upper-left corner of the enclosing element.

The following example illustrate the nested element placement. When an element is absolutely positioned inside another positioned element, the top and left property values are measured from the upper-left corner of the enclosing element.
The following example illustrate the nested element placement
Output

```
Hello. And now it's time to say goodbye
    Hello, again
!!!!!.
```

In the above example we insert a span element inside the div element. Positioning attributes for the span element place it 10 pixels in from the left and 30 pixels
down from the top of its positioning context—the div element in this case.

Relative Positioning

Relative positioning means an element is placed relative to its natural position in the document's flow. When you use relative positioning, an element is positioned relative to where it would regularly be. If the top and left properties are given, then relative positioning displace the element by the specified amount from the natural position.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">
< -- nestedap.html -->
<head>
<body>
<div id="someDiv" style="position:absolute; left:100px; top:50px">
Hello.
<span id="someSpan" style="position:absolute; left:10px; top:30px">
Hello, again!!!!!.
</span>
And now it's time to say goodbye.
</div>
```

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

The top property defines how far from the top of its *usual position* we want the top of element to appear. If we use a positive value, then our element is moved down from the usual position, whereas a negative value would move our element up from the usual position.

The left property defines how far from its usual position we want the left of our element to appear. Positive values will move the element right, and negative values will move it left the usual position.

If the top and left properties are not specified then element is positioned as if like the it is statically positioned. However, such an element can be moved later.

Relative positioning is used for creating different effects in the document. It can be used to highlight the special words in the text. The following example highlight the word "red" in line of text.

Output

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<!—relative.html -- >
<html xmlns = "http://www.w3.org/1999/xhtml">
   <body style = "font-family: Times; font-size: 24pt;">
   <p>
     Roses are <span style =
             "position: relative; top: 10px;
              font-family: Times; font-size: 48pt;
              font-style: italic; color: red;">
     red </span> in color.
   </p>
  </body>
```

```
</html>
```

Roses are *red* in color.

Relative positioning can be used to create superscripts. For example the following can be used to place “xyz” 10 pixels above the natural baseline of the text.

```
<p> The superscript in this name<span style=”position: relative;
        top:-3px” > xyz </span> is “xyz”.</p>
```

Static Positioning

'Static' positioning is identical to normally rendered HTML. These elements cannot be positioned or repositioned, nor do they define a coordinate system for child elements. This is the default value for 'position', except for the <BODY> element, which, while it cannot be positioned, does define a coordinate system for child elements.

Moving Elements

Dynamic movement of 'relative'ly /’absolute’ly positioned elements can provide animation effects in scripting environments. Element position is going to be changed by modifying the left and top properties of the element's style property. If position is set to absolute, the element moves to the new values of top and left, if its position is set to relative, it moves from its original position by distances given by the new values of top and left. The following example demonstrates this

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">
<! – move.html -- >
<title>Position</title>
<style type="text/css">
#divBlock {
 position:relative;
 height:100px;
 width:100px;
 top:100px;
 left:100px;
 background-color:red;
}</style>
<script type="text/javascript">
function init(){
 document.getElementById("divBlock").style.top = "100px";
 document.getElementById("divBlock").style.left = "100px";
}
function moveH(elem, distance){
 var objElem = document.getElementById(elem);
 var curLeft = parseInt(objElem.style.left);
 objElem.style.left = (curLeft + distance) + "px";
}
function moveV(elem, distance){
 var objElem = document.getElementById(elem);
 var curTop = parseInt(objElem.style.top);
 objElem.style.top = (curTop + distance) + "px";
}
</script>
</head>
<body onload="init();">
<form>
<input type="button" value="Left" onclick="moveH('divBlock',-10);">
<input type="button" value="Right" onclick="moveH('divBlock',10);">
<input type="button" value="Up" onclick="moveV('divBlock',-10);">
<input type="button" value="Down" onclick="moveV('divBlock',10);">
</form>
<div id="divBlock"></div>
</body>
</html>
 26
```

Output

In the output user can move the square in the 4 different directions by clicking the appropriate button.

The init() function set the top and left properties of the divBlock div, thus making the properties accessible to JavaScript.

The moveH() function uses parseInt() to cut off the units (e.g, px) from the value of the left property of the div and assign the resulting integer to the curLeft variable. It then modifies the left property of the element by adding the value passed in for distance.

The moveV() function does the same thing, but it modifies the top property rather than the left property.The functions are triggered with onclick event handlers.

Element Visibility

Document elements can be specified to be visible or hidden with the values if their visibility property. The two possible values for the visibility are – visible and hidden.

The following example displays the 4 table elements and allows the user to toggle each table element causing the element to appear and disappear in the document display.

Output

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title>Showing and Hiding Elements with JavaScript</title>
</head>
<!—vis.html -->
<script type="text/javascript">

function changeVisibility(TR){
 if (document.getElementById(TR).style.visibility=="hidden") {
  document.getElementById(TR).style.visibility = "visible";
 } else {
  document.getElementById(TR).style.visibility = "hidden";
 }
}
</script>
<body>
<h1>Hiding and Showing Elements</h1>
<table >
 <tr id="tr1"><td>tableElem Row 1</td></tr>
 <tr id="tr2"><td>tableElem Row 2</td></tr>
 <tr id="tr3"><td>tableElem Row 3</td></tr>
 <tr id="tr4"><td>tableElem Row 4</td></tr>
</table>
<form>
 <h2>visibility</h2>
 <input type="button" onclick="changeVisibility('tr1')" value="TR1">
 <input type="button" onclick="changeVisibility('tr2')" value="TR2">
 <input type="button" onclick="changeVisibility('tr3')" value="TR3">
 <input type="button" onclick="changeVisibility('tr4')" value="TR4">
</form>
</body>
</html>
```

Changing Colors and Fonts

The background and foreground colors of the document display and font properties of the text can be changed dynamically.

Changing the Color

Document colors can be set dynamically, although it is questionable whether this is a good idea. Your colors were chosen because they best presented your page contents. If, though, you wish to provide color alternatives, you can make these colors user selectable. In the following example, a range of background and foreground colors can be chosen by clicking radio buttons.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">
<!—color.html -- >
<head>
<script type="text/javascript">
 function ChangeBackground(Color) {
 document.body.style.backgroundColor = Color
 }
 function ChangeForeground(Color) {
 document.body.style.color = Color
}
</script>
</head>
<body>
<P class=head2>Scripting Document Colors</P>
<P>Some of these document properties have style sheet equivalents. For instance,the
<SPAN class=code><B>bgColor</B></SPAN> and <SPAN
class=code><B>fgColor</B></SPAN> properties are equivalent to the <SPAN
class=code>background-color</SPAN> and <SPAN class=code>color</SPAN> style
properties, respectively. Still, it is always preferable to use CSS style properties when
scripting style changes for compatibility with standards.</P><P>Document colors can be
set dynamically, although it is questionable whether this is a good idea. Supposedly, your
colors were chosen because they best presented your page contents. If, though, you wish
to provide color alternatives, you can make these colors user selectable. In the following
example, a range of background and foreground colors can be chosen by clicking radio
(continued on next page) buttons.</P>
```

In the above example functions ChangeBackground( ) and ChangeForeground( ) are called whenever user clicks the radio buttons of different colors to change the background and foreground color of the document respectively.

```
<b>Background Color:</b><br/>
<input type="radio" name="BG" onclick="ChangeBackground('red')"/>Red
 <input type="radio" name="BG" onclick="ChangeBackground('green')"/>Green
 <input type="radio" name="BG" onclick="ChangeBackground('blue')"/>Blue
 <input type="radio" name="BG" onclick="ChangeBackground('black')"/>Black
 <input type="radio" name="BG" onclick="ChangeBackground('white')"/>White
 <br/><br/>
 <b>Foreground Color:</b><br/>
 <input type="radio" name="FG" onclick="ChangeForeground('red')"/>Red
 <input type="radio" name="FG" onclick="ChangeForeground('green')"/>Green
 <input type="radio" name="FG" onclick="ChangeForeground('blue')"/>Blue
 <input type="radio" name="FG" onclick="ChangeForeground('black')"/>Black
 <input type="radio" name="FG" onclick="ChangeForeground('white')"/>White
</body>
</html>
```

Changing font

Web users are accustomed to having links in documents change color when the cursor is placed over them. Any property of a link can be changed by using the mouse event, ‘ mouseover ‘ to trigger JavaScript event handlers. Thus the font style and font size, as well as the color, can be changed when the cursor is placed over a link. The link can be changed back to its original form when an event handler is triggered with the ‘ mouseout ‘event. In the following example, the only element is a sentence with an embedded link. The foreground color for the document is the default black. The link is presented in blue. When the mouse cursor is placed over the link, its color changes to red and its font style changes to italic.

```
<?xml version = "1.0" encoding = "utf-8"?>

  <!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<!-- link.html  -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Dynamic fonts for links </title>
    <style type = "text/css"> .regText {font: Times; font-size: 16pt;}   </style>
  </head>
  <body>
    <p class = "regText">The subject <a style = "color: blue;"
                onmouseover = "this.style.color = 'red'; this.style.font = 'italic 16pt Times';"
                onmouseout = "this.style.color = 'blue'; this.style.font = 'normal 16pt
```

```
Times';"> Web Programming  </a>  is very interesting
    </p>
  </body>
</html>
```

Output
Display of link,html with the cursor not over the link

The subject Web Programming Is very interesting

Display of link,html with the cursor over the link

The subject *Web Programming* Is very interesting

Dynamic Content

Using JavaScript we can change the content of the different document elements like contents of text box, document title , table contents dynamically. In the following example HTML tables can be created and manipulated dynamically with JavaScript. Each table element contains a rows array and methods for inserting and deleting rows: insertRow() and deleteRow(). Each tr element contains a cells array and methods for inserting and deleting cells: insertCell() and deleteCell(). The following example shows how these objects can be used to dynamically create HTML tables.

The body of the page contains a table with an id of formName. The table contains a single row of headers.

Below the table is a form that allows the user to enter a first and last name. When the "Add Name" button is clicked, the addRow() function is called and passed in the id of the table (tblPeople) and a new array containing the user-entered values.

The addRow() function uses the insertRow() method of the table to add a new row at the end of the table and then loops through the passed-in array, creating and populating one cell for each item. The function also returns the new row. Although the returned value isn't used in this example, it can be useful if you then want to manipulate the new row further.

The above example display the document with empty table. Whenever user press the Add Name button a new row is added to the table.

```
<?xml version = "1.0" encoding = "utf-8"?>
```

```
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
< ! -- dynamic.html -- >
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title>Manipulating Tables</title>

<script type="text/javascript">
function addRow(tableId, cells){
 var tableElem = document.getElementById(tableId);
 var newRow = tableElem.insertRow(tableElem.rows.length);
 var newCell;
 for (var i = 0; i < cells.length; i++) {
  newCell = newRow.insertCell(newRow.cells.length);
  newCell.innerHTML = cells[i];
 }
 return newRow;
}
</script>
</head>

<body>
<table id="tblPeople" border="1">
<tr>
 <th>First Name</th>
 <th>Middle Name</th>
 <th>Last Name</th>
</tr>
</table>
<hr>
<form name="formName">
 First Name: <input type="text" name="FirstName">
 Middle Name: <input type="text" name="MiddleName">
 Last Name: <input type="text" name="LastName">
 <br />
 <br />
  <center><input type="button" value="Add Name"
  onclick="addRow('tblPeople',
   [this.form.FirstName.value,this.form.MiddleName.value, this.form.LastName.value]
);"> </center>
 <hr>
```

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

Stacking Elements

The multiple elements can occupy the same space in the document, one is considered to be on top and is displayed. The top element hides the parts of the lower elements on which it is imposed. When multiple elements occupy the same space on the document, then comes the question of which element is to be placed on the top of other elements. So for this we have to consider the third dimension of the document. Although the display is restricted to two dimensions, the effect of the third dimension is possible through the concept of stacked element. The placement of element in this third dimension is controlled by the z-index attribute of element. An element whose z-index is greater than that of element in the same space will be displayed over the other element, effectively hiding the element with smaller z-index value. The JavaScript property associated with the z-index attribute is zIndex.

In the following example, three images are placed on the display so that they verlap. In XHTML description of this, each image tag includes an onclick attribute, which is used to trigger the execution of JavaScript handler function. First the function defines DOM addresses the last top element and the new top element. Then the function sets the zIndex value of the two elements so that the old top element has the value of 0 and the new top element has the value 10, effectively putting it at the top. The script keeps track of which image is currently on top with the global variable top, which is changed every time a new element is moved to the top with the toTop function.

This example displays all 3 overlapping images. Whenever user clicks on the particular image,that image will be displayed on top of other two images.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
 <html xmlns = "http://www.w3.org/1999/xhtml">
<! – satck.html -->
```

```
 <head>
  <title> Dynamic stacking of images </title>
   <script type = "text/javascript">
     var top = "i1";
    function toTop(newTop) {
       domTop = document.getElementById(top).style;
       domNew = document.getElementById(newTop).style;
       domTop.zIndex = "0";
       domNew.zIndex = "10";
       top = newTop;
     }
   </script>


<style type = "text/css">
     .img1 {position: absolute; top: 0; left: 0;  z-index: 0;}
     .img2 {position: absolute; top: 50px; left: 110px;   z-index: 0;}
     .img3 {position: absolute; top: 100px; left: 220px;  z-index: 0;}    </style>
 </head>
 <body>
  <p>      <img class = "img1"  id = "i1"     src = "image1.jpg"   onclick =
"toTop('i1')" />
            <img  class = "img2"  id = "i2"     src = "image2.jpg"   onclick =
"toTop('i2')" />
            <img  class = "img3"  id = "i3"     src = "image3.jpg"   onclick =
"toTop('i3')" />
   </p>
 </body>
</html>
```

Locating the Mouse Cursor

An event is a notification about something specific that has occurred because of browser user action, such as mouse click on a form button, radio button etc. Strictly speaking event is an object that is implicitly created by the browser and the JavaScript system in response to something happened. Event object includes information about the event. For example mouse-click is one such event. Whenever user mouse-click on the document, event object is created which includes information like where and on which element event has occurred. A mouse-click event defines two pairs of properties that give geometric coordinates of the position of the element in the display that created the event. One pair of the coordinates, clientX and clientY gives the coordinates of the document display relative to the upper-left corner of the browser display window, in pixels. The other pair, screen and screenY gives coordinates of the element but relative client computer's screen.

In the following example, every time user clicks the mouse button an alter box displays the clientX, clientY, screenX and screen coordinates. The handler show_coords with event as parameter is triggred by the onclick attribute of the body element.

