PROGRAMMING THE WEB

Subject Code: **06CS73** IA Marks: 25
No. of Lecture Hrs./ Week : 04 Exam Hours : 03
Total No. of Lecture Hrs: 52 Exam Marks: 100

PART – A

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.

6 Hours

UNIT - 2

XHTML – 2: Images; Hypertext Links; Lists; Tables; Forms; Frames; Syntactic differences between HTML and XHTML.

6 Hours

UNIT - 3

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 and

tags; Conflict resolution.

6 Hours

UNIT - 4

JAVASCRIPT: 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.

8 Hours

PART - B

UNIT - 5

JAVASCRIPT AND HTML DOCUMENTS: 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.

6 Hours

UNIT - 6

DYNAMIC DOCUMENTS WITH JAVASCRIPT: 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.

6 Hours

UNIT - 7

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.

6 Hours

UNIT - 8

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.

8 Hours

TEXT BOOK:

1. **Programming the World Wide Web** – Robert W. Sebesta, 4th Edition, Pearson Education, 2008.

REFERENCE BOOKS:

1. **Internet & World Wide Web How to H program** – M. Deitel, P.J. Deitel, A. B. Goldberg, 3rd Edition, Pearson Education / PHI, 2004.
2. **Web Programming Building Internet Applications** – Chris Bates, 3rd Edition, Wiley India, 2006.
3. **The Web Warrior Guide to Web Programming** – Xue Bai et al, Thomson, 2003.

INDEX SHEET

UNIT - 1
Fundamentals of Web, XHTML – 1

1.1 Internet

1.2 WWW

1.3 Web Browsers

1.4 Web Servers

1.5 URLs

1.6 MIME

1.7 HTTP

1.8 Security

1.9 The Web Programmers Toolbox

1.10 XHTML: Origins and evolution of HTML and XHTML

1.11 Basic syntax

1.12 Standard XHTML document structure

1.13 Basic text markup

1.1 Internet

The **Internet** is a global system of interconnected computer networks that use the standard Internet Protocol Suite (TCP/IP) to serve billions of users worldwide. It is a network of networks that consists of millions of private, public, academic, business, and government networks of local to global scope that are linked by a broad array of electronic and optical networking technologies. The Internet carries a vast array of information resources and services, most notably the inter-linked hypertext documents of the World Wide Web (WWW) and the infrastructure to support electronic mail.

Most traditional communications media, such as telephone and television services, are reshaped or redefined using the technologies of the Internet, giving rise to services such as Voice over Internet Protocol (VoIP) and IPTV. Newspaper publishing has been reshaped into Web sites, blogging, and web feeds. The Internet has enabled or accelerated the creation of new forms of human interactions through instant messaging, Internet forums, and social networking sites.

The origins of the Internet reach back to the 1960s when the United States funded research projects of its military agencies to build robust, fault-tolerant and distributed computer networks. This research and a period of civilian funding of a new U.S. backbone by the National Science Foundation spawned worldwide participation in the development of new networking technologies and led to the commercialization of an international network in the mid 1990s, and resulted in the following popularization of countless applications in virtually every aspect of modern human life. As of 2009, an estimated quarter of Earth's population uses the services of the Internet.

1.2 WWW

The **World Wide Web**, abbreviated as **WWW** and commonly known as **the Web**, is a system of interlinked hypertext documents accessed via the Internet. With a web browser, one can view web pages that may contain text, images, videos, and other multimedia and

navigate between them by using hyperlinks. Using concepts from earlier hypertext systems, English engineer and computer scientist Sir Tim Berners-Lee, now the Director of the World Wide Web Consortium, wrote a proposal in March 1989 for what would eventually become the World Wide Web.[1] He was later joined by Belgian computer scientist Robert Cailliau while both were working at CERN in Geneva, Switzerland. In 1990, they proposed using "HyperText [...] to link and access information of various kinds as a web of nodes in which the user can browse at will", and released that web in December.

"The World-Wide Web (W3) was developed to be a pool of human knowledge, which would allow collaborators in remote sites to share their ideas and all aspects of a common project." If two projects are independently created, rather than have a central figure make the changes, the two bodies of information could form into one cohesive piece of work.

1.3 Web Browsers

A **web browser** is a software application for retrieving, presenting, and traversing information resources on the World Wide Web. An information resource is identified by a Uniform Resource Identifier (URI) and may be a web page, image, video, or other piece of content.[1] Hyperlinks present in resources enable users to easily navigate their browsers to related resources.

Although browsers are primarily intended to access the World Wide Web, they can also be used to access information provided by Web servers in private networks or files in file systems. Some browsers can be also used to save information resources to file systems.

Internet Explorer, the most used web browser

1.4 Web Servers

A **web server** is a computer program that delivers (serves) content, such as web pages, using the Hypertext Transfer Protocol (HTTP), over the World Wide Web. The term web server can also refer to the computer or virtual machine running the program. In large commercial deployments, a server computer running a web server can be rack-mounted with other servers to operate a web farm.

The inside and front of a Dell PowerEdge Web server

1.5 URLs

Uniform Resource Locator (**URL**) is a Uniform Resource Identifier (URI) that specifies where an identified resource is available and the mechanism for retrieving it. In popular usage and in many technical documents and verbal discussions it is often incorrectly used as a synonym for URI,[1]. The best-known example of a URL is the "address" of a web page on the World Wide Web, e.g. http://www.example.com.

1.6 MIME

Multipurpose Internet Mail Extensions (**MIME**) is an Internet standard that extends the format of e-mail to support:

- Text in character sets other than ASCII
- Non-text attachments
- Message bodies with multiple parts
- Header information in non-ASCII character sets

MIME's use, however, has grown beyond describing the content of e-mail to describing content type in general, including for the web (see Internet media type).

Virtually all human-written Internet e-mail and a fairly large proportion of automated e-mail is transmitted via SMTP in MIME format. Internet e-mail is so closely associated with the SMTP and MIME standards that it is sometimes called **SMTP/MIME** e-mail.[1]

The content types defined by MIME standards are also of importance outside of e-mail, such as in communication protocols like HTTP for the World Wide Web. HTTP requires that data be transmitted in the context of e-mail-like messages, although the data most often is not actually e-mail.

MIME is specified in six linked RFC memoranda: RFC 2045, RFC 2046, RFC 2047, RFC 4288, RFC 4289 and RFC 2049, which together define the specifications.

1.7 HTTP

The **Hypertext Transfer Protocol** (**HTTP**) is an Application Layer protocol for distributed, collaborative, hypermedia information systems.[1]

HTTP is a request-response standard typical of client-server computing. In HTTP, web browsers or spiders typically act as clients, while an application running on the computer hosting the web site acts as a server. The client, which submits HTTP requests, is also referred to as the user agent. The responding server, which stores or creates resources such as HTML files and images, may be called the origin server. In between the user agent and origin server may be several intermediaries, such as proxies, gateways, and tunnels.

HTTP is not constrained in principle to using TCP/IP, although this is its most popular implementation platform. Indeed HTTP can be "implemented on top of any other protocol on the Internet, or on other networks." HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used.[2]

Resources to be accessed by HTTP are identified using Uniform Resource Identifiers (URIs)—or, more specifically, Uniform Resource Locators (URLs)—using the http or https URI schemes.

Its use for retrieving inter-linked resources, called hypertext documents, led to the establishment of the World Wide Web in 1990 by English physicist Tim Berners-Lee.