Reacting to a Mouse Click

In the following example mousedown and mouseup events are used to show and hide the image on the display under the mouse cursor whenever the mouse button is clicked. Whenever user clicks the mouse button ie. onmousedown event handler displayIt( ) is called with event as parameter and whenever user leaves the mouse button ie. onmouseup event handler hideIt() is called.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<!—mouse.html -- >
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript">
function show_coords(event)
{
x=event.clientX;
y=event.clientY;
x1=event.screenX;
y1=event.screenY;

alert("ClientX coords: " + x + ", ClientY coords: " + y + ",ScreenX coords:
" + x1 + ", ScreenY coords: " + y1);
}
</script>
</head>
```

```
<body onclick="show_coords(event)">
<p>Click in the document. An alert box will alert the x and y coordinates of
the cursor.</p>
</body>
</html>

<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<!—reacting.html
     Display a image when the mouse button is pressed,
-->
<html   xmlns = "http://www.w3.org/1999/xhtml">
  <head>
   <title> Sense events anywhere </title>


<script type = "text/javascript">
      function displayIt(evt) {
        var dom = document.getElementById("image");
        dom.style.left = (evt.clientX - 30) + "px";
        dom.style.top = (evt.clientY - 25) + "px";
        dom.style.visibility = "visible";
      }

      function hideIt() {
        document.getElementById("image").style.visibility =
            "hidden";
   }
    </script>
  </head>
<body onmousedown="displayIt(event);"
      onmouseup = "hideIt();">
     <div id= "image"
           style = " visibility: hidden; position: relative">
         <img src = "smiley.gif"  />
      </div>
  </body>
</html>
```

Slow Movements of Elements

With JavaScript we can not only move the elements instantly, we can also move element slowly from one place to another. The one way to move an element slowly is to move it by small amount many times, with moves separated by small amounts of time. Such slow movement of elements is accomplished by using two window methods available in JavaScript: setTimeout and setInterval.

The setTimeout method has two parameters -- first one is string of JavaScript code to be executed and second parameter is number of mseconds of delay before executing the given script.

The setInterval method has two forms. One takes two parameter with first specifying the script code to be executed and second specifying the interval in mseconds between executions. The second form takes variable number of parameters. First one is the name of the function to be called, the second is the interval in mseconds between the calls to the function and remaining parameter are used as actual parameters to the function being called.

In the following example, image is moved slowly from position(50,100) to a new position (700,100). The move is accomplished by using the setTimeout method to call a move function every msecond until the final position is reached. Example includes separate html file and JavaScript file.

Initial position of the element is set with initImage( ) function. The onload attribute of the body element is used to call this function.

The moveImage( ) function is used to move the image. It moves the image 1 pixel towards the final position and then calls itself with the new coordinates using the setTimeout.

```
 /* moveImage.js JavaScript to move image*/

var dom, x, y, finalx = 700;

  function initImage() {
    dom = document.getElementById('image').style;

  /* Get the current position of the image */
    var x = parseInt(dom.left);
    var y = parseInt(dom.top);
  /* Call the function that moves it */
     moveImage(x, y);
  }
// A function to move the text from its original
//  position to (x, finaly)

  function moveImage(x, y) {

  /* If the x coordinates are not equal, move
      x toward finalx */

     if (x != finalx)
         x++;

     if ((x != finalx) ) {

        dom.left = x + "px";
        dom.top = y + "px";

  /* Recursive call, after a 1-millisecond delay */

        setTimeout("moveImage(" + x + "," + y + ")", 1);
     }

   }


<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<!-- Illustrates a moving image from one position to another
     Uses the JavaScript from file moveimage.js
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> Moving Image </title>
     <script type = "text/javascript"    src = "moveimage.js">
     </script>
  </head>
  <body onload = "initImage()">
    <p>
      <span id = 'image' style = "position: absolute; left: 50px; top: 100px">
       <img src = "smiley.gif"  />
      </span>
    </p>
  </body>
</html>
```

Dragging and Dropping Elements

One of more powerful effects of event handling is allowing the user to drag and drop elements around the display screen. The mouseup, mousedown and mousemove events can be used implement this. By changing the left and top properties, element can be move from one place to another. To illustrate drag and drop, we develop an example that allows the user to drag and drop a rectangle box. In this first we create a box that is to be moved around in document display. There three different handlers for mouseup, mousedown and mousemove events. The mousedown event handler mouseD, takes the event as its parameter. It gets the element to be moved and puts it in global variable so that it is vailable to the other handlers. Then it determines the coordinates of the current position of the element to be moved and computes the difference between them and the coordinates of the position of the mouse cursor. These two differences , which are used by the handler for mousemove to actually move the element.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<title>Drag and drop</title>
<style type="text/css">
body {
 background-color: #fff;
 margin: 0;
 }
   p {
 margin: 80px 0 0 100px;
}
#dObject {
 border: 2px solid;
 border-color: #6c0 #170 #170 #6c0;
 background-color: red;
 width: 40px;
 height: 40px;
 padding: 0.5em 0.8em;
 position: absolute;
 text-align: center;
 display: none;
 cursor: default;
 }
</style>

<script type="text/javascript">

var dragObject, offsetX, offsetY, isDragging=false;
window.onload = init;
document.onmousemove = mouseM;
document.onmouseup = mouseU;

function init() {
 var ob = document.getElementById("dObject");

 ob.style.left="100px";
 ob.style.top="100px";
 41
 ob.style.display="block";
}


function mouseD(ob,e) {
 dragObject = ob;
 if (window.event) e=window.event;

 var dragX = parseInt(dragObject.style.left);
 var dragY = parseInt(dragObject.style.top);
 var mouseX = e.clientX;
 var mouseY = e.clientY;

 offsetX = mouseX - dragX;
 offsetY = mouseY - dragY;
 isDragging = true;
 return false;
}

function mouseU() {
 if (!isDragging) return;
 isDragging = false;
 return false;
}

function mouseM(e) {
 if (!isDragging) return;

 if (window.event) e=window.event;

 var newX = e.clientX - offsetX;
 var newY = e.clientY - offsetY;

 dragObject.style.left = newX + "px";
 dragObject.style.top = newY + "px";

 return false;
}

</script>
</head>
<body>
<p style="position:absolute;left:300px"> Drag and Drop the Box </p>
<div id="dObject" onmousedown="mouseD(this,event)"></div>
</body>
</html>
```

UNIT - 5: XML

SYLLABUS:

Introduction; Syntax; Document structure;
Document Type definitions; Namespaces;
Displaying raw XML documents
Displaying XML documents with CSS;
XSLT style sheets; XML processors;
XML schemas;; Web services.

 XML imposes two distinct levels of syntax:
o There is a general low level syntax that is appreciable on all XML documents
o The other syntactic level is specified by DTD (Document Type Definition) or XML schemas.
 The DTDs and XML schemas specify a set of tag and attribute that can appear in a particular document or collection of documents.
 They also specify the order of occurrence in the document.
 The XML documents consists of data elements which form the statements of XML document.
 The XML document might also consists of markup declaration, which act as instructions to the XML parser
 All XML documents begin with an XML declaration. This declaration identifies that the document is a XML document and also specifies version number of XML standard.
 It also specifies encoding standard.
<?xml version = “1.0” encoding = “utf-8”?>
 Comments in XML is similar to HTML
 XML names are used to name elements and attributes.
 XML names are case-sensitive.
 There is no limitation on the length of the names.
 All XML document contains a single root element whose opening tag appears on first line of the code
 All other tags must be nested inside the root element
 As in case of XHTML, XML tags can also have attributes
 The values for the attributes must be in single or double quotation
Example:
1. <?xml version = “1.0” encoding = “utf-8”?>
<student>
  <name>Santhosh B S</name>

<usn>1RN10CS090</usn>
</student>
2. Tags with attributes
The above code can be also written as
<student name = “Santhosh B S” usn = “1RN10CS090”>
</student>
XML DOCUMENT STRUCTURE
 An XML document often consists of 2 files:
o One of the document – that specifies its tag set
o The other specifies the structural syntactic role and one that contains a style sheet to describe how content of the document is to be printed
 The structural roles are given as either a DTD or an XML schema
 An XML document consists of logically related collection of information known as entities
 The *document entity* is the physical file that represent the document itself
 The document is normally divided into multiple entities.
 One of the advantage dividing document into multiple entities is managing the document becomes simple
 If the same data appears in more than one place, defining it as an entity allows number of references to a single copy of the data
Many documents include information that cannot be represented as text. Ex: images Such information units are stored as binary data These binary data must be a separate unit to be able to include in XML document These entities are called as *Binary entities* When an XML processor encounters the name of a non-binary entity in a document, it replaces name with value it references Binary entities can be handled only by browsers XML processor or parsers can only deal with text Entity names can be of any length. They must begin with a letter, dash or a colon A reference to an entity is its name with a prepended ampersand and an appended semicolon
Example: if **stud_name** is the name of entity, **&stud_name;** is a reference to it One of the use of entities is to allow characters used as markup delimiters to appears as themselvesThe entity references are normally placed in CDATA section Syntax: **<![CDATA[** *content* **]] >**
For example, instead of The last word of the line is &gt;&gt;&gt; here &lt;&lt;&lt;. the following could be used:
<![CDATA[The last word of the line is >>> here <<<]]>
UMENT TYPE DEFINITIONS
A DTD is a set of structural rules called declarations which specify a set of elements that can appethe document. It also specifies how and where these elements appear DTD also specify entity definitions DTD is more useful when the same tag set definition is used by collection of documents A DTD can be embedded in XML document whose syntax rules it describes In this case, a DTD is called as *internal DTD* or a separate file can be created

which can be linked to file. In this case the DTD is called as *External DTD* An external DTD can be used with more than one XML file Syntactically, a DTD is a sequence of declarations. Each declaration has the form of markup declaratiExample: <!keyword...>
Four possible keywords can be used in a declaration:
o ELEMENT, used to define tags;
o ATTLIST, used to define tag attributes;
o ENTITY, used to define entities; and
o NOTATION, used to define data type notations.

ARING ELEMENTS

DTD follows rules of context-free grammar for element declaration A DTD describes the syntactic structure of a particular set of documents Each element declaration in a DTD specifies the structure of one category of elements An element is a node in such a tree either a leaf node or an internal node If element is leaf node, its syntactic description is its character pattern If the element is internal node, its syntactic description is a list of its child element The form of an element declaration for elements that contain elements is as follows:

<!ELEMENT *element_name* (*list of names of child elements*)>

For example, consider the following declaration:
<!ELEMENT memo (from, to, date, re, body)>
This element declaration would describe the document tree structure shown in Figure 7.1.

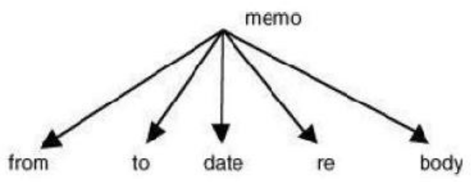
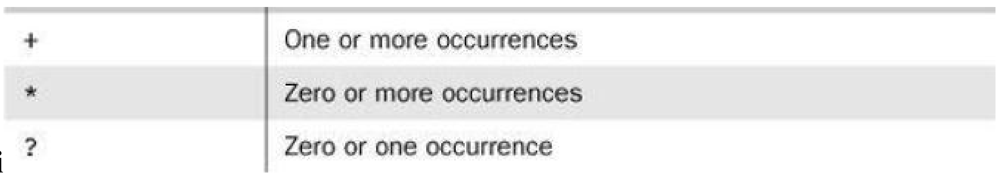


Figure 7.1 An example of the document tree structure for an element definition

In many cases, it is necessary to specify the number of times that a child element may appear. This can be done in a DTD declaration by adding a modifier to the child element specification. These modifiersdescribed in Table 7.1, are borrowed from regular expressions. Any child element specification can be followed by one of the modifiers.

Modifier **Meaning**

Modifier



+	One or more occurrences
*	Zero or more occurrences
?	Zero or one occurrence

Meani

Consider the following DTD declaration:
<!ELEMENT person (parent+, age, spouse?, sibling*)>
□ In this example, a person element is specified to have the following child elements: one or more parent elements, one age element, possibly a spouse element, and zero or more sibling elements.
□ The leaf nodes of a DTD specify the data types of the content of their parent nodes, which are elements.
□ In most cases, the content of an element is type PCDATA, for parsable character data. Parsable character data is a string of any printable characters except "less than" (<), "greater than" (>), and the ampersand (&).
□ Two other content types can be specified: EMPTY and ANY.
□ The EMPTY type specifies that the element has no content; it is used for elements similar to the XHTML img element.
□ The ANY type is used when the element may contain literally any content.
□ The form of a leaf element declaration is as follows:
<!ELEMENT *element_name* (#PCDATA)>
DECLARING ATTRIBUTES
The attributes of an element are declared separately from the element declaration in a DTD. An attribute declaration must include the name of the element to which the attribute belongs, the attribute's name, its type, and a default option. The general form of an attribute declaration is as follows:

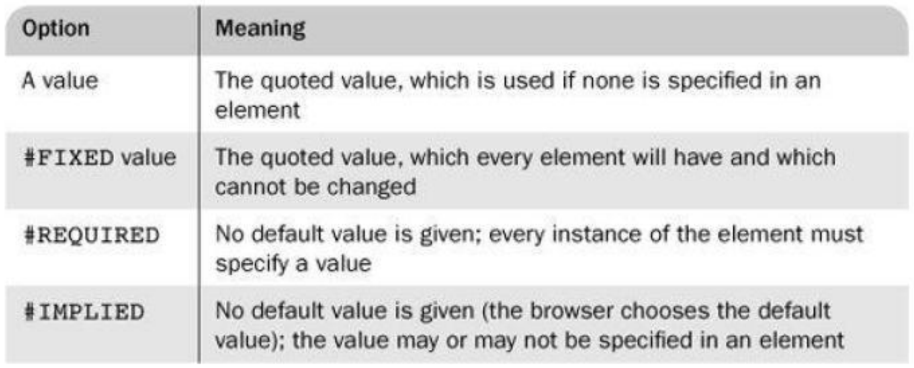
<!ATTLIST *element_name attribute_name attribute type default_option*>

If more than one attribute is declared for a given element, the declarations can be combined, as in the following
element:

```
<!ATTLIST element_name
          attribute name_1  attribute type  default_option_1
          attribute_name_2  attribute_type  default_option_2
          . . .
          attribute_name_n  attribute_type  default_option_n
>
```

The default option in an attribute declaration can specify either an actual value or a requirement for the value of the attribute in the XML document.

Table 7.2 Possible default options for attributes



Option	Meaning
A value	The quoted value, which is used if none is specified in an element
#FIXED value	The quoted value, which every element will have and which cannot be changed
#REQUIRED	No default value is given; every instance of the element must specify a value
#IMPLIED	No default value is given (the browser chooses the default value); the value may or may not be specified in an element

For example, suppose the DTD included the following attribute specifications:

Then the following XML element would be valid for this DTD:
<airplane places = “10” engine_type = “jet”> </airplane>

DECLARING ENTITIES

 Entities can be defined so that they can be referenced anywhere in the content of an XML document, in which case they are called general entities. The predefined entities are all general entities.