The original version of HTTP, designated HTTP/1.0, was revised in HTTP/1.1. One of the characteristics in HTTP/1.0 was that it uses a separate connection to the same server for every document, while HTTP/1.1 can reuse the same connection to download, for

instance, images for the just served page. Hence HTTP/1.1 may be faster as it takes time to set up such connections.

The standards development of HTTP has been coordinated by the World Wide Web Consortium and the Internet Engineering Task Force (IETF), culminating in the publication of a series of Requests for Comments (RFCs), most notably RFC 2616 (June 1999), which defines HTTP/1.1, the version of HTTP in common use.

Support for pre-standard HTTP/1.1 based on the then developing RFC 2068 was rapidly adopted by the major browser developers in early 1996. By March 1996, pre-standard HTTP/1.1 was supported in Netscape 2.0, Netscape Navigator Gold 2.01, Mosaic 2.7, Lynx 2.5, and in Internet Explorer 3.0. End user adoption of the new browsers was rapid. In March 1996, one web hosting company reported that over 40% of browsers in use on the Internet were HTTP 1.1 compliant. That same web hosting company reported that by June 1996, 65% of all browsers accessing their servers were HTTP/1.1 compliant.[3] The HTTP/1.1 standard as defined in RFC 2068 was officially released in January 1997. Improvements and updates to the HTTP/1.1 standard were released under RFC 2616 in June 1999.

1.8 Security

Computer security is a branch of computer technology known as information security as applied to computers and networks. The objective of computer security includes protection of information and property from theft, corruption, or natural disaster, while allowing the information and property to remain accessible and productive to its intended users. The term computer system security means the collective processes and mechanisms by which sensitive and valuable information and services are protected from publication, tampering or collapse by unauthorized activities or untrustworthy individuals and unplanned events respectively. The strategies and methodologies of computer security often differ from most other computer technologies because of its somewhat elusive objective of preventing unwanted computer behavior instead of enabling wanted computer behavior.

The technologies of computer security are based on logic. As security is not necessarily the primary goal of most computer applications, designing a program with security in mind often imposes restrictions on that program's behavior.

There are 4 approaches to security in computing, sometimes a combination of approaches is valid:

1. Trust all the software to abide by a security policy but the software is not trustworthy (this is computer insecurity).
2. Trust all the software to abide by a security policy and the software is validated as trustworthy (by tedious branch and path analysis for example).
3. Trust no software but enforce a security policy with mechanisms that are not trustworthy (again this is computer insecurity).
4. Trust no software but enforce a security policy with trustworthy hardware mechanisms.

Computers consist of software executing atop hardware, and a "computer system" is, by frank definition, a combination of hardware, software (and, arguably, firmware, should one choose so separately to categorize it) that provides specific functionality, to include either an explicitly expressed or (as is more often the case) implicitly carried along security policy. Indeed, citing the Department of Defense Trusted Computer System Evaluation Criteria (the TCSEC, or Orange Book)—archaic though that may be —the inclusion of specially designed hardware features, to include such approaches as tagged architectures and (to particularly address "stack smashing" attacks of recent notoriety) restriction of executable text to specific memory regions and/or register groups, was a sine qua non of the higher evaluation classes, to wit, B2 and above.)

Many systems have unintentionally resulted in the first possibility. Since approach two is expensive and non-deterministic, its use is very limited. Approaches one and three lead to failure. Because approach number four is often based on hardware mechanisms and avoids abstractions and a multiplicity of degrees of freedom, it is more practical.

Combinations of approaches two and four are often used in a layered architecture with thin layers of two and thick layers of four.

There are various strategies and techniques used to design security systems. However there are few, if any, effective strategies to enhance security after design. One technique enforces the principle of least privilege to great extent, where an entity has only the privileges that are needed for its function. That way even if an attacker gains access to one part of the system, fine-grained security ensures that it is just as difficult for them to access the rest.

1.9 XHTML: Origins and evolution of HTML and XHTML

HTML

What is HTML?

HTML is a language for describing web pages.

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- HTML is not a programming language, it is a **markup language**
- A markup language is a set of **markup tags**
- HTML uses **markup tags** to describe web pages

HTML Tags

HTML markup tags are usually called HTML tags

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

Origins

Tim Berners-Lee

In 1980, physicist Tim Berners-Lee, who was a contractor at CERN, proposed and prototyped ENQUIRE, a system for CERN researchers to use and share documents. In 1989, Berners-Lee wrote a memo proposing an Internet-based hypertext system.[2] Berners-Lee specified HTML and wrote the browser and server software in the last part of 1990. In that year, Berners-Lee and CERN data systems engineer Robert Cailliau collaborated on a joint request for funding, but the project was not formally adopted by CERN. In his personal notes,[3] from 1990 he lists[4] "some of the many areas in which hypertext is used", and puts an encyclopedia first.

First specifications

The first publicly available description of HTML was a document called HTML Tags, first mentioned on the Internet by Berners-Lee in late 1991.[5][6] It describes 20 elements comprising the initial, relatively simple design of HTML. Except for the hyperlink tag, these were strongly influenced by SGMLguid, an in-house SGML based documentation format at CERN. Thirteen of these elements still exist in HTML 4.[7]

HTML is a text and image formatting language used by web browsers to dynamically format web pages. Many of the text elements are found in the 1988 ISO technical report TR 9537 Techniques for using SGML, which in turn covers the features of early text formatting languages such as that used by the RUNOFF command developed in the early 1960s for the CTSS (Compatible Time-Sharing System) operating system: these formatting commands were derived from the commands used by typesetters to manually format documents. However the SGML concept of generalized markup is based on elements (nested annotated ranges with attributes) rather than merely point effects, and also the separation of structure and processing: HTML has been progressively moved in this direction with CSS.

Berners-Lee considered HTML to be an application of SGML, and it was formally defined as such by the Internet Engineering Task Force (IETF) with the mid-1993 publication of the first proposal for an HTML specification: "Hypertext Markup Language (HTML)" Internet-Draft by Berners-Lee and Dan Connolly, which included an SGML Document Type Definition to define the grammar.[8] The draft expired after six months, but was notable for its acknowledgment of the NCSA Mosaic browser's custom tag for embedding in-line images, reflecting the IETF's philosophy of basing standards on successful prototypes.[9] Similarly, Dave Raggett's competing Internet-Draft, "HTML+ (Hypertext Markup Format)", from late 1993, suggested standardizing already-implemented features like tables and fill-out forms.[10]

After the HTML and HTML+ drafts expired in early 1994, the IETF created an HTML Working Group, which in 1995 completed "HTML 2.0", the first HTML specification intended to be treated as a standard against which future implementations should be based.[9] Published as Request for Comments 1866, HTML 2.0 included ideas from the HTML and HTML+ drafts.[11] The 2.0 designation was intended to distinguish the new edition from previous drafts.[12]

Further development under the auspices of the IETF was stalled by competing interests. Since 1996, the HTML specifications have been maintained, with input from commercial software vendors, by the World Wide Web Consortium (W3C).[13] However, in 2000, HTML also became an international standard (ISO/IEC 15445:2000). The last HTML specification published by the W3C is the HTML 4.01 Recommendation, published in late 1999. Its issues and errors were last acknowledged by errata published in 2001.

Version history of the standard

HTML version timeline

November 24, 1995

HTML 2.0 was published as IETF RFC 1866. Supplemental RFCs added capabilities:

- November 25, 1995: RFC 1867 (form-based file upload)

- May 1996: RFC 1942 (tables)
- August 1996: RFC 1980 (client-side image maps)
- January 1997: RFC 2070 (internationalization)

In June 2000, all of these were declared obsolete/historic by RFC 2854.

January 1997

HTML 3.2[14] was published as a W3C Recommendation. It was the first version developed and standardized exclusively by the W3C, as the IETF had closed its HTML Working Group in September 1996.[15]

HTML 3.2 dropped math formulas entirely, reconciled overlap among various proprietary extensions, and adopted most of Netscape's visual markup tags. Netscape's blink element and Microsoft's marquee element were omitted due to a mutual agreement between the two companies.[13] A markup for mathematical formulas similar to that in HTML wasn't standardized until 14 months later in MathML.

December 1997

HTML 4.0[16] was published as a W3C Recommendation. It offers three variations:

- Strict, in which deprecated elements are forbidden,
- Transitional, in which deprecated elements are allowed,
- Frameset, in which mostly only frame related elements are allowed;

Initially code-named "Cougar",[17] HTML 4.0 adopted many browser-specific element types and attributes, but at the same time sought to phase out Netscape's visual markup features by marking them as deprecated in favor of style sheets. HTML 4 is an SGML application conforming to ISO 8879 - SGML.[18]

April 1998

HTML 4.0[19] was reissued with minor edits without incrementing the version number.

December 1999

HTML 4.01[20] was published as a W3C Recommendation. It offers the same three variations as HTML 4.0, and its last errata were published May 12, 2001.

May 2000

ISO/IEC 15445:2000[21][22] ("ISO HTML", based on HTML 4.01 Strict) was published as an ISO/IEC international standard. In the ISO this standard falls in the domain of the ISO/IEC JTC1/SC34 (ISO/IEC Joint Technical Committee 1, Subcommittee 34 - Document description and processing languages).[21]

As of mid-2008, HTML 4.01 and ISO/IEC 15445:2000 are the most recent versions of HTML. Development of the parallel, XML-based language XHTML occupied the W3C's HTML Working Group through the early and mid-2000s.

HTML draft version timeline

October 1991

HTML Tags,[5] an informal CERN document listing twelve HTML tags, was first mentioned in public.

June 1992

First informal draft of the HTML DTD, with seven[citation needed] subsequent revisions

November 1992

HTML DTD 1.1 (the first with a version number, based on RCS revisions, which start with 1.1 rather than 1.0), an informal draft

June 1993

Hypertext Markup Language was published by the IETF IIIR Working Group as an Internet-Draft (a rough proposal for a standard). It was replaced by a second version [1] one month later, followed by six further drafts published by IETF itself [2] that finally led to HTML 2.0 in RFC1866

November 1993

HTML+ was published by the IETF as an Internet-Draft and was a competing proposal to the Hypertext Markup Language draft. It expired in May 1994.

April 1995 (authored March 1995)

HTML 3.0 was proposed as a standard to the IETF, but the proposal expired five months later without further action. It included many of the capabilities that were in Raggett's HTML+ proposal, such as support for tables, text flow around figures, and the display of complex mathematical formulas.[25]

W3C began development of its own Arena browser for testing support for HTML 3 and Cascading Style Sheets, but HTML 3.0 did not succeed for several reasons. The draft was considered very large at 150 pages and the pace of browser development, as well as the number of interested parties, had outstripped the resources of the IETF.[13] Browser vendors, including Microsoft and Netscape at the time, chose to implement different subsets of HTML 3's draft features as well as to introduce their own extensions to it.[13] (See Browser wars) These included extensions to control stylistic aspects of documents, contrary to the "belief [of the academic engineering community] that such things as text color, background texture, font size and font face were definitely outside the scope of a language when their only intent was to specify how a document would be organized." Dave Raggett, who has been a W3C Fellow for many years has commented for example, "To a certain extent, Microsoft built its business on the Web by extending HTML features."

January 2008

HTML 5 was published as a Working Draft by the W3C.

Although its syntax closely resembles that of SGML, HTML 5 has abandoned any attempt to be an SGML application, and has explicitly defined its own "html" serialization, in addition to an alternative XML-based XHTML 5 serialization.[27]

XHTML versions

Main article: XHTML

XHTML is a separate language that began as a reformulation of HTML 4.01 using XML 1.0. It continues to be developed:

- XHTML 1.0, published January 26, 2000 as a W3C Recommendation, later revised and republished August 1, 2002. It offers the same three variations as HTML 4.0 and 4.01, reformulated in XML, with minor restrictions.
- XHTML 1.1, published May 31, 2001 as a W3C Recommendation. It is based on XHTML 1.0 Strict, but includes minor changes, can be customized, and is reformulated using modules from Modularization of XHTML, which was published April 10, 2001 as a W3C Recommendation.
- XHTML 2.0,. There is no XHTML 2.0 standard. XHTML 2.0 is incompatible with XHTML 1.x and, therefore, would be more accurate to characterize as an XHTML-inspired new language than an update to XHTML 1.x.
- XHTML 5, which is an update to XHTML 1.x, is being defined alongside HTML 5 in the HTML 5 draft.

XHTML

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

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

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

Origin

XHTML 1.0 is "a reformulation of the three HTML 4 document types as applications of XML 1.0". The World Wide Web Consortium (W3C) also continues to maintain the HTML 4.01 Recommendation, and the specifications for HTML5 and XHTML5 are being actively developed. In the current XHTML 1.0 Recommendation document, as published and revised to August 2002, the W3C commented that, "The XHTML family is the next step in the evolution of the Internet. By migrating to XHTML today, content developers can enter the XML world with all of its attendant benefits, while still remaining confident in their content's backward and future compatibility."[1]

However, in 2004, the Web Hypertext Application Technology Working Group (WHATWG) formed, independently of the W3C, to work on advancing ordinary HTML not based on XHTML. Most major browser vendors were unwilling to implement the features in new W3C XHTML 2.0 drafts, and felt that they didn't serve the needs of modern web development. The WHATWG eventually began working on a standard that supported both XML and non-XML serializations, HTML 5, in parallel to W3C standards such as XHTML 2. In 2007, the W3C's HTML working group voted to officially recognize HTML 5 and work on it as the next-generated HTML standard. In 2009, the W3C allowed the XHTML 2 Working Group's charter to expire, acknowledging that HTML 5 would be the sole next-generation HTML standard, including both XML and non-XML serializations.

Evolution

XHTML 1.0

December 1998 saw the publication of a W3C Working Draft entitled Reformulating HTML in XML. This introduced Voyager, the codename for a new markup language based on HTML 4 but adhering to the stricter syntax rules of XML. By February 1999 the specification had changed name to XHTML 1.0: The Extensible HyperText Markup Language, and in January 2000 it was officially adopted as a W3C Recommendation.[29]

There are three formal DTDs for XHTML 1.0, corresponding to the three different versions of HTML 4.01:

- **XHTML 1.0 Strict** is the XML equivalent to strict HTML 4.01, and includes elements and attributes that have not been marked deprecated in the HTML 4.01 specification.
- **XHTML 1.0 Transitional** is the XML equivalent of HTML 4.01 Transitional, and includes the presentational elements (such as center, font and strike) excluded from the strict version.
- **XHTML 1.0 Frameset** is the XML equivalent of HTML 4.01 Frameset, and allows for the definition of frameset documents—a common Web feature in the late 1990s.