 Entities can also be defined so that they can be referenced only in DTDs, in which case they are called parameter entities.

 The form of an entity declaration is

<!ENTITY [%] *entity_name* “*entity_value*”>

 When the optional percent sign (%) is present in an entity declaration, it specifies that the entity is a parameter entity rather than a general entity.

 Example: <!ENTITY sbs “Santhosh B Suresh”>

 When an entity is longer than a few words, its text is defined outside the DTD. In such cases, the entity is called an external text entity. The form of the declaration of an external text entity is

<!ENTITY *entity_name* SYSTEM “*file_location*”>

A Sample DTD

```
<?xml version = "1.0" encoding = "utf-8"?>

<!-- planes.dtd - a document type definition for
                  the planes.xml document, which specifies
                  a list of used airplanes for sale  -->

<!ELEMENT planes_for_sale (ad+)>
<!ELEMENT ad (year, make, model, color, description,
              price?, seller, location)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT make (#PCDATA)>
<!ELEMENT model (#PCDATA)>
<!ELEMENT color (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT seller (#PCDATA)>
<!ELEMENT location (city, state)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>

<!ATTLIST seller phone CDATA #REQUIRED>
<!ATTLIST seller email CDATA #IMPLIED>

<!ENTITY c "Cessna">
<!ENTITY p "Piper">
<!ENTITY b "Beechcraft">
```

Some XML parsers check documents that have DTDs in order to ensure that the documents conform to the structure specified in the DTDs. These parsers are called validating parsers.

If an XML document specifies a DTD and is parsed by a validating XML parser, and the parser determines that the document conforms to the DTD, the document is called valid.

Handwritten XML documents often are not well formed, which means that they do not follow XML'ssyntactic rules.

Any errors they contain are detected by all XML parsers, which must report them.

XML parsers are not allowed to either repair or ignore errors.

Validating XML parsers detect and report all inconsistencies in documents relative to their DTDs.

INTERNAL AND EXTERNAL DTDs

Internal DTD Example:

```
<?xml version = "1.0" encoding = "utf-8"?>
     <!DOCTYPE planes [
         <!-- The DTD for planes -->
     ]>
<!-- The planes XML document -->
```

External DTD Example: [assuming that the DTD is stored in the file named *planes.dtd*]

<!DOCTYPE planes_for_sale SYSTEM “planes.dtd”>

```
//sampleDTD.xml
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE vtu_stud_info SYSTEM "vtu.dtd">
<VTU>
<students>
      <USN> 1RN10CS090 </USN>
      <name> Santhosh B S</name>
      <college> RNSIT </college>
      <branch> CSE </branch>
      <year> 2010 </year>
      <email> santhosh.b.suresh@gmail.com </email>
</students>
<students>
      <USN> 1RN0IS016 </USN>
       <name> Divya K </name>
      <college> RNSIT </college>
      <branch> ISE </branch>
      <year> 2009 </year>
      <email> divya@gmail.com </email>
</students>
</VTU>
```

NAMESPACES

• One problem with using different markup vocabularies in the same document is that collisions between names that are defined in two or more of those tag sets could result.

• An example of this situation is having a <table> tag for a category of furniture and a <table> tag from XHTML for information tables.

• Clearly, software systems that process XML documents must be capable of unambiguously recognizing the element names in those documents.

• To deal with this problem, the W3C has developed a standard for XML namespaces (at http://www.w3.org/TR/REC-xml-names).

• An XML namespace is a collection of element and attribute names used in XML documents. The name of a namespace usually has the form of a uniform resource identifier (URI).

• A namespace for the elements and attributes of the hierarchy rooted at a particular element is declared as the value of the attribute xmlns.

The form of a namespace declaration for an element is

<*element_name* xmlns[:*prefix*] = URI>

• The square brackets indicate that what is within them is optional. The prefix, if included, is the name that must be attached to the names in the declared namespace.

• If the prefix is not included, the namespace is the default for the document.

• A prefix is used for two reasons. First, most URIs are too long to be typed on every occurrence of every name from the namespace. Second, a URI includes characters that are invalid in XML.
• Note that the element for which a namespace is declared is usually the root of a document.
• For ex: all XHTML documents in this notes declare the xmlns namespace on the root element, html:
<html xmlns = “http://www.w3.org/1999/xhtml”>
• This declaration defines the default namespace for XHTML documents, which is http://www.w3.org/1999/xhtml.
• The next example declares two namespaces. The first is declared to be the default namespace; the
second defines the prefix, cap:

```
<states>
  xmlns = "http://www.states-info.org/states"
  xmlns:cap = "http://www.states-info.org/state-capitals"
  <state>
    <name> South Dakota </name>
    <population> 754844 </population>
    <capital>
      <cap:name> Pierre </cap:name>
      <cap:population> 12429 </cap:population>
    </capital>
  </state>
  <!-- More states -->
</states>
```

XML SCHEMAS
XML schemas is similar to DTD i.e. schemas are used to define the structure of the document
DTDs had several disadvantages:
 The syntax of the DTD was un-related to XML, therefore they cannot be analysed with an XML processor
 It was very difficult for the programmers to deal with 2 different types of syntaxes
 DTDs does not support the datatype of content of the tag. All of them are specified as text
Hence, schemas were introduced
SCHEMA FUNDAMENTALS
 Schemas can be considered as a class in object oriented programming
 A XML document that conforms to the standard or to the structure of the schema is similar to an object
 The XML schemas have 2 primary purposes.

o They are used to specify the structure of its instance of XML document, including which elements and attributes may appear in instance document. It also specifies where and how often the elements may appear
o The schema specifies the datatype of every element and attributes of XML
 The XML schemas are ***namespace-centric***

DEFINING A SCHEMA

Schemas themselves are written with the use of a collection of tags, or a vocabulary, from a namespace that is, in effect, a schema of schemas. The name of this namespace is http://www.w3.org/2001/XMLSchema.
 Every schema has schema as its root element. This namespace specification appears as follows:
xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
 The name of the namespace defined by a schema must be specified with the targetNamespace attribute of the schema element.
targetNamespace = "http://cs.uccs.edu/planeSchema"
 If the elements and attributes that are not defined directly in the schema element are to be included in the target namespace, schema's elementFormDefault must be set to qualified, as follows: **elementFormDefault = "qualified"**
 The default namespace, which is the source of the unprefixed names in the schema, is given withanother xmlns specification, but this time without the prefix: **xmlns = http://cs.uccs.edu/planeSchema** Example in 2 alternate methods of defining a schema

The above is an alternative to the preceding opening tag would be to make the XMLSchema names the default so that they do not need to be prefixed in the schema. Then the names in the target namespace would need to be prefixed.

DEFINING A SCHEMA INSTANCE

The above is an alternative to the preceding opening tag would be to make the XMLSchema names the default so that they do not need to be prefixed in the schema. Then the names in the target namespace would need to be prefixed.
 An instance document normally defines its default namespace to be the one defined in its schema.
For example, if the root element is planes, we could have
<planes
xmlns = "http://cs.uccs.edu/planeSchema"
... >
 The second attribute specification in the root element of an instance document is for the schemaLocation attribute. This attribute is used to name the standard namespace for instances, which includes the name XMLSchema-instance.
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

 Third, the instance document must specify the filename of the schema in which the default namespace is defined. This is accomplished with the schemaLocation attribute, which takes two values: the namespace of the schema and the filename of the schema.

```
xsi:schemaLocation = "http://cs.uccs.edu/planeSchema
                      planes.xsd"
```

 Combining everything, we get,

```
<planes
   xmlns = "http://cs.uccs.edu/planeSchema"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://cs.uccs.edu/planeSchema
                         planes.xsd">
```

AN OVERVIEW OF DATA TYPES

There are two categories of user-defined schema data types: simple and complex.

 A simple data type is a data type whose content is restricted to strings. A simple type cannot have attributes or include nested elements.

 A complex type can have attributes and include other data types as child elements.

Data declarations in an XML schema can be either local or global.

 A local declaration is a declaration that appears inside an element that is a child of the schema element.

 A global declaration is a declaration that appears as a child of the schema element. Global elements
are visible in the whole schema in which they are declared.

SIMPLE TYPES

 Elements are defined in an XML schema with the element tag.

<xsd:element name = “engine” type = “xsd:string” />

 An instance of the schema in which the engine element is defined could have the following element:

<engine> inline six cylinder fuel injected </engine>

 An element can be given a default value with the default attribute:

```
<xsd:element name = "engine"   type = "xsd:string"
             default = "fuel injected V-6"  />
```

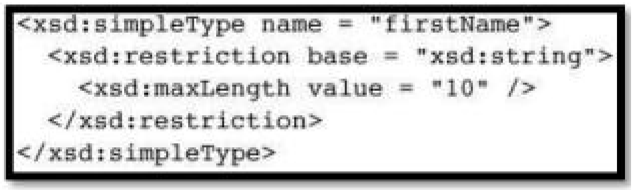
 Constant values are given with the fixed attribute, as in the following example:

```
<xsd:element name = "plane"   type = "xsd:string"
             fixed = "single wing"  />
```

 A simple user-defined data type is described in a simpleType element with the use of facets.

 Facets must be specified in the content of a restriction element, which gives the base type name.

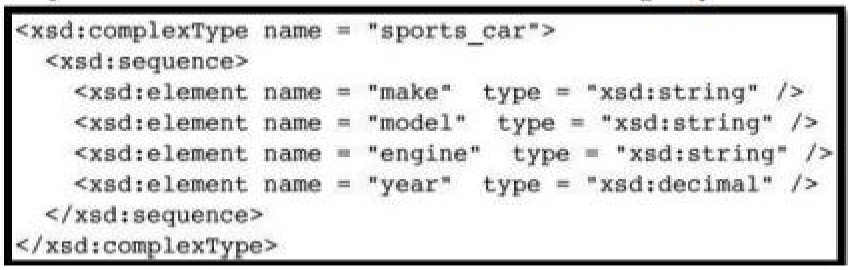
 The facets themselves are given in elements named for the facets: the value attribute specifies the value of the facet.



```
<xsd:simpleType name = "firstName">
  <xsd:restriction base = "xsd:string">
    <xsd:maxLength value = "10" />
  </xsd:restriction>
</xsd:simpleType>
```

COMPLEX TYPES

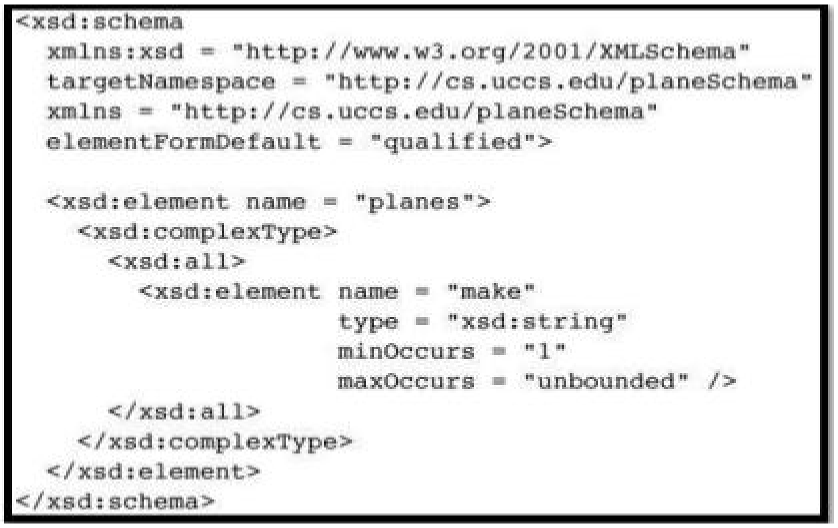
Complex types are defined with the complexType tag. The elements that are the content of an element-only element must be contained in an ordered group, an unordered group, a choice, or a named group. The sequence element is used to contain an ordered group of elements. Example:



```
<xsd:complexType name = "sports_car">
  <xsd:sequence>
    <xsd:element name = "make"   type = "xsd:string" />
    <xsd:element name = "model"  type = "xsd:string" />
    <xsd:element name = "engine"  type = "xsd:string" />
    <xsd:element name = "year"   type = "xsd:decimal" />
  </xsd:sequence>
</xsd:complexType>
```

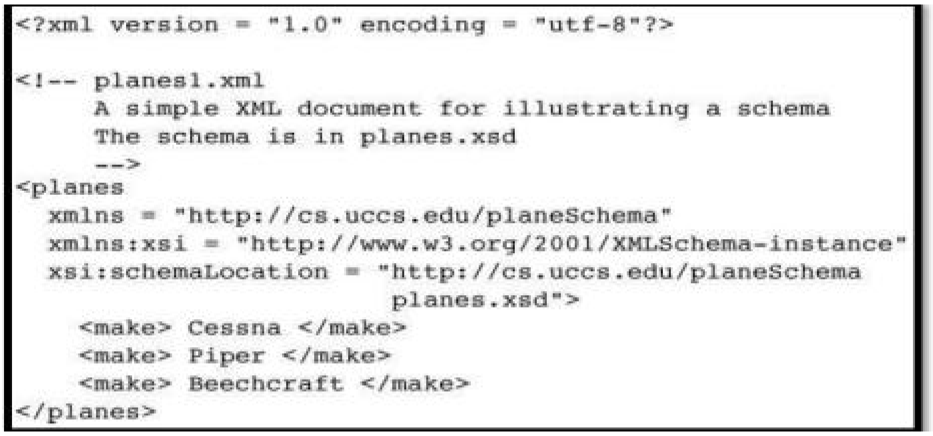
A complex type whose elements are an unordered group is defined in an all element. Elements in all and
sequence groups can include the minOccurs and maxOccurs attributes to specify the numbers of occurrences. Example:

<?xml version = “1.0” encoding = “utf-8”?>



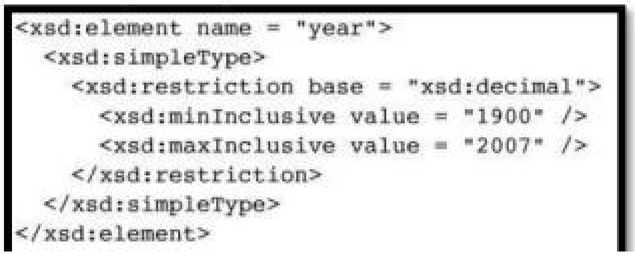
```
<xsd:schema
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://cs.uccs.edu/planeSchema"
  xmlns = "http://cs.uccs.edu/planeSchema"
  elementFormDefault = "qualified">

  <xsd:element name = "planes">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name = "make"
                     type = "xsd:string"
                     minOccurs = "1"
                     maxOccurs = "unbounded" />
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

An XML instance that conforms to the planes.xsd schema is as follows:



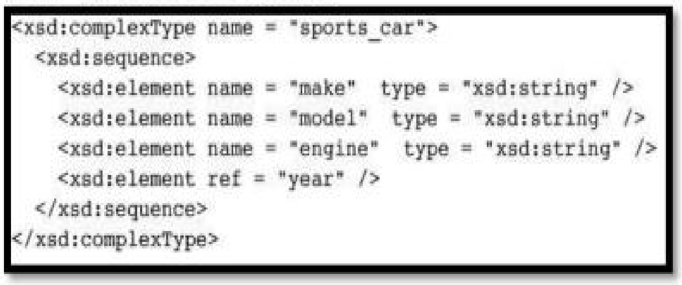
```
<?xml version = "1.0" encoding = "utf-8"?>

<!-- planes1.xml
     A simple XML document for illustrating a schema
     The schema is in planes.xsd
     -->
<planes
  xmlns = "http://cs.uccs.edu/planeSchema"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://cs.uccs.edu/planeSchema
                        planes.xsd">
    <make> Cessna </make>
    <make> Piper </make>
    <make> Beechcraft </make>
</planes>
```

For example, the `year` element could be defined as follows:



```
<xsd:element name = "year">
  <xsd:simpleType>
    <xsd:restriction base = "xsd:decimal">
      <xsd:minInclusive value = "1900" />
      <xsd:maxInclusive value = "2007" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

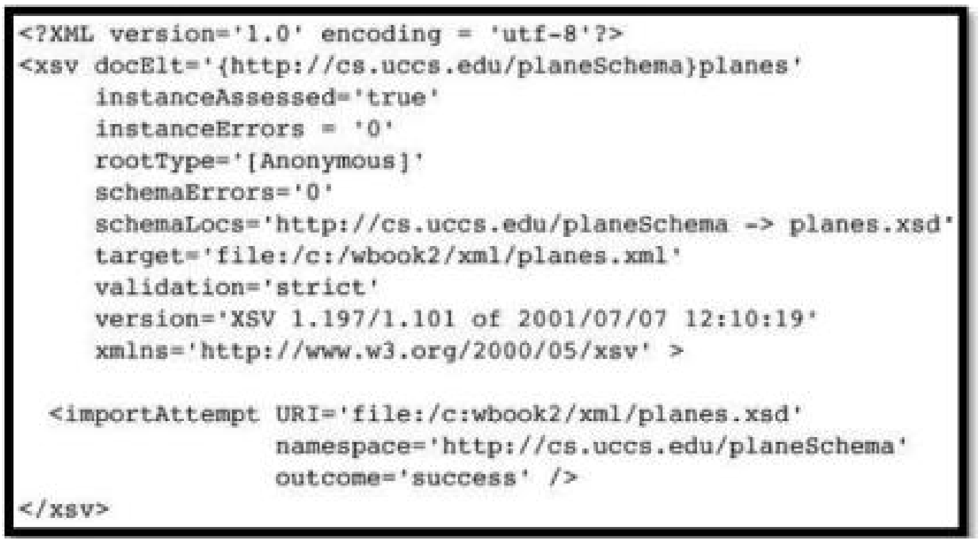
With the `year` element defined globally, the `sports_car` element can be defined with a reference to the `year` with the `ref` attribute:



```
<xsd:complexType name = "sports_car">
  <xsd:sequence>
    <xsd:element name = "make"  type = "xsd:string" />
    <xsd:element name = "model"  type = "xsd:string" />
    <xsd:element name = "engine"  type = "xsd:string" />
    <xsd:element ref = "year" />
  </xsd:sequence>
</xsd:complexType>
```

VALIDATING INSTANCES OF SCHEMAS

XSV is an abbreviation for ***XML Schema Validator***. If the schema and the instance document are available on the Web, xsv can be used online, like the XHTML validation tool at the W3C Web site. This tool can also be downloaded and run on any computer. The Web site for xsv is http://www.w3.org/XML/Schema#XSV.
The output of xsv is an XML document. When the tool is run from the command line, the output document appears on the screen with no formatting, so it is a bit difficult to read. The following is the output of xsv run on planes.xml:



```
<?XML version='1.0' encoding = 'utf-8'?>
<xsv docElt='{http://cs.uccs.edu/planeSchema}planes'
     instanceAssessed='true'
     instanceErrors = '0'
     rootType='[Anonymous]'
     schemaErrors='0'
     schemaLocs='http://cs.uccs.edu/planeSchema -> planes.xsd'
     target='file:/c:/wbook2/xml/planes.xml'
     validation='strict'
     version='XSV 1.197/1.101 of 2001/07/07 12:10:19'
     xmlns='http://www.w3.org/2000/05/xsv' >

  <importAttempt URI='file:/c:wbook2/xml/planes.xsd'
                 namespace='http://cs.uccs.edu/planeSchema'
                 outcome='success' />
</xsv>
```

DISPLAYING RAW XML DOCUMENTS

If an XML document is displayed without a style sheet that defines presentation styles for the document's tags, the displayed document will not have formatted content.

DISPLAYING XML DOCUMENTS WITH CSS

//6a.xml

```
<?xml version = "1.0" encoding = "utf-8"?>

<?xml-stylesheet type = "text/css" href = "6a.css"?>
<VTU>
<students>
      <USN> 1RN10CS090 </USN>
      <name> Santhosh B S</name>
      <college> RNSIT </college>
      <branch> CSE </branch>
      <YOJ> 2010 </YOJ>
      <email> santhosh.b.suresh@gmail.com </email>
</students>
<students>
      <USN> 1RN10CS003 </USN>
       <name> Akash Bangera </name>
      <college> RNSIT </college>
      <branch> CSE </branch>
      <YOJ> 2010 </YOJ>
      <email> akash.bangera@gmail.com </email>
</students>
<students>
      <USN> 1RN10CS050 </USN>
      <name> Manoj Kumar</name>
```

```
    <college> RNSIT </college>
    <branch>CSE </branch>
    <YOJ> 2010</YOJ>
    <email> manoj.kumar@gmail.com </email>
</students>
</VTU>
```

//6a.css

```
students
{ clear: both; float : left;}
USN
{color: green; }
name
{background: yellow;}
college
{ display: none;}
branch
{color : #cd00dc; text-align: right;}
YOJ
{background : red; color : white;}
email
{ color: blue;}
```

XSLT STYLE SHEETS

 The eXtensible Stylesheet Language (XSL) is a family of recommendations for defining the presentation and transformations of XML documents.

 It consists of three related standards:

o XSL Transformations (XSLT),

o XML Path Language (XPath), and

o XSL Formatting Objects (XSL-FO).

 XSLT style sheets are used to transform XML documents into different forms or formats, perhaps using different DTDs.

 One common use for XSLT is to transform XML documents into XHTML documents, primarily for display. In the transformation of an XML document, the content of elements can be moved, modified, sorted, and converted to attribute values, among other things.

 XSLT style sheets are XML documents, so they can be validated against DTDs.

 They can even be transformed with the use of other XSLT style sheets.

 The XSLT standard is given at http://www.w3.org/TR/xslt.

 XPath is a language for expressions, which are often used to identify parts of XML documents, such as specific elements that are in specific positions in the document or elements that have particular attribute values.

 XPath is also used for XML document querying languages, such as XQL, and to build new XML document structures with XPointer. The XPath standard is given at http://www.w3.org/TR/xpath.

OVERVIEW OF XSLT

 XSLT is actually a simple functional-style programming language.

 Included in XSLT are functions, parameters, names to which values can be bound, selection constructs, and conditional expressions for multiple selection.

 XSLT processors take both an XML document and an XSLT document as input. T

 The XSLT document is the program to be executed; the XML document is the input data to the program.

 Parts of the XML document are selected, possibly modified, and merged with parts of the XSLT
document to form a new document, which is sometimes called an XSL document.

 The transformation process used by an XSLT processor is shown in Figure 7.5.

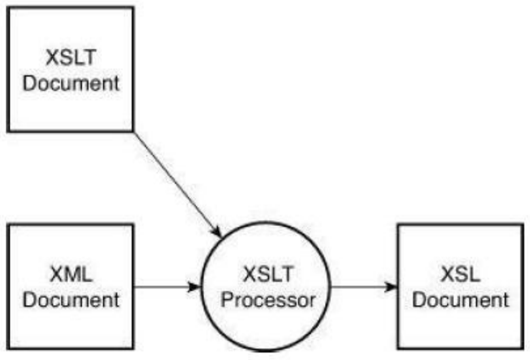


Figure 7.5 XSLT processing

An XSLT document consists primarily of one or more templates.
Each template describes a function that is executed whenever the XSLT processor finds a match to thetemplate's pattern.
One XSLT model of processing XML data is called the template-driven model, which works well when the data consists of multiple instances of highly regular data collections, as with files containingrecords.
XSLT can also deal with irregular and recursive data, using template fragments in what is called thedata-driven model.
A single XSLT style sheet can include the mechanisms for both the template- and data-driven models.

XSL TRANSFORMATIONS FOR PRESENTATION

Consider a sample program:

//6b.xml

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="6b.xsl"?>
```

```
<vtu>
<student>
       <name>Santhosh B S</name>
<usn>1RN10CS090</usn>
       <collegeName>RNSIT</collegeName>
       <branch>CSE</branch>
       <year>2010</year>
       <email> santhosh.b.suresh@gmail.com </email>
</student>
<student>
       <name>Akash Bangera</name>
 <usn>1RN10CS003</usn>
       <collegeName>RNSIT</collegeName>
       <branch>CSE</branch>
       <year>2010</year>
       <email>akash.bangera@gmail.com</email>
</student>
<student>
       <name>Manoj Kumar</name>
 <usn>1RN10CS050</usn>
       <collegeName>RNSIT</collegeName>
       <branch>CSE</branch>
       <year>2010</year>
       <email>manoj.kumar@gmail.com</email>
</student>
</vtu>
```

An XML document that is to be used as data to an XSLT style sheet must include a processing instruction to inform the XSLT processor that the style sheet is to be used. The form of this instruction is as follows:

```
<?xml-stylesheet type = "text/xsl" href =
                                  "XSL_stylesheet_name" ?>
```

//6b.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>VTU Student Information</h2>
    <table border="1">
```

```
      <tr bgcolor="#99cd32">
        <th>name</th>
        <th>usn</th>
        <th>collegeName</th>
        <th>branch</th>
        <th>year</th>
        <th>email</th>
         </tr>

      <xsl:for-each select="vtu/student">
      <xsl:choose>
        <xsl:when test="name = ’Santhosh B S”">
        <tr bgcolor="yellow">
        <td><xsl:value-of select="name"/></td>
        <td><xsl:value-of select="usn"/></td>
        <td><xsl:value-of select="collegeName"/></td>
        <td><xsl:value-of select="branch"/></td>
<td><xsl:value-of select="year"/></td>
        <td><xsl:value-of select="email"/></td>
        </tr>
        </xsl:when>
        <xsl:otherwise>
         <tr >
        <td><xsl:value-of select="name"/></td>
        <td><xsl:value-of select="usn"/></td>
        <td><xsl:value-of select="collegeName"/></td>
        <td><xsl:value-of select="branch"/></td>
        <td><xsl:value-of select="year"/></td>
        <td><xsl:value-of select="email"/></td>
      </tr>
        </xsl:otherwise>
      </xsl:choose>
      </xsl:for-each>
    </table>
<h2>selected student is highlighted</h2>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

An XSLT style sheet is an XML document whose root element is the special-purpose element stylesheet. The stylesheet tag defines namespaces as its attributes and encloses the collection of elements that defines its transformations. It also identifies the document as an XSLT document.

<xsl:stylesheet version="1.0" xmlns:xsl=”http://www.w3.org/1999/XSL/Transform”>

In many XSLT documents, a template is included to match the root node of the XML document.

<xsl:template match="/">

In many cases, the content of an element of the XML document is to be copied to the output document. This is done with the value-of element, which uses a select attribute to specify the element of the XML
document whose contents are to be copied.

<xsl:value-of select="name"/>

The select attribute can specify any node of the XML document. This is an advantage of XSLT formatting over CSS, in which the order of data as stored is the only possible order of display.

XML PROCESSORS

The XML processor takes the XML document and DTD and processes the information so that it may then be used by applications requesting the information. The processor is a software module that reads the XML document to find out the structure and content of the XML document. The structure and content can be derived by the processor because XML documents contain self-explanatory data.

THE PURPOSES OF XML PROCESSORS

 First, the processor must check the basic syntax of the document for well-formedness.

 Second, the processor must replace all references to entities in an XML document with their definitions.

 Third, attributes in DTDs and elements in XML schemas can specify that their values in an XML document have default values, which must be copied into the XML document during processing.

 Fourth, when a DTD or an XML schema is specified and the processor includes a validating parser, the structure of the XML document must be checked to ensure that it is legitimate.

THE SAX APPROACH

 The Simple API for XML (SAX) approach to processing is called event processing.

 The processor scans the XML document from beginning to end.

 Every time a syntactic structure of the document is recognized, the processor signals an event to the application by calling an event handler for the particular structure that was found.

 The syntactic structures of interest naturally include opening tags, attributes, text, and closing tags.

 The interfaces that describe the event handlers form the SAX API.

THE DOM APPROACH

 The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents.

 It defines the logical structure of documents and the way a document is accessed and manipulated

 Properties of DOM

o Programmers can build documents, navigate their structure, and add, modify, or delete elements and content.

o Provides a standard programming interface that can be used in a wide variety of environments and applications.

o structural isomorphism.

 The DOM representation of an XML document has several advantages over the sequential listing provided by SAX parsers.

 First, it has an obvious advantage if any part of the document must be accessed more than once by the application.

 Second, if the application must perform any rearrangement of the elements of the document, that can most easily be done if the whole document is accessible at the same time.

 Third, accesses to random parts of the document are possible.

 Finally, because the parser sees the whole document before any processing takes place, this approach avoids any processing of a document that is later found to be invalid.

WEB SERVICES

A Web service is a method that resides and is executed on a Web server, but that can be called from any computer on the Web. The standard technologies to support Web services are WSDL, UDDI, SOAP, and XML.

WSDL - It is used to describe the specific operations provided by the Web service, as well as the protocols for the messages the Web service can send and receive.

UDDI - also provides ways to query a Web services registry to determine what specific services are available.

SOAP - was originally an acronym for Standard Object Access Protocol, designed to describe data objects.

XML - provides a standard way for a group of users to define the structure of their data documents, using a subject-specific mark-up language.

UNIT - 6: PERL, CGI PROGRAMMING

Origins and uses of Perl; Scalars and their operations
Assignment statements and simple input and output; Control statements;
Fundamentals of arrays; Hashes; References;
Pattern matching; File input and output;
Functions; Examples. The Common Gateway Interface;
CGI linkage; Query string format; CGI.pm module;
A survey example; Cookies. Database access with Perl and MySQL

Origins and uses of Perl

Began in the late 1980s as a more powerful replacement for the capabilities of awk (text file processing) and sh (UNIX system administration)
- Now includes sockets for communications and modules for OOP, among other things
- Now the most commonly used language for CGI, in part because of its pattern matching capabilities
- Perl programs are usually processed the same way as many Java programs, compilation to an intermediate form, followed by interpretation

Scalars and their operations

- Scalars are variables that can store either numbers, strings, or references (discussed later)
- Numbers are stored in double format; integers are rarely used
- Numeric literals have the same form as in other common languages

Perl has two kinds of string literals, those delimited by double quotes and those delimited by single quotes
- Single-quoted literals cannot include escape sequences
- Double-quoted literals can include them
- In both cases, the delimiting quote can be embedded by preceding it with a backslash
- If you want a string literal with single-quote characteristics, but don’t want delimit it with single quotes, use qx, where x is a new delimiter
- For double quotes, use qq
- If the new delimiter is a parenthesis, a brace, a bracket, or a pointed bracket, the right delimiter must be the other member of the pair
  - A null string can be ”or ""

Scalar type is specified by preceding the name with a $

- Name must begin with a letter; any number of letters, digits, or underscore characters can follow
- Names are case sensitive
- By convention, names of variables use only lowercase letters
- Names embedded in double-quoted string literals are interpolated
e.g., If the value of $salary is 47500, the value of
"Jack makes $salary dollars per year" is "Jack makes 47500 dollars per year"
- Variables are implicitly declared
- A scalar variable that has not been assigned a value has the value undef (numeric value is 0; string value is the null string)
- Perl has many implicit variables, the most common
    - of which is $_ (Look at perldoc perlvar)
- Numeric Operators
- Like those of C, Java, etc.

Operator	Associativity
++, --	nonassociative
unary -	right
**	right
*, /, %	left
binary +, -	left

- String Operators
- Catenation - denoted by a period e.g., If the value of $dessert is "apple", the value of $dessert . " pie" is "apple pie"
- Repetition - denoted by x e.g., If the value of $greeting is "hello ", the value of
    - $greeting x 3 is "hello hello hello "
- String Functions
- Functions and operators are closely related in Perl
- e.g., if cube is a predefined function, it can be called with either cube(x) or cube x Name Parameters Result chomp a string the string w/terminating newline characters removed length a string the number of characters in the string lc a string the string with uppercase letters converted to lower uc a string the string with lowercase letters converted to upper hex a string the decimal value of the hexadecimal number in the
string join a character and the strings catenated a list of strings together with the character inserted between them

Control statements

In the last chapter you learned how to decode form data, and mail it to yourself. However, one problem with the guestbook program is that it didn't do any error-checking or specialized processing. You might not want to get blank forms, or you may want to require certain fields to be filled out. You might also want to write a quiz or questionnaire, and have your program take different actions depending on the answers. All of these things require some more advanced processing of the form data, and that will usually involve using control structures in your Perl code.

Control structures include conditional statements, such as if/elsif/else blocks, as well as loops like foreach, for and while.

If Conditions

You've already seen if/elsif in action. The structure is always started by the word if, followed by a condition to be evaluated, then a pair of braces indicating the beginning and end of the code to be executed if the condition is true. The condition is enclosed in parentheses:

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

The condition statement can be anything that evaluates to true or false. In Perl, any string is true except the empty string and 0. Any number is true except 0. An undefined value (or undef) is false.You can also test whether a certain value equals something, or doesn't equal something, or is greater than or less than something. There are different conditional test operators, depending on whether the variable you want to test is a string or a number:

Relational and Equality Operators

Test	Numbers	Strings
$x is equal to $y	$x == $y	$x eq $y
$x is not equal to $y	$x != $y	$x ne $y
$x is greater than $y	$x > $y	$x gt $y
$x is greater than or equal to $y	$x >= $y	$x ge $y
$x is less than $y	$x < $y	$x lt $y
$x is less than or equal to $y	$x <= $y	$x le $y

If it's a string test, you use the letter operators (eq, ne, lt, etc.), and if it's a numeric test, you use the symbols (==, !=, etc.). Also, if you are doing numeric tests, keep in mind that $x >= $y is not the same as $x => $y. Be sure to use the correct operator!

Here is an example of a numeric test. If $varname is greater than 23, the code inside the curly braces is executed:

```
if ($varname > 23) {
   # do stuff here if the condition is true
}
```

If you need to have more than one condition, you can add elsif and else blocks:

```
if ($varname eq "somestring") {
   # do stuff here if the condition is true
}
elsif ($varname eq "someotherstring") {
   # do other stuff
}
else {
   # do this if none of the other conditions are met
}
```

The line breaks are not required; this example is just as valid:

```
if ($varname > 23) {
  print "$varname is greater than 23";
} elsif ($varname == 23) {
  print "$varname is 23";
} else { print "$varname is less than 23"; }
```

You can join conditions together by using logical operators:

Logical Operators

Operator	Example	Explanation
&&	condition1 && condition2	True if condition1 and condition2 are both true
||	condition1 || condition2	True if either condition1 or condition2 is true
and	condition1 and condition2	Same as && but lower precedence
or	condition1 or condition2	Same as || but lower precedence

Logical operators are evaluated from left to right. Precedence indicates which operator is evaluated first, in the event that more than one operator appears on one line. In a case like this:

```
condition1 || condition2 && condition3
```

condition2 && condition3 is evaluated first, then the result of that evaluation is used in the || evaluation.

and and or work the same way as && and ||, although they have lower precedence than their symbolic counterparts.

Unless

unless is similar to if. Let's say you wanted to execute code only if a certain condition were false. You could do something like this:

```
if ($varname != 23) {
  # code to execute if $varname is not 23
}
```

The same test can be done using unless:

```
unless ($varname == 23) {
  # code to execute if $varname is not 23
}
```

There is no "elseunless", but you can use an else clause:

```
unless ($varname == 23) {
   # code to execute if $varname is not 23
} else {
   # code to execute if $varname IS 23
}
```

Validating Form Data

You should always validate data submitted on a form; that is, check to see that the form fields aren't blank, and that the data submitted is in the format you expected. This is typically done with if/elsif blocks.

Here are some examples. This condition checks to see if the "name" field isn't blank:

```
if (param('name') eq "") {
   &dienice("Please fill out the field for your name.");
}
```

You can also test multiple fields at the same time:

```
if (param('name') eq "" or param('email') eq "") {
   &dienice("Please fill out the fields for your name
and email address.");
}
```

The above code will return an error if either the name or email fields are left blank.

param('fieldname') always returns one of the following:

undef — or undefined	fieldname is not defined in the form itself, or it's a checkbox/radio button field that wasn't checked.
the empty string	fieldname exists in the form but the user didn't type anything into that field (for text fields)
one or more values	whatever the user typed into the field(s)

If your form has more than one field containing the same fieldname, then the values are stored sequentially in an array, accessed by param('fieldname').

You should always validate all form data — even fields that are submitted as hidden fields in your form. Don't assume that your form is always the one calling your program. Any external site can send data to your CGI. Never trust form input data.

Looping

Loops allow you to repeat code for as long as a condition is met. Perl has several loop control structures: foreach, for, while and until.

Foreach Loops

foreach iterates through a list of values:

```
foreach my $i (@arrayname) {
   # code here
}
```

This loops through each element of @arrayname, setting $i to the current array element for each pass through the loop. You may omit the loop variable $i:

```
foreach (@arrayname) {
   # $_ is the current array element
}
```

This sets the special Perl variable $_ to each array element. $_ does not need to be declared (it's part of the Perl language) and its scope localized to the loop itself.

For Loops

Perl also supports C-style for loops:

```
for ($i = 1; $i < 23; $i++) {
   # code here
}
```

The for statement uses a 3-part conditional: the loop initializer; the loop condition (how long to run the loop); and the loop re-initializer (what to do at the end of each iteration of the loop). In the above example, the loop initializes with $i being set to 1. The loop will run for as long as $i is less than 23, and at the end of each iteration $i is incremented by 1 using the auto-increment operator (++).

The conditional expressions are optional. You can do infinite loops by omitting all three conditions:

```
for (;;) {
   # code here
}
```

You can also write infinite loops with while.

While Loops

A while loop executes as long as particular condition is true:

```
while (condition) {
   # code to run as long as condition is true
}
```

Until Loops

until is the reverse of while. It executes as long as a particular condition is NOT true:

```
until (condition) {
   # code to run as long as condition is not true
}
```

Infinite Loops

An infinite loop is usually written like so:

```
while (1) {
   # code here
}
```

Obviously unless you want your program to run forever, you'll need some way to break out of these infinite loops. We'll look at breaking next.

Breaking from Loops

There are several ways to break from a loop. To stop the current loop iteration (and move on to the next one), use the next command:

```
foreach my $i (1..20) {
   if ($i == 13) {
      next;
   }
   print "$i\n";
}
```

This example prints the numbers from 1 to 20, except for the number 13. When it reaches 13, it skips to the next iteration of the loop.

To break out of a loop entirely, use the last command:

```
foreach my $i (1..20) {
   if ($i == 13) {
      last;
   }
   print "$i\n";
}
```

This example prints the numbers from 1 to 12, then terminates the loop when it reaches 13.

next and last only effect the innermost loop structure, so if you have something like this:

```
foreach my $i (@list1) {
   foreach my $j (@list2) {
      if ($i == 5 && $j == 23) {
         last;
      }
   }
   # this is where that last sends you
}
```

The last command only terminates the innermost loop. If you want to break out of the outer loop, you need to use loop labels:

```
OUTER: foreach my $i (@list1) {
   INNER: foreach my $j (@list2) {
      if ($i == 5 && $j == 23) {
         last OUTER;
      }
   }
}
# this is where that last sends you
```

The loop label is a string that appears before the loop command (foreach, for, or while). In this example we used OUTER as the label for the outer foreach loop and INNER for the inner loop label.

Now that you've seen the various types of Perl control structures, let's look at how to apply them to handling advanced form data.

Fundamentals of arrays

An array stores an ordered list of values. While a scalar variable can only store one value, an array can store many. Perl array names are prefixed with an @-sign. Here is an example:

```
my @colors = ("red","green","blue");
```

Each individual item (or element) of an array may be referred to by its index number. Array indices start with 0, so to access the first element of the array @colors, you use $colors[0]. Notice that when you're referring to a single element of an array, you prefix the name with $ instead of @. The $-sign again indicates that it's a single (scalar) value; the @-sign means you're talking about the entire array.

If you want to loop through an array, printing out all of the values, you could print each element one at a time:

```
my @colors = ("red","green","blue");

print "$colors[0]\n";             # prints "red"
print "$colors[1]\n";             # prints "green"
print "$colors[2]\n";             # prints "blue"
```

A much easier way to do this is to use a foreach loop:

```
my @colors = ("red","green","blue");
foreach my $i (@colors) {
   print "$i\n";
}
```

For each iteration of the foreach loop, $i is set to an element of the @colors array. In this example, $i is "red" the first time through the loop. The braces {} define where the loop begins and ends, so for any code appearing between the braces, $i is set to the current loop iterator.

Notice we've used my again here to declare the variables. In the foreach loop, my $i declares the loop iterator ($i) and also limits its scope to the foreach loop itself. After the loop completes, $i no longer exists.

We'll cover loops more in Chapter 5.

Getting Data Into And Out Of Arrays

An array is an ordered list of elements. You can think of it like a group of people standing in line waiting to buy tickets. Before the line forms, the array is empty:

```
my @people = ();
```

Then Howard walks up. He's the first person in line. To add him to the @people array, use the push function:

```
push(@people, "Howard");
```

Now Sara, Ken, and Josh get in line. Again they are added to the array using the push function. You can push a list of values onto the array:

```
push(@people, ("Sara", "Ken", "Josh"));
```

This pushes the list containing "Sara", "Ken" and "Josh" onto the end of the @people array, so that @people now looks like this: ("Howard", "Sara", "Ken", "Josh")

Now the ticket office opens, and Howard buys his ticket and leaves the line. To remove the first item from the array, use the shift function:

```
my $who = shift(@people);
```

This sets $who to "Howard", and also removes "Howard" from the @people array, so @people now looks like this: ("Sara", "Ken", "Josh")

Suppose Josh gets paged, and has to leave. To remove the last item from the array, use the pop function:

```
my $who = pop(@people);
```

This sets $who to "Josh", and @people is now ("Sara", "Ken")

Both shift and pop change the array itself, by removing an element from the array.

Finding the Length of Arrays

If you want to find out how many elements are in a given array, you can use the scalar function:

```
my @people = ("Howard", "Sara", "Ken", "Josh");
my $linelen = scalar(@people);
print "There are $linelen people in line.\n";
```

This prints "There are 4 people in line." Of course, there's always more than one way to do things in Perl, and that's true here — the scalar function is not actually needed. All you have to do is evaluate the array in a scalar context. You can do this by assigning it to a scalar variable:

```
my $linelen = @people;
```

This sets $linelen to 4.

What if you want to print the name of the last person in line? Remember that Perl array indices start with 0, so the index of the last element in the array is actually length-1:

```
print "The last person in line is $people[$linelen-1].\n";
```

Perl also has a handy shortcut for finding the index of the last element of an array, the $# shortcut:

```
print "The last person in line is $people[$#people].\n";
```

$#arrayname is equivalent to scalar(@arrayname)-1. This is often used in foreach loops where you loop through an array by its index number:

```
my @colors = ("cyan", "magenta", "yellow", "black");
foreach my $i (0..$#colors) {
  print "color $i is $colors[$i]\n";
}
```

This will print out "color 0 is cyan, color 1 is magenta", etc.

The $#arrayname syntax is one example where an #-sign does not indicate a comment.

Array Slices

You can retrieve part of an array by specifying the range of indices to retrieve:

```
my @colors = ("cyan", "magenta", "yellow", "black");
my @slice = @colors[1..2];
```

This example sets @slice to ("magenta", "yellow").

Finding An Item In An Array

If you want to find out if a particular element exists in an array, you can use the grep function:

```
my @results = grep(/pattern/,@listname);
```

/pattern/ is a regular expression for the pattern you're looking for. It can be a plain string, such as /Box kite/, or a complex regular expression pattern.

/pattern/ will match partial strings inside each array element. To match the entire array element, use /^pattern$/, which anchors the pattern match to the beginning (^) and end ($) of the string.

grep returns a list of the elements that matched the pattern.

Sorting Arrays

You can do an alphabetical (ASCII) sort on an array of strings using the sort function:

```
my @colors = ("cyan", "magenta", "yellow", "black");
my @colors2 = sort(@colors);
```

@colors2 becomes the @colors array in alphabetically sorted order ("black", "cyan", "magenta", "yellow"). Note that the sort function, unlike push and pop, does not change the original array. If you want to save the sorted array, you have to assign it to a variable. If you want to save it back to the original array variable, you'd do:

```
@colors = sort @colors;
```

You can invert the order of the array with the reverse function:

```
my @colors = ("cyan", "magenta", "yellow", "black");
@colors = reverse(@colors);
```

@colors is now ("black", "yellow", "magenta", "cyan").

To do a reverse sort, use both functions:

```
my @colors = ("cyan", "magenta", "yellow", "black");
@colors = reverse(sort(@colors));
```

@colors is now ("yellow", "magenta", "cyan", "black").

The sort function, by default, compares the ASCII values of the array elements (see http://www.cgi101.com/book/ch2/ascii.html for the chart of ASCII values). This means if you try to sort a list of numbers, you get "12" before "2". You can do a true numeric sort like so:

```
my @numberlist = (8, 4, 3, 12, 7, 15, 5);
my @sortednumberlist = sort( {$a <=> $b;} @numberlist);
```

{ $a <=> $b; } is actually a small subroutine, embedded right in your code, that gets called for each pair of items in the array. It compares the first number ($a) to the second number ($b) and returns a number indicating whether $a is greater than, equal to, or less than $b.

This is done repeatedly with all the numbers in the array until the array is completely sorted.

Joining Array Elements Into A String

You can merge an array into a single string using the join function:

```
my @colors = ("cyan", "magenta", "yellow", "black");
my $colorstring = join(", ",@colors);
```

This joins @colors into a single string variable ($colorstring), with each element of the @colors array combined and separated by a comma and a space. In this example $colorstring becomes "cyan, magenta, yellow, black".

You can use any string (including the empty string) as the separator. The separator is the first argument to the join function:

```
join(separator, list);
```

The opposite of join is split, which splits a string into a list of values. See Chapter 7 for more on split.

Array or List?

In general, any function or syntax that works for arrays will also work for a list of values:

```
my $color = ("red", "green", "blue")[1];
# $color is "green"

my $colorstring = join(", ", ("red", "green", "blue"));
# $colorstring is now "red, green, blue"

my ($first, $second, $third) = sort("red", "green", "blue");
# $first is "blue", $second is "green", $third is "red"
```

Hashes

A hash is a special kind of array — an associative array, or paired list of elements. Each pair consists of a string key and a data value.

Perl hash names are prefixed with a percent sign (%). Here's how they're defined:

```
Hash Name        key     value

my %colors = (  "red",   "#ff0000",
                "green", "#00ff00",
                "blue",  "#0000ff",
                "black", "#000000",
                "white", "#ffffff" );
```

This particular example creates a hash named %colors which stores the RGB HEX values for the named colors. The color names are the hash keys; the hex codes are the hash values.

Remember that there's more than one way to do things in Perl, and here's the other way to define the same hash:

```
Hash Name        key       value
my %colors = (   red   => "#ff0000",
                 green => "#00ff00",
                 blue  => "#0000ff",
                 black => "#000000",
                 white => "#ffffff" );
```

The => operator automatically quotes the left side of the argument, so enclosing quotes around the key names are not needed.

To refer to the individual elements of the hash, you'll do:

```
$colors{'red'}
```

Here, "red" is the key, and $colors{'red'} is the value associated with that key. In this case, the value is "#ff0000".

You don't usually need the enclosing quotes around the value, either; $colors{red} also works if the key name doesn't contain characters that are also Perl operators (things like +, -, =, * and /).

To print out all the values in a hash, you can use a foreach loop:

```
foreach my $color (keys %colors) {
   print "$colors{$color}=$color\n";
}
```

This example uses the keys function, which returns a list of the keys of the named hash. One drawback is that keys %hashname will return the keys in unpredictable order — in this example, keys %colors could return ("red", "blue", "green", "black", "white") or ("red", "white", "green", "black", "blue") or any combination thereof. If you want to print out the hash in exact order, you have to specify the keys in the foreach loop:

```
foreach my $color ("red","green","blue","black","white") {
   print "$colors{$color}=$color\n";
}
```

Let's write a CGI program using the colors hash. Start a new file called colors.cgi:

Program 2-2: colors.cgi - Print Hash Variables Program

```
#!/usr/bin/perl -wT
use CGI qw(:standard);
use CGI::Carp qw(warningsToBrowser fatalsToBrowser);
use strict;

# declare the colors hash:
my %colors = (       red => "#ff0000", green=> "#00ff00",
    blue => "#0000ff",           black => "#000000",
    white => "#ffffff" );

# print the html headers
print header;
print start_html("Colors");

foreach my $color (keys %colors) {
    print "<font color=\"$colors{$color}\">$color</font>\n";
}
print end_html;
```

Save it and chmod 755 colors.cgi, then test it in your web browser.

Notice we've had to add backslashes to escape the quotes in this double-quoted string:

```
print "<font color=\"$colors{$color}\">$color</font>\n";
```

A better way to do this is to use Perl's qq operator:

```
print qq(<font color="$colors{$colors}">$color</font>\n);
```

qq creates a double-quoted string for you. And it's much easier to read without all those backslashes in there.

Adding Items to a Hash

To add a new value to a hash, you simply do:

```
$hashname{newkey} = newvalue;
```

Using our colors example again, here's how to add a new value with the key "purple":

```
$colors{purple} = "#ff00ff";
```

If the named key already exists in the hash, then an assignment like this overwrites the previous value associated with that key.

Determining Whether an Item Exists in a Hash

You can use the exists function to see if a particular key/value pair exists in the hash:

```
exists $hashname{key}
```

This returns a true or false value. Here's an example of it in use:

```
if (exists $colors{purple}) {
   print "Sorry, the color purple is already in the hash.<br>\n";
} else {
   $colors{purple} = "#ff00ff";
}
```

This checks to see if the key "purple" is already in the hash; if not, it adds it.

Deleting Items From a Hash

You can delete an individual key/value pair from a hash with the delete function:

```
delete $hashname{key};
```

If you want to empty out the entire hash, do:

```
%hashname = ();
```

Values

We've already seen that the keys function returns a list of the keys of a given hash. Similarly, the values function returns a list of the hash values:

```
my %colors = (red => "#ff0000", green=> "#00ff00",
          blue => "#0000ff", black => "#000000",
          white => "#ffffff" );

my @keyslice = keys %colors;
# @keyslice now equals a randomly ordered list of
# the hash keys:
# ("red", "green", "blue", "black", "white")

my @valueslice = values %colors;
# @valueslice now equals a randomly ordered list of
# the hash values:
# ("ff0000", "#00ff00", "#0000ff", "#000000", "#ffffff")
```

As with keys, values returns the values in unpredictable order.

Determining Whether a Hash is Empty

You can use the scalar function on hashes as well:

```
scalar($hashname);
```

This returns true or false value — true if the hash contains any key/value pairs. The value returned does not indicate how many pairs are in the hash, however. If you want to find that number, use:

```
scalar keys(%hashname);
```

Here's an example:

```
my %colors = (red => "#ff0000", green=> "#00ff00",
          blue => "#0000ff",    black => "#000000",
          white => "#ffffff" );

my $numcolors = scalar(keys(%colors));
```

```
        print "There are $numcolors in this hash.\n";
```

This will print out "There are 5 colors in this hash."

Functions

The real power of PHP comes from its functions.In PHP, there are more than 700 built-in functions.To keep the script from being executed when the page loads, you can put it into a function. A function will be executed by a call to the function. You may call a function from anywhere within a page.

Create a PHP Function

A function will be executed by a call to the function.

Syntax

```
function functionName()
{
code to be executed;
}
```

PHP function guidelines:

- Give the function a name that reflects what the function does
- The function name can start with a letter or underscore (not a number)

Example

A simple function that writes my name when it is called:

```
 <html>
<body>

<?php

function writeName()

{

echo "Kai Jim Refsnes";
}

echo "My name is ";

writeName();
?>
</body>
</html>
```

Output:

```
My name is Kai Jim Refsnes
```

PHP Functions - Adding parameters

To add more functionality to a function, we can add parameters. A parameter is just like a variable.Parameters are specified after the function name, inside the parentheses.

Example 1

The following example will write different first names, but equal last name:

```
<html>
<body>
<?php function writeName($fname)

{

echo $fname .

" Refsnes.<br />";

}

echo "My name is ";writeName("Kai Jim");

echo "My sister’s name is ";

writeName("Hege");

echo "My brother’s name is";

writeName("Stale");?></body></html>
```

Output:

```
My name is Kai Jim Refsnes.

My sister's name is Hege Refsnes.

My brother's name is Stale Refsnes.
```

Example 2

The following function has two parameters:

```
<html>

<body>

<?php function writeName($fname,$punctuation)

{

echo $fname . " Refsnes" . $punctuation . "<br />";

}

echo "My name is ";

writeName("Kai Jim",".");

 echo "My sister's name is ";

writeName("Hege","!");echo "My brother's name is ";

writeName("Ståle","?");
?>
</body>
</html>
```

Output:

```
My name is Kai Jim Refsnes.
```

My sister’s name is Hege Refsnes!

My brother’s name is Ståle Refsnes?

PHP Functions - Return values

To let a function return a value, use the return statement.

Example

```
<html>
<body>

<?php
function add($x,$y)
{
$total=$x+$y;
return $total;
}

echo "1 + 16 = " . add(1,16);
?>

</body>
</html>
```

Output:

```
1 + 16 = 17
```

Pattern matching

Pattern-Matching Operators

Zoologically speaking, Perl’s pattern-matching operators function as a kind of cage for regular expressions, to keep them from getting out. This is by design; if we were to let the regex beasties wander throughout the language, Perl would be a total jungle. The world needs its jungles, of course--they’re the engines of biological diversity, after all--but jungles should stay where they belong. Similarly, despite being the engines of combinatorial diversity, regular expressions should stay inside pattern match operators where they belong. It's a jungle in there.

As if regular expressions weren't powerful enough, the m// and s/// operators also provide the (likewise confined) power of double-quote interpolation. Since patterns are parsed like double-quoted strings, all the normal double-quote conventions will work, including variable interpolation (unless you use single quotes as the delimiter) and special characters indicated with backslash escapes. (See "Specific Characters" later in this chapter.) These are applied before the string is interpreted as a regular expression. (This is one of the few places in the Perl language where a string undergoes more than one pass of processing.) The first pass is not quite normal double-quote interpolation, in that it knows what it should interpolate and what it should pass on to the regular expression parser. So, for instance, any $ immediately followed by a vertical bar, closing parenthesis, or the end of the string will be treated not as a variable interpolation, but as the traditional regex assertion meaning end-of-line. So if you say:

```
$foo = "bar";
/$foo$/;
```

the double-quote interpolation pass knows that those two $ signs are functioning differently. It does the interpolation of $foo, then hands this to the regular expression parser:

```
/bar$/;
```

Another consequence of this two-pass parsing is that the ordinary Perl tokener finds the end of the regular expression first, just as if it were looking for the terminating delimiter of an ordinary string. Only after it has found the end of the string (and done any variable interpolation) is the pattern treated as a regular expression. Among other things, this means you can't "hide" the terminating delimiter of a pattern inside a regex construct (such as a character class or a regex comment, which we haven't covered yet). Perl will see the delimiter wherever it is and terminate the pattern at that point.

You should also know that interpolating variables into a pattern slows down the pattern matcher, because it feels it needs to check whether the variable has changed, in case it has to recompile the pattern (which will slow it down even further). See "Variable Interpolation" later in this chapter.

The tr/// transliteration operator does not interpolate variables; it doesn't even use regular expressions! (In fact, it probably doesn't belong in this chapter at all, but we couldn't think of a better place to put it.) It does share one feature with m// and s///, however: it binds to variables using the =~ and !~ operators.

The =~ and !~ operators, described in Chapter 3, "Unary and Binary Operators", bind the scalar expression on their lefthand side to one of three quote-like operators on their right: m// for matching a pattern, s/// for substituting some string for a substring matched by a pattern, and tr/// (or its synonym, y///) for transliterating one set of characters to another set. (You may write m// as //, without the m, if slashes are used for the delimiter.) If the righthand side of =~ or !~ is none of these three, it still counts as a m// matching operation, but there'll be no place to put any trailing modifiers (see "Pattern Modifiers" later), and you'll have to handle your own quoting:

```
print "matches" if $somestring =~ $somepattern;
```
Really, there's little reason not to spell it out explicitly:
```
print "matches" if $somestring =~ m/$somepattern/;
```
When used for a matching operation, =~ and !~ are sometimes pronounced "matches" and "doesn't match" respectively (although "contains" and "doesn't contain" might cause less confusion).

Apart from the m// and s/// operators, regular expressions show up in two other places in Perl. The first argument to the split function is a special match operator specifying what not to return when breaking a string into multiple substrings. See the description and examples for split in Chapter 29, "Functions". The qr// ("quote regex") operator also specifies a pattern via a regex, but it doesn't try to match anything (unlike m//, which does). Instead, the compiled form of the regex is returned for future use. See "Variable Interpolation" for more information.

You apply one of the m//, s///, or tr/// operators to a particular string with the =~ binding operator (which isn't a real operator, just a kind of topicalizer, linguistically speaking). Here are some examples:

```
$haystack =~ m/needle/              # match a simple pattern
$haystack =~  /needle/              # same thing

$italiano =~ s/butter/olive oil/      # a healthy substitution

$rotate13 =~ tr/a-zA-Z/n-za-mN-ZA-M/  # easy encryption (to break)
```
Without a binding operator, $_ is implicitly used as the "topic":
```
/new life/ and             # search in $_ and (if found)
   /new civilizations/     #    boldly search $_ again

s/sugar/aspartame/         # substitute a substitute into $_

tr/ATCG/TAGC/              # complement the DNA stranded in $_
```

Because s/// and tr/// change the scalar to which they're applied, you may only use them on valid lvalues:

```
"onshore" =~ s/on/off/;      # WRONG: compile-time error
```

However, m// works on the result of any scalar expression:

```
if ((lc $magic_hat->fetch_contents->as_string) =~ /rabbit/) {
   print "Nyaa, what's up doc?\n";
}
else {
   print "That trick never works!\n";
}
```

But you have to be a wee bit careful, since =~ and !~ have rather high precedence--in our previous example the parentheses are necessary around the left term.[3] The !~ binding operator works like =~, but negates the logical result of the operation:

```
if ($song !~ /words/) {
   print qq/"$song" appears to be a song without words.\n/;
}
```

Since m//, s///, and tr/// are quote operators, you may pick your own delimiters. These work in the same way as the quoting operators q//, qq//, qr//, and qw// (see the section Section 5.6.3, "Pick Your Own Quotes" in Chapter 2, "Bits and Pieces").