The second edition of XHTML 1.0 became a W3C Recommendation in August 2002.[30]

Modularization of XHTML

Modularization provides an abstract collection of components through which XHTML can be subsetted and extended. The feature is intended to help XHTML extend its reach onto emerging platforms, such as mobile devices and Web-enabled televisions. The initial draft of Modularization of XHTML became available in April 1999, and reached Recommendation status in April 2001.

The first XHTML Family Markup Languages to be developed with this technique were XHTML 1.1 and XHTML Basic 1.0. Another example is XHTML-Print (W3C Recommendation, September 2006), a language designed for printing from mobile devices to low-cost printers.

In October 2008 Modularization of XHTML was superseded by XHTML Modularization 1.1, which adds an XML Schema implementation.

XHTML 1.1—Module-based XHTML

XHTML 1.1 evolved out of the work surrounding the initial Modularization of XHTML specification. The W3C released a first draft in September 1999; Recommendation status was reached in May 2001. The modules combined within XHTML 1.1 effectively recreate XHTML 1.0 Strict, with the addition of ruby annotation elements (ruby, rbc, rtc, rb, rt and rp) to better support East-Asian languages. Other changes include removal of the lang attribute (in favour of xml:lang), and removal of the name attribute from the a and map elements.

Although XHTML 1.1 is largely compatible with XHTML 1.0 and HTML 4, in August 2002 the HTML WG (renamed to XHTML2 WG since) issued a Working Group Note advising that it should not be transmitted with the HTML media type. With limited browser support for the alternate application/xhtml+xml media type, XHTML 1.1 proved unable to gain widespread use. In January 2009 a second edition of the document (XHTML Media Types - Second Edition, not to be confused with the XHTML 1.1 - 2nd ed) was issued, relaxing this restriction and allowing XHTML 1.1 to be served as text/html.

XHTML 1.1 Second Edition (W3C Proposed Edited Recommendation) was issued on 7 May 2009 and rescinded on 19 May 2009. (This does not affect the text/html media type usage for XHTML 1.1 as specified in the: XHTML Media Types - Second Edition)

XHTML Basic and XHTML-MP

To support constrained devices, XHTML Basic was created by the W3C; it reached Recommendation status in December 2000. XHTML Basic 1.0 is the most restrictive version of XHTML, providing a minimal set of features that even the most limited devices can be expected to support.

The Open Mobile Alliance and its predecessor the WAP Forum released three specifications between 2001 and 2006 that extended XHTML Basic 1.0. Known as XHTML Mobile Profile or XHTML-MP, they were strongly focused on uniting the

differing markup languages used on mobile handsets at the time. All provide richer form controls than XHTML Basic 1.0, along with varying levels of scripting support.

XHTML Basic 1.1 became a W3C Recommendation in July 2008, superseding XHTML-MP 1.2. XHTML Basic 1.1 is almost but not quite a subset of regular XHTML 1.1. The most notable addition over XHTML 1.1 is the inputmode attribute—also found in XHTML-MP 1.2—which provides hints to help browsers improve form entry.

XHTML 1.2

The XHTML 2 Working Group is considering the creation of a new language based on XHTML 1.1. If XHTML 1.2 is created, it will include WAI-ARIA and role attributes to better support accessible web applications, and improved Semantic Web support through RDFa. The inputmode attribute from XHTML Basic 1.1, along with the target attribute (for specifying frame targets) may also be present. It's important to note that the XHTML2 WG have not yet been chartered to carry out the development of XHTML1.2 and the W3C has announced that it does not intend to recharter the XHTML2 WG, this means that the XHTML1.2 proposal may not eventuate.

XHTML 2.0

Between August 2002 and July 2006 the W3C released the first eight Working Drafts of XHTML 2.0, a new version of XHTML able to make a clean break from the past by discarding the requirement of backward compatibility. This lack of compatibility with XHTML 1.x and HTML 4 caused some early controversy in the web developer community. Some parts of the language (such as the role and RDFa attributes) were subsequently split out of the specification and worked on as separate modules, partially to help make the transition from XHTML 1.x to XHTML 2.0 smoother. A ninth draft of XHTML 2.0 was expected to appear in 2009, but on July 2, 2009, the W3C decided to let the XHTML2 Working Group charter expire by that year's end, effectively halting any further development of the draft into a standard.

New features introduced by XHTML 2.0 include:

- HTML forms will be replaced by XForms, an XML-based user input specification allowing forms to be displayed appropriately for different rendering devices.
- HTML frames will be replaced by XFrames.
- The DOM Events will be replaced by XML Events, which uses the XML Document Object Model.
- A new list element type, the nl element type, will be included to specifically designate a list as a navigation list. This will be useful in creating nested menus, which are currently created by a wide variety of means like nested unordered lists or nested definition lists.
- Any element will be able to act as a hyperlink, e. g., <li href="articles.html">Articles, similar to XLink. However, XLink itself is not compatible with XHTML due to design differences.
- Any element will be able to reference alternative media with the src attribute, e. g., London Bridge is the same as <object src="lbridge.jpg" type="image/jpeg"> London Bridge </object>.
- The alt attribute of the img element has been removed: alternative text will be given in the content of the img element, much like the object element, e. g., <img src="hms_audacious.jpg">HMS Audacious</img>.
- A single heading element (h) will be added. The level of these headings are determined by the depth of the nesting. This allows the use of headings to be infinite, rather than limiting use to six levels deep.
- The remaining presentational elements i, b and tt, still allowed in XHTML 1.x (even Strict), will be absent from XHTML 2.0. The only somewhat presentational elements remaining will be sup and sub for superscript and subscript respectively, because they have significant non-presentational uses and are required by certain languages. All other tags are meant to be semantic instead (e. g. for **strong or bolded** text) while allowing the user agent to control the presentation of elements via CSS.
- The addition of RDF triple with the property and about attributes to facilitate the conversion from XHTML to RDF/XML.

HTML5—Vocabulary and APIs for HTML5 and XHTML5

Main article: HTML5

HTML5 initially grew independently of the W3C, through a loose group of browser manufacturers and other interested parties calling themselves the WHATWG, or Web Hypertext Application Technology Working Group. The WHATWG announced the existence of an open mailing list in June 2004, along with a website bearing the strapline "Maintaining and evolving HTML since 2004." The key motive of the group was to create a platform for dynamic web applications; they considered XHTML 2.0 to be too document-centric, and not suitable for the creation of internet forum sites or online shops.

In April 2007, the Mozilla Foundation and Opera Software joined Apple in requesting that the newly rechartered HTML Working Group of the W3C adopt the work, under the name of HTML 5. The group resolved to do this the following month, and the First Public Working Draft of HTML 5 was issued by the W3C in January 2008. The most recent W3C Working Draft was published in June 2008.

HTML5 has both a regular text/html serialization and an XML serialization, which is known as XHTML5. In addition to the markup language, the specification includes a number of application programming interfaces. The Document Object Model is extended with APIs for editing, drag-and-drop, data storage and network communication.

The language is more compatible with HTML 4 and XHTML 1.x than XHTML 2.0, due to the decision to keep the existing HTML form elements and events model. It adds many new elements not found in XHTML 1.x, however, such as section and aside. (The XHTML 1.2 equivalent (which (X)HTML5 replaces) of these structural elements would be and .)

As of 2009-09-03, the latest editor's draft includes WAI-ARIA support.