```
$path =~ s#/tmp#/var/tmp/scratch#;

if ($dir =~ m[/bin]) {
   print "No binary directories please.\n";
}
```

When using paired delimiters with s/// or tr///, if the first part is one of the four customary bracketing pairs (angle, round, square, or curly), you may choose different delimiters for the second part than you chose for the first:

```
s(egg)<larva>;
s{larva}{pupa};
s[pupa]/imago/;
```

Whitespace is allowed in front of the opening delimiters:

```
s (egg)   <larva>;
s {larva} {pupa};
s [pupa]  /imago/;
```

Each time a pattern successfully matches (including the pattern in a substitution), it sets the $‘, $&, and $’ variables to the text left of the match, the whole match, and the text right of the match. This is useful for pulling apart strings into their components:

```
"hot cross buns" =~ /cross/;
print "Matched: <$‘> $& <$’>\n";    # Matched: <hot > cross < buns>
print "Left:    <$‘>\n";            # Left:    <hot >
print "Match:   <$&>\n";            # Match:   <cross>
```

print "Right: <$’>\n"; # Right: < buns>

For better granularity and efficiency, use parentheses to capture the particular portions that you want to keep around. Each pair of parentheses captures the substring corresponding to the subpattern in the parentheses. The pairs of parentheses are numbered from left to right by the positions of the left parentheses; the substrings corresponding to those subpatterns are available after the match in the numbered variables, $1, $2, $3, and so on:[4]

$_ = "Bilbo Baggins’s birthday is September 22";
/(.*)’s birthday is (.*)/;
print "Person: $1\n";
print "Date: $2\n";

$‘, $&, $’, and the numbered variables are global variables implicitly localized to the enclosing dynamic scope. They last until the next successful pattern match or the end of the current scope, whichever comes first. More on this later, in a different scope.

[3] Without the parentheses, the lower-precedence lc would have applied to the whole pattern match instead of just the method call on the magic hat object.

[4] Not $0, though, which holds the name of your program.

Once Perl sees that you need one of $‘, $&, or $’ anywhere in the program, it provides them for every pattern match. This will slow down your program a bit. Perl uses a similar mechanism to produce $1, $2, and so on, so you also pay a price for each pattern that contains capturing parentheses. (See "Clustering" to avoid the cost of capturing while still retaining the grouping behavior.) But if you never use $‘$&, or $’, then patterns without capturing parentheses will not be penalized. So it’s usually best to avoid $‘, $&, and $’ if you can, especially in library modules. But if you must use them once (and some algorithms really appreciate their convenience), then use them at will, because you’ve already paid the price. $& is not so costly as the other two in recent versions of Perl.

File input and output

As you start to program more advanced CGI applications, you’ll want to store data so you can use it later. Maybe you have a guestbook program and want to keep a log of the names and email addresses of visitors, or a page counter that must update a counter file, or a program that scans a flat-file database and draws info from it to generate a page. You can do this by reading and writing data files (often called file I/O).

File Permissions

Most web servers run with very limited permissions; this protects the server (and the system it’s running on) from malicious attacks by users or web visitors. On Unix systems, the web process runs under its own userid, typically the "web" or "nobody" user. Unfortunately this means the server doesn’t have permission to create files in your directory. In order to write to a data file, you must usually make the file (or the directory where the file will be created) world-writable — or at least writable by the web process userid. In Unix a file can be made world-writable using the **chmod** command:

```
chmod 666 myfile.dat
```

To set a directory world-writable, you’d do:

```
chmod 777 directoryname
```

See Appendix A for a chart of the various chmod permissions.

Unfortunately, if the file is world-writable, it can be written to (or even deleted) by other users on the system. You should be very cautious about creating world-writable files in your web space, and you should never create a world-writable directory there. (An attacker could use this to install their own CGI programs there.) If you must have a world-writable directory, either use /tmp (on Unix), or a directory outside of your web space. For example if your web pages are in /home/you/public_html, set up your writable files and directories in /home/you.

A much better solution is to configure the server to run your programs with your userid. Some examples of this are CGIwrap (platform independent) and suEXEC (for Apache/Unix). Both of these force CGI programs on the web server to run under the program owner’s userid and permissions. Obviously if your CGI program is running with your userid, it will be able to create, read and write files in your directory without needing the files to be world-writable.

The Apache web server also allows the webmaster to define what user and group the server runs under. If you have your own domain, ask your webmaster to set up your domain to run under your own userid and group permissions.

Permissions are less of a problem if you only want to read a file. If you set the file permissions so that it is group- and world-readable, your CGI programs can then safely read from that file. Use caution, though; if your program can read the file, so can the webserver, and if the file is in your webspace, someone can type the direct URL and view the contents of the file. Be sure not to put sensitive data in a publicly readable file.

Opening Files

Reading and writing files is done by opening a file and associating it with a filehandle. This is done with the statement:

```
open(filehandle,filename);
```

The filename may be prefixed with a >, which means to overwrite anything that's in the file now, or with a >>, which means to append to the bottom of the existing file. If both > and >> are omitted, the file is opened for reading only. Here are some examples:

```
open(INF,"out.txt");          # opens mydata.txt for reading
open(OUTF,">out.txt");        # opens out.txt for overwriting
open(OUTF,">>out.txt");       # opens out.txt for appending
open(FH, "+<out.txt");        # opens existing file out.txt for reading AND writing
```

The filehandles in these cases are INF, OUTF and FH. You can use just about any name for the filehandle.

Also, a warning: your web server might do strange things with the path your programs run under, so it's possible you'll have to use the full path to the file (such as /home/you/public_html/somedata.txt), rather than just the filename. This is generally not the case with the Apache web server, but some other servers behave differently. Try opening files with just the filename first (provided the file is in the same directory as your CGI program), and if it doesn't work, then use the full path.

One problem with the above code is that it doesn't check the return value of open to ensure the file was really opened. open returns nonzero upon success, or undef (which is a false value) otherwise. The safe way to open a file is as follows:

```
open(OUTF,">outdata.txt") or &dienice("Can't open outdata.txt for writing: $!");
```

This uses the "dienice" subroutine we wrote in Chapter 4 to display an error message and exit if the file can't be opened. You should do this for all file opens, because if you don't, your CGI program will continue running even if the file isn't open, and you could end up losing data. It can be quite frustrating to realize you've had a survey running for several weeks while no data was being saved to the output file.

The $! in the above example is a special Perl variable that stores the error code returned by the failed open statement. Printing it may help you figure out why the open failed.

Guestbook Form with File Write

Let’s try this by modifying the guestbook program you wrote in Chapter 4. The program already sends you e-mail with the information; we’re going to have it write its data to a file as well.

First you’ll need to create the output file and make it writable, because your CGI program probably can’t create new files in your directory. If you’re using Unix, log into the Unix shell, **cd** to the directory where your guestbook program is located, and type the following:

```
touch guestbook.txt
chmod 622 guestbook.txt
```

The Unix **touch** command, in this case, creates a new, empty file called "guestbook.txt". (If the file already exists, touch simply updates the last-modified timestamp of the file.) The chmod 622 command makes the file read/write for you (the owner), and write-only for everyone else.

If you don’t have Unix shell access (or you aren’t using a Unix system), you should create or upload an empty file called guestbook.txt in the directory where your guestbook.cgi program is located, then adjust the file permissions on it using your FTP program.

Now you’ll need to modify guestbook.cgi to write to the file:

Program 6-1: guestbook.cgi - Guestbook Program With File Write

```
#!/usr/bin/perl -wT
use CGI qw(:standard);
use CGI::Carp qw(warningsToBrowser fatalsToBrowser);
use strict;

print header;
print start_html("Results");

# first print the mail message...

$ENV{PATH} = "/usr/sbin";
open (MAIL, "|/usr/sbin/sendmail -oi -t -odq") or
  &dienice("Can’t fork for sendmail: $!\n");
print MAIL "To: recipient\@cgi101.com\n";
print MAIL "From: nobody\@cgi101.com\n";
print MAIL "Subject: Form Data\n\n";
foreach my $p (param()) {
   print MAIL "$p = ", param($p), "\n";
}
close(MAIL);

# now write (append) to the file

open(OUT, ">>guestbook.txt") or &dienice("Couldn't open output file: $!");
foreach my $p (param()) {
   print OUT param($p), "|";
}
print OUT "\n";
close(OUT);

print <<EndHTML;
<h2>Thank You</h2>
<p>Thank you for writing!</p>
<p>Return to our <a href="index.html">home page</a>.</p>
EndHTML

print end_html;

sub dienice {
   my($errmsg) = @_;
   print "<h2>Error</h2>\n";
   print "<p>$errmsg</p>\n";
   print end_html;
   exit;
}
```

Now go back to your browser and fill out the guestbook form again. If your CGI program runs without any errors, you should see data added to the guestbook.txt file. The resulting file will show the submitted form data in pipe-separated form:

```
Someone|someone@wherever.com|comments here
```

Ideally you'll have one line of data (or record) for each form that is filled out. This is what's called a flat-file database.

Unfortunately if the visitor enters multiple lines in the comments field, you'll end up with multiple lines in the data file. To remove the newlines, you should substitute newline characters (\n) as well as hard returns (\r). Perl has powerful pattern matching and replacement capabilities; it can match the most complex patterns in a string using regular expressions (see Chapter 13). The basic syntax for substitution is:

```
$mystring =~ s/pattern/replacement/;
```

This command substitutes "pattern" for "replacement" in the scalar variable $mystring. Notice the operator is a =~ (an equals sign followed by a tilde); this is Perl's binding operator and indicates a regular expression pattern match/substitution/replacement is about to follow.

Here is how to replace the end-of-line characters in your guestbook program:

```
foreach my $p (param()) {
   my $value = param($p);
   $value =~ s/\n/ /g;     # replace newlines with spaces
   $value =~ s/\r//g;      # remove hard returns
   print OUT "$p = $value,";
}
```

Go ahead and change your program, then test it again in your browser. View the guestbook.txt file in your browser or in a text editor and observe the results.

File Locking

CGI processes on a Unix web server can run simultaneously, and if two programs try to open and write the same file at the same time, the file may be erased, and you'll lose all of your data. To prevent this, you need to lock the files you are writing to. There are two types of file locks:

- A shared lock allows more than one program (or other process) to access the file at the same time. A program should use a shared lock when reading from a file.
- An exclusive lock allows only one program or process to access the file while the lock is held. A program should use an exclusive lock when writing to a file.

File locking is accomplished in Perl using the Fcntl module (which is part of the standard library), and the flock function. The use statement is like CGI.pm's:

```
use Fcntl qw(:flock);
```

The Fcntl module provides symbolic values (like abbreviations) representing the correct lock numbers for the flock function, but you must specify: flock in the use statement in order for Fctnl to export those values. The values are as follows:

LOCK_SH	shared lock
LOCK_EX	exclusive lock
LOCK_NB	non-blocking lock
LOCK_UN	unlock

These abbreviations can then be passed to flock. The flock function takes two arguments: the filehandle and the lock type, which is typically a number. The number may vary depending on what operating system you are using, so it's best to use the symbolic values provided by Fcntl. A file is locked after you open it (because the filehandle doesn't exist before you open the file):

```
open(FH, "filename") or &dienice("Can"t open file: $!");
flock(FH, LOCK_SH);
```

The lock will be released automatically when you close the file or when the program finishes.

Keep in mind that file locking is only effective if all of the programs that read and write to that file also use flock. Programs that don't will ignore the locks held by other processes.

Since flock may force your CGI program to wait for another process to finish writing to a file, you should also reset the file pointer, using the seek function:

```
seek(filehandle, offset, whence);
```

offset is the number of bytes to move the pointer, relative to whence, which is one of the following:

0	beginning of file
1	current file position
2	end of file

So seek(OUTF,0,2) repositions the pointer to the end of the file. If you were reading the file instead of writing to it, you'd want to do seek(OUTF,0,0) to reset the pointer to the beginning of the file.

The Fcntl module also provides symbolic values for the seek pointers:

SEEK_SET	beginning of file
SEEK_CUR	current file position
SEEK_END	end of file

To use these, add :seek to the use Fcntl statement:

```
use Fcntl qw(:flock :seek);
```

Now you can use seek(OUTF,0,SEEK_END) to reset the file pointer to the end of the file, or seek(OUTF,0,SEEK_SET) to reset it to the beginning of the file.

Closing Files

When you're finished writing to a file, it's best to close the file, like so:

```
close(filehandle);
```

Files are automatically closed when your program ends. File locks are released when the file is closed, so it is not necessary to actually unlock the file before closing it. (In fact, releasing the lock before the file is closed can be dangerous and cause you to lose data.)

Reading Files

There are two ways you can handle reading data from a file: you can either read one line at a time, or read the entire file into an array. Here's an example:

```
open(FH,"guestbook.txt") or &dienice("Can't open guestbook.txt: $!");

my $a = <FH>;     # reads one line from the file into
                  # the scalar $a
my @b = <FH>;     # reads the ENTIRE FILE into array @b

close(FH);     # closes the file
```

If you were to use this code in your program, you'd end up with the first line of guestbook.txt being stored in $a, and the remainder of the file in array @b (with each element of @b containing one line of data from the file). The actual read occurs with <filehandle>; the amount of data read depends on the type of variable you save it into.

The following section of code shows how to read the entire file into an array, then loop through each element of the array to print out each line:

```
open(FH,"guestbook.txt") or &dienice("Can"t open guestbook.txt: $!");
my @ary = <FH>;
close(FH);

foreach my $line (@ary) {
   print $line;
}
```

This code minimizes the amount of time the file is actually open. The drawback is it causes your CGI program to consume as much memory as the size of the file. Obviously for very large files that's not a good idea; if your program consumes more memory than the machine has available, it could crash the whole machine (or at the very least make things extremely slow). To process data from a very large file, it's better to use a while loop to read one line at a time:

```
open(FH,"guestbook.txt") or &dienice("Can"t open guestbook.txt: $!");
while (my $line = <FH>) {
   print $line;
}
close(FH);
```

Poll Program

Let's try another example: a web poll. You've probably seen them on various news sites. A basic poll consists of one question and several potential answers (as radio buttons); you pick one of the answers, vote, then see the poll results on the next page.

Start by creating the poll HTML form. Use whatever question and answer set you wish.

Program 6-2: poll.html - Poll HTML Form

```
<form action="poll.cgi" method="POST">
Which was your favorite <i>Lord of the Rings</i> film?<br>
<input type="radio" name="pick" value="fotr">The Fellowship of the Ring<br>
<input type="radio" name="pick" value="ttt">The Two Towers<br>
<input type="radio" name="pick" value="rotk">Return of the King<br>
```

```
<input type="radio" name="pick" value="none">I didn’t watch them<br>
<input type="submit" value="Vote">
</form>
<a href="results.cgi">View Results</a><br>
```

In this example we’re using abbreviations for the radio button values. Our CGI program will translate the abbreviations appropriately.

Now the voting CGI program will write the result to a file. Rather than having this program analyze the results, we’ll simply use a redirect to bounce the viewer to a third program (results.cgi). That way you won’t need to write the results code twice.

Here is how the voting program (poll.cgi) should look:

Program 6-3: poll.cgi - Poll Program

```
#!/usr/bin/perl -wT
use CGI qw(:standard);
use CGI::Carp qw(warningsToBrowser fatalsToBrowser);
use strict;
use Fcntl qw(:flock :seek);

my $outfile = "poll.out";