1.11 Basic syntax

Writing valid HTML (or XHTML) is not a terribly difficult task once you know what the rules are, although the rules are slightly more stringent in XHTML than in HTML. The list below provides a quick reference to the rules that will ensure your markup is well-formed and valid. Note that there are other differences between HTML and XHTML which go beyond simple syntax requirements; those differences are covered in HTML Versus XHTML.

The Document Tree

A web page is, at its heart, little more than a collection of HTML elements—the defining structures that signify a paragraph, a table, a table cell, a quote, and so on. The element is created by writing an opening tag, and completed by writing a closing tag. In the case of a paragraph, you'd create a p element by typing Content goes here .

The elements in a web page are contained in a tree structure in which html is the root element that splits into the head and body elements (as explained in Basic Structure of a Web Page). An element may contain other nested elements (although this very much depends on what the parent element is; for example, a p element can contain span, em, or strong elements, among others). Where this occurs, the opening and closing tags must be symmetrical. If an opening paragraph tag is followed by the opening em element, the closing tags must appear in the reverse order, like so: Content goes here, and some of it needs emphasis too . If you were to type Content goes here, and some of it needs emphasis too , you'd have created invalid markup.

Case Sensitivity

In HTML, tag names are case insensitive, but in XHTML they're case sensitive. As such, in HTML, you can write the markup in lowercase, mixed case, or uppercase letters. So this is a paragraph , as is <P>this example</P>, and even <P>this markup would be valid . In XHTML, however, you must use lowercase for markup: This is a valid paragraph in XHTML .

Opening and Closing Tags

In HTML, it's possible to omit some closing tags (check each element's reference to see whether an HTML closing tag is required), so this is valid markup: This is my first paragraph. This is my second paragraph. And here's the last one..

In XHTML, all elements must be closed. Hence the paragraph example above would need to be changed to: This is my first paragraph. This is my second paragraph. And here's the last one.

As well as letting you omit some closing tags, HTML allows you to omit start tags—but only on the html, head, body, and tbody elements. This is not a recommended practice, but is technically possible.

For empty elements such as img, XHTML (that is not served with the application/xhtml+xml) requires us to use the XML empty element syntax: <elementname attribute="attributevalue"/>

If serving the document as application/xhtml+xml, it's also valid to close empty elements using a start and end tag, for example the img element, as <img></img>

Readability Considerations

A browser doesn't care whether you use a single space to separate attributes, ten spaces, or even complete line breaks; it doesn't matter, as long as some space is present. As such, all of the examples below are perfectly acceptable (although the more spaces you include, the larger your web page's file size will be—each occurrence of whitespace takes up additional bytes—so the first example is still the most preferable):

```
<img src="/images/burj.jpg" alt="Burj Al Arab, iconic hotel in
Dubai" class="gallery"/>

<img
    src="/images/burj.jpg"
```

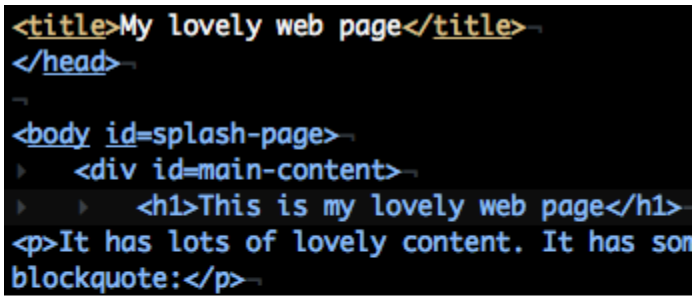
```
    alt="Burj Al Arab, iconic hotel in Dubai"
    class="gallery"
/>

<img

    src="/images/burj.jpg"

    alt="Burj Al Arab, iconic hotel in Dubai"

    class="gallery"/>
```

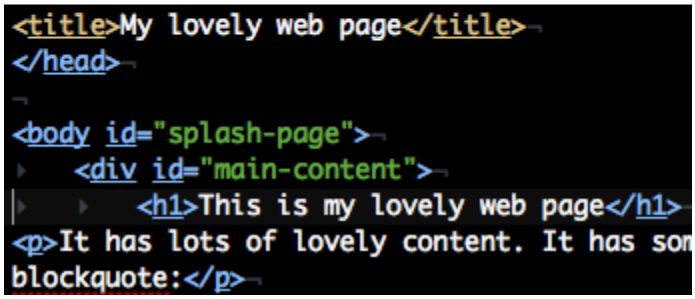
In XHTML all attribute values must be quoted, so you'll need to write class="gallery" rather than class=gallery. It's valid to omit the quotes from your HTML, though it may make reading the markup more difficult for developers revisiting old markup (although this really depends on the developer—it's a subjective thing). It's simply easier always to add quotes, rather than to have to remember in which scenarios attribute values require quotes in HTML, as the following piece of HTML demonstrates:

```
<a href="http://example.org"> needs to be quoted because it contains a /
<a href=index.html> acceptable without quotes in HTML
```

Another reason why it's a good idea always to quote your attributes, even if you're using HTML 4.01, is that your HTML editor may be able to provide syntax coloring that makes the code even easier to scan through. Without the quotes, the software may not be able to identify the difference between elements, attributes, and attribute values. This fact is illustrated in Figure , which shows a comparison between quoted and unquoted syntax coloring in the Mac text editor TextMate.

Figure. TextMate's syntax coloring taking effect to display quoted attributes





Commenting Markup

You may add comments in your HTML, perhaps to make it clear where sections start or end, or to provide a note to remind yourself why you approached the creation of a page in a certain way. What you use comments for isn't important, but the way that you craft a comment is important. The HTML comment looks like this: <!-- this is a comment -->. It's derived from SGML, which starts with an <! and ends with an >; the actual comment is, in effect, inside the opening -- and the closing -- parts. These hyphens tell the browser when to start ignoring text content, and when to start paying attention again. The fact that the double hyphen -- characters signify the beginning and end of the comment means that you should not use double hyphens anywhere inside a comment, even if you believe that your usage of these characters conforms to SGML rules. Single hyphens are allowed, however.

The markup below shows examples of good and bad HTML comments—see the remark associated with each example for more information:

Take the next right.<!-- Look out for the
signpost for 'Castle' --> a valid comment

Take the next right.<!-- Look out for -- Castle -->
not a valid comment; the double dashes in the middle could be misinterpreted as the end of the comment

Take the next right.<!-- Look out for -- -- Castle -->
a valid comment; 'Look out for' is one comment, 'Castle' is another

Take the next right.
<!---------------------------------
This is just asking for trouble. Too
many hyphens! -->
a valid comment; don't use hyphens or <> characters to format comment text

<p <!-- class="lively" -->>Wowzers!
It's not possible to comment out attributes inside an HTML element

1.12 Standard XHTML document structure

An XHTML document consists of three main parts:

- DOCTYPE
- Head
- Body

The basic document structure is:

```
<!DOCTYPE ...>

<html  ... >
<head> ... </head>
```

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

The <head> area contains information about the document, such as ownership, copyright, and keywords; and the <body> area contains the content of the document to be displayed.

Listing 1 shows you how this structure might be used in practice:

Listing 1. An XHTML example

```
1.  <?xml version="1.0"?>
2.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
    Transitional//EN"  "DTD/xhtml1-transitional.dtd">
3.  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
                lang="en">
4.  <head>
    <title>My XHTML Sample Page</title>

    </head>
5.  <body bgcolor="white">
    <center><h1>Welcome to XHTML !</h1></center>
    </body>
6.  </html>
```

Line 1: Since XHTML is HTML expressed in an XML document, it must include the initial XML declaration <?xml version="1.0"?> at the top of the document.