# only record the vote if they actually picked something
if (param(’pick’)) {
  open(OUT, ">>$outfile") or &dienice("Couldn’t open $outfile: $!");
  flock(OUT, LOCK_EX);      # set an exclusive lock
  seek(OUT, 0, SEEK_END);   # then seek the end of file
  print OUT param(’pick’),"\n";
  close(OUT);
} else {
# this is optional, but if they didn’t vote, you might
# want to tell them about it...
  &dienice("You didn’t pick anything!");
}

# redirect to the results.cgi.
# (Change to your own URL...)
print redirect("http://cgi101.com/book/ch6/results.cgi");
```

```
sub dienice {
   my($msg) = @_;
   print header;
   print start_html("Error");
   print h2("Error");
   print $msg;
   print end_html;
   exit;
}
```

Finally results.cgi reads the file where the votes are stored, totals the overall votes as well as the votes for each choice, and displays them in table format.

Program 6-4: results.cgi - Poll Results Program

```
#!/usr/bin/perl -wT
use CGI qw(:standard);
use CGI::Carp qw(warningsToBrowser fatalsToBrowser);
use strict;
use Fcntl qw(:flock :seek);

my $outfile = "poll.out";

print header;
print start_html("Results");

# open the file for reading
open(IN, "$outfile") or &dienice("Couldn't open $outfile: $!");
# set a shared lock
flock(IN, LOCK_SH);
# then seek the beginning of the file
seek(IN, 0, SEEK_SET);

# declare the totals variables
my($total_votes, %results);
# initialize all of the counts to zero:
foreach my $i ("fotr", "ttt", "rotk", "none") {
  $results{$i} = 0;
}

# now read the file one line at a time:
```

```
while (my $rec = <IN>) {
   chomp($rec);
   $total_votes = $total_votes + 1;
   $results{$rec} = $results{$rec} + 1;
}
close(IN);

# now display a summary:
print <<End;
<b>Which was your favorite <i>Lord of the Rings</i> film?
</b><br>
<table border=0 width=50%>
<tr>
  <td>The Fellowship of the Ring</td>
  <td>$results{fotr} votes</td>
</tr>
<tr>
  <td>The Two Towers</td>
  <td>$results{ttt} votes</td>
</tr>
<tr>
  <td>Return of the King</td>
  <td>$results{rotk} votes</td>
</tr>
<tr>
  <td>didn't watch them</td>
  <td>$results{none} votes</td>
</tr>
</table>
<p>
$total_votes votes total
</p>
End

print end_html;

sub dienice {
    my($msg) = @_;
    print h2("Error");
    print $msg;
    print end_html;
```

```
        exit;
    }
```

The Common Gateway Interface

The **Common Gateway Interface** (**CGI**) is a standard (see RFC3875: CGI Version 1.1) that defines how webserver software can delegate the generation of webpages to a console application. Such applications are known as CGI scripts; they can be written in any programming language, although scripting languages are often used. In simple words the CGI provides an interface between the webservers and the clients.

Purpose

The task of a webserver is to respond to requests for webpages issued by clients (usually web browsers) by analyzing the content of the request (which is mostly in its URL), determining an appropriate document to send in response, and returning it to the client.

If the request identifies a file on disk, the server can just return the file's contents. Alternatively, the document's content can be composed on the fly. One way of doing this is to let a console application compute the document's contents, and tell the web server to use that console application. CGI specifies which information is communicated between the webserver and such a console application, and how.

The webserver software will invoke the console application as a command. CGI defines how information about the request (such as the URL) is passed to the command in the form of arguments and environment variables. The application is supposed to write the output document to standard output; CGI defines how it can pass back extra information about the output (such as the MIME type, which defines the type of document being returned) by prepending it with headers.

CGI linkage

CGI programs often are stored in a directory named cgi-bin
- Some CGI programs are in machine code, but Perl programs are usually kept in source form, so perl must be run on them
- A source file can be made to be "executable" by adding a line at their beginning that specifies that a language processing program be run on them first
For Perl programs, if the perl system is stored in
/usr/local/bin/perl, as is often is in UNIX
systems, this is
#!/usr/local/bin/perl -w

- An HTML document specifies a CGI program with the hypertext reference attribute, href, of an anchor tag, <a>, as in

```
<a href =
"http://www.cs.uccs.edu/cgi-bin/reply.pl>"
Click here to run the CGI program, reply.pl
</a>
<!-- reply.html - calls a trivial cgi program
-->
<html>
<head>
<title>
HTML to call the CGI-Perl program reply.pl
</title>
</head>
<body>
This is our first CGI-Perl example
<a href =
"http://www.cs.ucp.edu/cgi-bin/reply.pl">
Click here to run the CGI program, reply.pl
</a>
</body>
</html>
```

- The connection from a CGI program back to the requesting browser is through standard output, usually through the server
- The HTTP header needs only the content type, followed by a blank line, as is created with:

```
print "Content-type: text/html \n\n";
#!/usr/local/bin/perl
# reply.pl – a CGI program that returns a
# greeting to the user
print "Content-type: text/html \n\n",
"<html> <head> \n",
"<title> reply.pl example </title>",
" </head> \n", "<body> \n",
"<h1> Greetings from your Web server!",
" </h1> \n </body> </html> \n";
```

Query string format

In World Wide Web, a **query string** is the part of a Uniform Resource Locator (URL) that contains data to be passed to web applications such as CGI programs.

The Mozilla URL location bar showing an URL with the query string title=Main_page&action=raw

When a web page is requested via the Hypertext Transfer Protocol, the server locates a file in its file system based on the requested URL. This file may be a regular file or a program. In the second case, the server may (depending on its configuration) run the program, sending its output as the required page. The query string is a part of the URL which is passed to the program. Its use permits data to be passed from the HTTP client (often a web browser) to the program which generates the web page.

Structure

A typical URL containing a query string is as follows:

```
http://server/path/program?query_string
```

When a server receives a request for such a page, it runs a program (if configured to do so), passing the query_string unchanged to the program. The question mark is used as a separator and is not part of the query string.

A link in a web page may have a URL that contains a query string. However, the main use of query strings is to contain the content of an HTML form, also known as web form. In particular, when a form containing the fields $field_1$, $field_2$, $field_3$ is submitted, the content of the fields is encoded as a query string as follows:

$field_1=value_1\&field_2=value_2\&field_3=value_3...$

- The query string is composed of a series of field-value pairs.
- The field-value pairs are each separated by an equal sign.
- The series of pairs is separated by the ampersand, '&' or semicolon, ';'.

For each field of the form, the query string contains a pair field=value. Web forms may include fields that are not visible to the user; these fields are included in the query string when the form is submitted

This convention is a W3C recommendation. W3C recommends that all web servers support semicolon separators in the place of ampersand separators.

Technically, the form content is only encoded as a query string when the form submission method is GET. The same encoding is used by default when the submission method is POST, but the result is not sent as a query string, that is, is not added to the action URL of the form. Rather, the string is sent as the body of the request.

URL encoding

Main article: URL encoding

Some characters cannot be part of a URL (for example, the space) and some other characters have a special meaning in a URL: for example, the character # can be used to further specify a subsection (or fragment) of a document; the character = is used to separate a name from a value. A query string may need to be converted to satisfy these constraints. This can be done using a schema known as URL encoding.

In particular, encoding the query string uses the following rules:

- Letters (A-Z and a-z), numbers (0-9) and the characters '.', '-', '~' and '_' are left as-is
- SPACE is encoded as '+'
- All other characters are encoded as %FF hex representation with any non-ASCII characters first encoded as UTF-8 (or other specified encoding)

The encoding of SPACE as '+' and the selection of "as-is" characters distinguishes this encoding from RFC 1738.

Example

If a form is embedded in an HTML page as follows:

```
<form action="cgi-bin/test.cgi" method="get">
  <input type="text" name="first">
  <input type="text" name="second">
  <input type="submit">
</form>
```

and the user inserts the strings “this is a field” and “was it clear (already)?” in the two text fields and presses the submit button, the program test.cgi will receive the following query string:

```
first=this+is+a+field&second=was+it+clear+%28already%29%3F
```

If the form is processed on the server by a CGI script, the script may typically receive the query string as an environment variable named QUERY_STRING.

CGI.pm module

CGI.pm is a large and widely used Perl module for programming Common Gateway Interface (CGI) web applications, providing a consistent API for receiving user input and producing HTML or XHTML output. The module is written and maintained by Lincoln D. Stein.

A Sample CGI Page

Here is a simple CGI page, written in Perl using CGI.pm (in object oriented style):

```
#!/usr/bin/perl -w
#
use strict;
use warnings;
use CGI;

my $cgi = CGI->new();

print $cgi->header(’text/html’);
print $cgi->start_html(’A Simple CGI Page’),
$cgi->h1(’A Simple CGI Page’),
$cgi->start_form,
’Name: ’,
$cgi->textfield(’name’), $cgi->br,
’Age: ’,
$cgi->textfield(’age’), $cgi->p,
$cgi->submit(’Submit!’),
$cgi->end_form, $cgi->p,
$cgi->hr;

if ( $cgi->param(’name’) ) {
   print ’Your name is ’, $cgi->param(’name’), $cgi->br;
}
```

```
if ( $cgi->param('age') ) {
    print 'You are ', $cgi->param('age'), ' years old.';
}

print $cgi->end_html;
```

This would print a very simple webform, asking for your name and age, and after having been submitted, redisplaying the form with the name and age displayed below it. This sample makes use of CGI.pm's object-oriented abilities; it can also be done by calling functions directly, without the $cgi->.

Note: in many examples $q, short for query, is used to store a CGI object. As the above example illustrates, this might be very misleading.

Here is another script that produces the same output using CGI.pm's procedural interface:

```
#!/usr/bin/perl
use strict;
use warnings;
use CGI ':standard';

print header,
    start_html('A Simple CGI Page'),
    h1('A Simple CGI Page'),
    start_form,
    'Name: ',
    textfield('name'), br,
    'Age: ',
    textfield('age'), p,
    submit('Submit!'),
    end_form, p,
    hr;

print 'Your name is ', param('name'), br if param 'name';
print 'You are ', param('age'), ' years old.' if param 'age';

print end_html;
```

Cookies

Cookie, also known as a **web cookie**, **browser cookie**, and **HTTP cookie**, is a text string stored by a user's web browser. A cookie consists of one or more name-value pairs containing bits of information, which may be encrypted for information privacy and data security purposes.

The cookie is sent as an HTTP header by a web server to a web browser and then sent back unchanged by the browser each time it accesses that server. A cookie can be used for authentication, session tracking (state maintenance), storing site preferences, shopping cart contents, the identifier for a server-based session, or anything else that can be accomplished through storing textual data.

As text, cookies are not executable. Because they are not executed, they cannot replicate themselves and are not viruses. However, due to the browser mechanism to set and read cookies, they can be used as spyware. Anti-spyware products may warn users about some cookies because cookies can be used to track people—a privacy concern.

Most modern browsers allow users to decide whether to accept cookies, and the time frame to keep them, but rejecting cookies makes some websites unusable.

Uses

Session management

Cookies may be used to maintain data related to the user during navigation, possibly across multiple visits. Cookies were introduced to provide a way to implement a "shopping cart" (or "shopping basket"),[2][3] a virtual device into which users can store items they want to purchase as they navigate throughout the site.

Shopping basket applications today usually store the list of basket contents in a database on the server side, rather than storing basket items in the cookie itself. A web server typically sends a cookie containing a unique session identifier. The web browser will send back that session identifier with each subsequent request and shopping basket items are stored associated with a unique session identifier.

Allowing users to log in to a website is a frequent use of cookies. Typically the web server will first send a cookie containing a unique session identifier. Users then submit their credentials and the web application authenticates the session and allows the user access to services.

Personalization

Cookies may be used to remember the information about the user who has visited a website in order to show relevant content in the future. For example a web server may send a cookie containing the username last used to log in to a web site so that it may be filled in for future visits.

Many websites use cookies for personalization based on users' preferences. Users select their preferences by entering them in a web form and submitting the form to the server. The server encodes the preferences in a cookie and sends the cookie back to the browser. This way, every time the user accesses a page, the server is also sent the cookie where the preferences are stored, and can personalize the page according to the user preferences. For example, the Wikipedia website allows authenticated users to choose the webpage skin they like best; the Google search engine allows users (even non-registered ones) to decide how many search results per page they want to see.

Tracking

Tracking cookies may be used to track internet users' web browsing habits. This can also be done in part by using the IP address of the computer requesting the page or the referrer field of the HTTP header, but cookies allow for a greater precision. This can be done for example as follows:

1. If the user requests a page of the site, but the request contains no cookie, the server presumes that this is the first page visited by the user; the server creates a random string and sends it as a cookie back to the browser together with the requested page;
2. From this point on, the cookie will be automatically sent by the browser to the server every time a new page from the site is requested; the server sends the page as usual, but also stores the URL of the requested page, the date/time of the request, and the cookie in a log file.

By looking at the log file, it is then possible to find out which pages the user has visited and in what sequence. For example, if the log contains some requests done using the cookie id=abc, it can be determined that these requests all come from the same user. The URL and date/time stored with the cookie allows for finding out which pages the user has visited, and at what time.

Third-party cookies and Web bugs, explained below, also allow for tracking across multiple sites. Tracking within a site is typically used to produce usage statistics, while tracking across sites is typically used by advertising companies to produce anonymous user profiles (which are then used to determine what advertisements should be shown to the user).

A tracking cookie may potentially infringe upon the user's privacy but they can be easily removed. Current versions of popular web browsers include options to delete 'persistent' cookies when the application is closed.

Third-party cookies

When viewing a Web page, images or other objects contained within this page may reside on servers besides just the URL shown in your browser. While rendering the page, the browser downloads all these objects. Most modern websites that you view contain information from lots of different sources. For example, if you type www.domain.com into your browser, widgets and advertisements within this page are often served from a different domain source. While this information is being retrieved, some of these sources may set cookies in your browser. First-party cookies are cookies that are set by the same domain that is in your browser's address bar. Third-party cookies are cookies being set by one of these widgets or other inserts coming from a different domain.

Modern browsers, such as Mozilla Firefox, Internet Explorer and Opera, by default, allow third-party cookies, although users can change the settings to block them. There is no inherent security risk of third-party cookies (they do not harm the user's computer) and they make lots of functionality of the web possible, however some internet users disable them because they can be used to track a user browsing from one website to another. This tracking is most often done by on-line advertising companies to assist in targeting advertisements. For example: Suppose a user visits www.domain1.com and an advertiser sets a cookie in the user's browser, and then the user later visits www.domain2.com. If the same company advertises on both sites, the advertiser knows that this particular user who is now viewing www.domain2.com also viewed www.domain1.com in the past and may avoid repeating advertisements. The advertiser does not know anything more about the user than that—they do not know the user's name or address or any other personal information (unless they obtain it from another source such as from the user or by reading another cookie).