Line 2: XHTML documents must be identified by one of three standard sets of rules. These rules are stored in a separate document called a Document Type Declaration

(DTD), and are utilized to validate the accuracy of the XHTML document structure. The purpose of a DTD is to describe, in precise terms, the language and syntax allowed in XHTML.

Line 3: The second tag in an XHTML document must include the opening <html> tag with the XML namespace identified by the xmlns=http://www.w3.org/1999/xhtml attribute. The XML namespace identifies the range of tags used by the XHTML document. It is used to ensure that names used by one DTD don't conflict with user-defined tags or tags defined in other DTDs.

Line 4: XHTML documents must include a full header area. This area contains the opening <head> tag and the title tags (<title></title>), and is then completed with the closing </head> tag.

Line 5: XHTML documents must include opening and closing <body></body> tags. Within these tags you can place your traditional HTML coding tags. To be XHTML conformant, the coding of these tags must be well-formed.

Line 6: Finally, the XHTML document is completed with the closing </html> tag.

1.13 Basic text markup

A **markup language** is a modern system for annotating a text in a way that is syntactically distinguishable from that text. The idea and terminology evolved from the "marking up" of manuscripts, i.e. the revision instructions by editors, traditionally written with a blue pencil on authors' manuscripts. Examples are typesetting instructions such as those found in troff and LaTeX, and structural markers such as XML tags. Markup is typically omitted from the version of the text which is displayed for end-user consumption. Some markup languages, like HTML have presentation semantics, meaning their specification prescribes how the structured data is to be presented, but other markup languages, like XML, have no predefined semantics.

A well-known example of a markup language in widespread use today is HyperText Markup Language (HTML), one of the document formats of the World Wide Web. HTML is mostly an instance of SGML (though, strictly, it does not comply with all the rules of SGML) and follows many of the markup conventions used in the publishing industry in the communication of printed work between authors, editors, and printers.

UNIT - 2

XHTML – 2

2.1 Images

2.2 Hypertext Links

2.3 Lists

2.4 Tables

2.5 Forms

2.6 Frames

2.7 Syntactic differences between HTML and XHTML

UNIT - 2

XHTML – 2

HTML is defined using the Standard Generalized Markup Language (SGML) which is an ISO standard notation for describing text formatting languages. The Original version of HTML was designed in conjunction with the structure of the web and the first browser. It was designed to specify document structure at a higher and more abstract level. It also defines details of text such as font style, size and color. The use of web became more popular when the first browser MOSAIC was developed and marketed by Netscape. The next browser was Internet explorer by Microsoft.

History of HTML

HTML was the standard developed by Microsoft for web technologies. First HTML standard was HTML 2.0 was released in 1995. Next HTML standard was HTML 3.2 was released in 1997. The latest version of HTML standard is HTML 4.01 was released and approved by W3C in 1999. The XHTML1.0 standard was approved in early 2000 which is a redefinition of HTML 4.01 using XML IE7 and FIREFOX2 (FX2) support XHTML 1.1

2.1 Images

Images can be included to enhance appearance of the document. Most common methods of representing the images are GIF (graphical interchange format and JPEG (joint photographic experts group) format. The former is 8bit color representation whereas the latter is 24 bit color representation. The image tag specifies an image that appears in a document. It has attributes like src which specifies the source of the image.

```
<html xmlns = "http://www.w3.org/1999/xhtml">
<head><title> Images</title></head>
<body>
<h1> Twinkle twinkle
<h2>little star
<h3> how I wonder
<h4>what you are ???
up above the world so high
```

like a diamond in the sky.

</body>

</HTML>

XHTML validation

2.2 Hypertext Links

A hypertext link in a XHTML document acts as a pointer to some resource. It could be an XHTML document anywhere in the web or another place in the document currently displayed. Links that point to another document specifies the address of the document. It could be a filename, complete url, directory path and a filename. Links are specified in an attribute of an anchor tag <a> which is an inline tag. The anchor tag is the source of an link whereas the document is the target of the link. Links facilitate reader to click on links to learn more about a particular subtopic of interest and also return back to the location of the link

If target is in the same document as the link it is specified in the href attribute value by preceding the id with a pound sign(#)

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

<head><title> Images</title></head>

<body>

<h1> Twinkle twinkle

<h2>little star

<h3> how I wonder

<h4>what you are ???

up above the world so high like a diamond in the sky.

<p>

The blue hill image document

<p>

```
</body>
</HTML>
<html xmlns = "http://www.w3.org/1999/xhtml">
<head><title> Images</title></head>
<body>
<h1 id = "twinkle" > Twinkle twinkle </h1>
<h2>little star</h2>
<h3> how I wonder </h3>
<h4 >what you are ???
up above the world so high
like a diamond in the sky.</h4>
<a href = "#twinkle">Which poem
</a>
</body>
</HTML>
```

Twinkle twinkle

little star

how I wonder

what you are ??? up above the world so high like a diamond in the sky.

Which poem

2.3 Lists

XHTML provides simple and effective ways to specify both ordered and unordered lists <ul> are tags for unordered and ordered lists. Each item in a list is specified with an <li> tag. Any tags can appear in a list item including nested lists.

Definition lists

They are used to specify list of terms and their definitions such as glossaries. They are given by <dl> tag which is a block tag. The definitions are specified as the content of <dd> tag and the definition list is given as the content of a <dt> tag

```
<html xmlns = "http://www.w3.org/1999/xhtml">
<head><title> ordered lists</title></head>
<body>
<h3 > Lists of poems</h1>
<ul>
<li> Twinkle twinkle</li>
<li> Baa Baa black sheep</li>
<li> pussy cat </li>
<li> Humpty dumpty</li>
</ul>
</body>
</html>
<body>
<h3 > Lists of poems</h1>
<ol>
<li> Twinkle twinkle</li>
<li> Baa Baa black sheep</li>
<li> pussy cat </li>
<li> Humpty dumpty</li>
</ol>
</body>
```

Lists of poems

1. Twinkle twinkle
2. Baa Baa black sheep
3. pussy cat
4. Humpty dumpty

Lists of poems

- Twinkle twinkle
- Baa Baa black sheep
- pussy cat
- Humpty dumpty

```
head><title> ordered lists</title></head>
<body>
<ol>
<li > Lists of poems
<ol>
<li> Twinkle twinkle</li>
<li> Baa Baa black sheep</li>
<li> pussy cat </li>
<li> Humpty dumpty</li>
</ol>
<ol>
</li>
<li > Lists of stories
<ol>
<li> Thirsty crow</li>
<li> Lion and the mouse</li>
<li> pussy cat </li>
<li> Midas touch</li>
</ol>
</li>
</ol>
```

1. Lists of poems
    1. Twinkle twinkle
    2. Baa Baa black sheep
    3. pussy cat
    4. Humpty dumpty
2. Lists of stories
    1. Thirsty crow
    2. Lion and the mouse
    3. pussy cat
    4. Midas touch

```
<html xmlns = "http://www.w3.org/1999/xhtml">
<head><title> ordered lists</title></head>
<body>
<h3 > Lists of poems </h3>
<dl>
<dt> 111 </dt>
<dd> Twinkle twinkle</dd>
<dt> 222 </dt>
<dd> Pily poly</dd>
<dt> 333 </dt>
<dd> Rudolf the reindeer</dd>
</dl>
</body>
</HTML>
```

Lists of poems

111
: Twinkle twinkle

222
: Pily poly

333
: Rudolf the reindeer

2.4 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 which is a block tag. A border attribute in the <table> tag specifies a border between the cells. Rule specifies the lines that separate 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 in pixels(0 is no border no rules). 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. The empty cell is specified with a table header tag that includes no content <th> </th>.

```
<body>
<table border = "border">
<caption> fruit juice drinks </caption>
<tr>
<th> </th>
<th> Apple </th>
<th> Mango </th>
<th> Strawberry </th>
</tr>
<tr> <th> Breakfast </th>
<th> 0</th>
<th> 1 </th>
<th> 0</th>
</tr>
<tr> <th> lunch </th>
<th> 1</th>
<th> 1 </th>
<th>1 </th>
</tr>
<tr> <th> dinner </th>
```

<th> 0 </th>

<th> 1 </th>

<th> 0 </th></tr><table></body></HTML>

Colspan Rowspan attributes

A table can have two levels column labels and also row 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>

<caption> fruit juice drinks </caption>

<tr>

<td rowspan="2"></td>

<th colspan="3">Juices chart</th>

</tr>

<tr>

<th> </th>

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>

fruit juice drinks

	Apple	Mango	Strawberry
Breakfast	0	1	0
lunch	1	1	1
dinner	0	1	0

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. 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.

<head><title> simple table</title></head>
<body>
<table border = "border">
<caption> align</caption>
<tr align ="center">
<th> </th>
<th> Apple </th>
<th> Mango </th>
<th> Strawberry </th>
</tr>
<tr>
<th> align </th>
<td align="left">left</td>
<td align="center">center</td>
<td align="right">right</td>

```
</tr>
<tr>
<th> valign<br /> <br /></th>
<td >default</td>
<td valign="top">top</td>
<td valign="bottom">bottom</td>
</tr
<table>
</body>
</HTML>
```

align

	Apple	Mango	Strawberry
align	left	center	right
valign	default	top	bottom

2.5 Forms

A form is the usual way information is gotten from a Web 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). All control tags are inline tags. These controls gather information used from user in the form of either text or button selections. Each control has a value given through the user input. Collectively values of all these controls in a form are called the form data. When the Submit button of a form is clicked, the form's values are sent to the Web server for processing.

The <form> tag

All of the widgets, or components of a form are defined in tphe content of a <form> tag which is a block tag. This tag can have many attributes of which the most required attribute is the action. The action attribute specifies the URL of the application on the web server that is to be called when the user clicks the Submit button.

Eg: 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 techniques, get or post, used to pass the form data to the server. get is the default, so if no method attribute is given in the <form> tag, get will be used. The alternative technique is post. With these techniques the form data is encoded into text string on click of submit button Widgets or Controls Many commonly used controls are created with the <input> tag which specifies the kind of control. It is used for the text, passwords, checkboxes, radio buttons and the action buttons Reset and Submit.

The type attribute of <input> specifies the kind of widget being created. Except Reset and Submit all other controls have a name attribute other than type attribute.

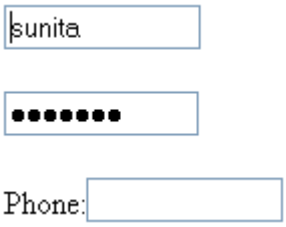
Text: Creates a horizontal box for text input

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 max length, which causes excess input to be ignored

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

If the contents of the textbox should not be displayed when user types it than a password control should be used. Labeling a text box can be done by adding label control to the text box. Both the controls can be encapsulated. This has several advantages. The text content of the label will indicate content of text box and when a label is selected the cursor is implicitly moved to the control.

```
<form action = "">
<p>
<input type = "text" name = "Phone" size = "12" /></p>
<p>
<input type = "password" name = "myPassword" size = "12" maxlength="12"/ >
</p>
<p>
<label>Phone:<input type = "text" name = "Phone" size = "12" /><label>
</p>
```



Checkboxes

Checkboxes 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".

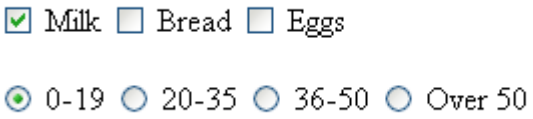
Radio Buttons

Radio buttons are 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. If no button in a radio button group is 'pressed', the browser often 'presses' the first one. Checkboxes and radio buttons are both multiple choice input from the user

```
<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>
<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



Menus

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. 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 beset 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

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.

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>.

A Submit button is required in every form

<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>
<p>
<textarea name = "aspirations" rows = "3”
cols = "40">
(Be brief in expressing your views)
</textarea>
</p>
<p>
<input type = "Submit” value = "Submit Form" />
<input type = "reset" value = "Reset Form" />
</p>
</form>
```

2.6 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>. We 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. A percentage specifies the percentage of total window height for the row. 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. 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)Nested frames - to divide the screen in more interesting ways

<head> <title> Nested frames </title> </head>

<frameset cols = "40%, *">

<frameset rows = "50%, *" >

<frame src = "1my.html" />

<frame src = "12list.html" />

</frameset>

<frameset rows = "20%,35%, *" >

<frame src = "1my.html" />

<frame src = "12list.html" />

<frame src = "2twinkle.html" />

<frame src = "5head.html" />

</frameset>

</frameset>

</html>

2.7 Syntactic differences between HTML and XHTML

Case sensitivity: In HTML, tag and attribute names are case insensitive meaning that <FORM>, <form>,and <Form> are equivalent. In XHTML, all tag and attribute names must be in lowercase.

Closing tags: In HTML, closing tag may be omitted if the processing agent can infer heir presence. For example, in HTML, paragraph elements often do not have closing tags. For example

<p> During Spring, flowers are born. …

<p>

During Fall, flowers die…..

HTML documents are case insensitive whereas XHTML documents have to be in lowercase Closing tags may be omitted in HTML but not in XHTML where all elements must have closing tags except for content tags where it is not a must

<input type= "text" name="address" />

Quoted attribute values :all attribute values must be quoted whether it is numeric or character based

Explicit attribute values: In HTML some attributes are implicit

Quoted attribute values: In HTML, attribute values must be quoted only if there are embedded special characters or white space characters.

Explicit attribute values: In HTML, some attribute values are implicit, that is, they need not be explicitly stated. For example, if the border attribute appears in a <table> tag without a value, it specifies a default width border on the table. For example: <table border>. id and name attributes. HTML markup often uses the name attribute for elements. Element nesting: Although HTML has rules against improper nesting of elements, they are not enforced. Examples of nesting rules are:

1. An anchor element cannot contain another anchor element, and a form element cannot contain another form element.
2. If one element appears inside another element its closing tag should appear before the closing tag of the outer element.
3. Block elements cannot be nested inside inline elements.
4. Text cannot be nested directly in body or form elements.
5. List elements cannot be directly nested in list elements.

UNIT - 3

CSS

3.1 Introduction

3.2 Levels of style sheets

3.3 Style specification formats

3.4 Selector forms

3.5 Property value forms

3.6 Font properties

3.7 List properties

3.8 Color

3.9 Alignment of text

3.10 The Box model

3.11 Background images

3.12 The and tags

3.13 Conflict resolution.

UNIT - 3

CSS

3.1 Introduction

The CSS1 specification was developed in 1996 by W3C. CSS2 was released in 1998 which added many properties and values to CSS1. CSS3 has been under development since the late 1990s. CSSs provide the means to control and change presentation of HTML documents. CSS is not technically HTML, but can be embedded in HTML documents.

Cascading style sheets were introduced to provide a uniform and consistent way to specify presentation details in XHTML documents. Most of the style tags and attributes are those deprecated from HTML 4.0 in favor of style sheets. Idea of style sheets is not a new concept as it existed in desktop publishing systems and word processors. Style sheets allow you to impose a standard style on a whole document, or even a whole collection of documents. Style sheets can be defined at three levels to specify the style of a document .Hence called Cascading style sheets. Style is specified for a tag by the values of its properties.

For example:

<h2 { font-size: 22pt; } >

3.2 Levels of Style Sheets

There are three levels of style sheets, in order from lowest level to highest level, are inline, document level, and external. Inline style sheets are specified for a specific occurrence of a tag and apply only to the content of that tag. This application of style, which defeats the purpose of style sheets – that of imposing uniform style on the tags of at least one whole document. Another disadvantage of inline style sheets is that they result in style information, which is expressed in a expressed in a language distinct from XHTML markup, being embedded in various places in documents.

Document-level style specifications appear in the document head section and apply to the whole body of the document. External style sheets are not part of the documents to which they apply.

They are stored separately and are referenced in all documents that use them. They are written as text files with MIME type text/css.

```
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title> Our first document </title>
<style>
h2 {font-size: 32pt; font-weight: bold;font-family: 'Times New Roman';}
h3,h4 { font-size: 18pt; font-family: 'Courier New'; font-style:italic;
font-weight:bold}
</style>
</head> <body>
<h1> Twinkle twinkle little star</h1>
<h2> how I wonder </h2>
<h3>what you are ???</h3>
<h4>up above the world so high</h4>
<h5> like a diamond</h5>
<h6> in the sky.</h6>
</body>
</html>
```

Twinkle twinkle little star

how I wonder

what you are ???

up above the world so high

like a diamond

in the *sky*.

Linking an External Style sheet

A <link> tag is used to specify that the browser is to fetch and use an external style sheet file through href. The href attribute of <link> is used to specify the URL of the style sheet document, as in the following example:

<link rel = "stylesheet" type = "text/css"

href = “http://www.wherever.org/termpaper.css"> </link>

This link must appear in the head of the document. External style sheets can be validated, with the service http://jigsaw.w3.org/css-validator/ validator-upload.html. External style sheets can be added using other alternate style specification known as file import

- @import url (filename);

Filename is not quoted. Import appears only at the beginning of the content of a style element. The file imported can contain both markup as well as style rules

3.3 Style Specification Formats

The format of a style specification depends on the level of the style sheet. Inline **s**tyle sheet appears as the value of the style attribute of the tag. The general form of which is as follows:

style = "property_1: value_1;

property_2: value_2;

…

property_n: value_n;"

Format for Document-level

Document style specifications appear as the content of a style element within the header of a document, although the format of the specification is quite different from that of inline style sheets. The <style> tag must include the type attribute, set to "text/css“ (as there are other style sheets in JavaScript).

The list of rules must be placed in a comment, because CSS is not XHTML. Style element must be placed within the header of a document. Comments in the rule list must have a different form use C comments (/*…*/). The general form of the content of a style element is as follows:

<style type = "text/css">

/*

rule list(/* styles for paragraphs and other tags*/)

*/</style>

Form of the rules:

Each style in a rule list has two parts: selector, which indicates the tag or tags affected by the rules. Each property/value pair has the form->property: value

Selector { property_1: value_1; property_2: value_2:… property_n: value_n;}

Pairs are separated by semicolons, just as in the value of a <style> tag.

3.4 Selector Forms

Selector can have variety of forms like:

1. Simple selector form
2. Class selector
3. Generic selector
4. Id selector
5. Universal selector
6. Pseudo classes

Simple selector form

Simple selector form is a list of style rules, as in the content of a <style> tag for document-level style sheets. The selector is a tag name or a list of tag names, separated by commas. Consider the following examples, in which the property is font-size and the property value is a number of points :

h1, h3 { font-size: 24pt ;}

h2 { font-size: 20pt ;}

Selectors can also specify that the style should apply only to elements in certain positions in the document .This is done by listing the element hierarchy in the selector.

• Contextual selectors: Selectors can also specify that the style should apply only to elements in certain positions in the document .

• In the eg selector applies its style to the content of emphasis elements that are descendants of bold elements in the body of the document. body b em {font-size: 24pt ;} Also called as descendant selectors. It will not apply to emphasis element not descendant of bold face element.

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 the tag's name with a period.

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,

Once upon a time there lived a king in the place called Ayodhya.

...

Once upon a time there lived a king in the place called Ayodhya.

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 without a tag name in its name.

For Example:

.really-big { … }

Use it as if it were a normal style class

<h1 class = "really-big"> This Tuesday is a holiday </h1>...

 …

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

<head>

<title> Absolute positioning </title>

<style type = "text/css">

.regtext {font-family: Times; font-size: 14pt; width: 600px}

.abstext {position: absolute; top: 25px; left: 50px; font-family: Times; font-size: 24pt; fontstyle:

italic; letter-spacing: 1em; color: rgb(102,102,102); width: 500px}

</style>

</head>

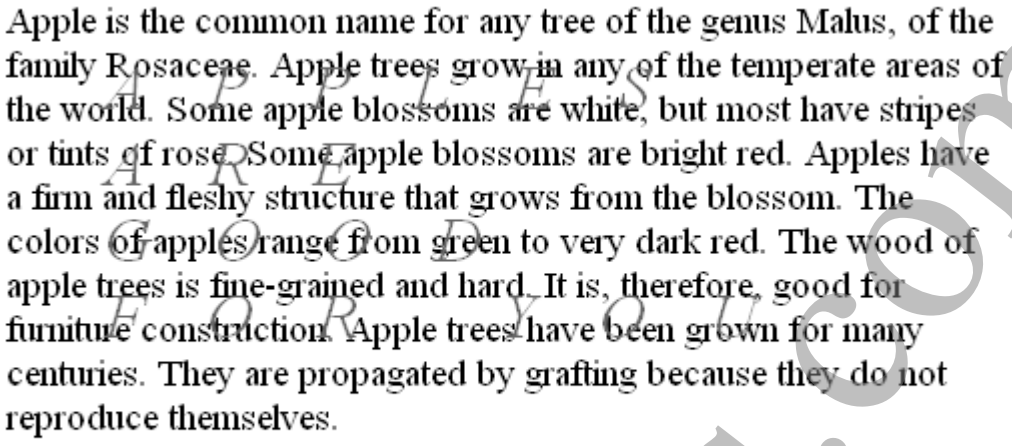
<body>

Apple is the common name for any tree of the genus Malus, of the family Rosaceae. Apple trees grow in any of the temperate areas of the world. Some apple blossoms are white, but most have stripes or tints of rose. Some apple blossoms are bright red. Apples have a firm and fleshy structure that grows from the blossom. The colors of apples range from green to very dark red. The wood of apple trees is fine-grained and hard. It is, therefore, good for furniture construction. Apple trees have been grown for many centuries. They are propagated by grafting because they do not reproduce themselves.

 APPLES ARE GOOD FOR YOU

</body>

</html>



Id Selectors

An id selector allow the application of a style to one specific element. The general form of an id selector is as follows :

#specific-id {property-value list}

Example:

#section14 {font-size: 20} specifies a font size of 20 points to the element

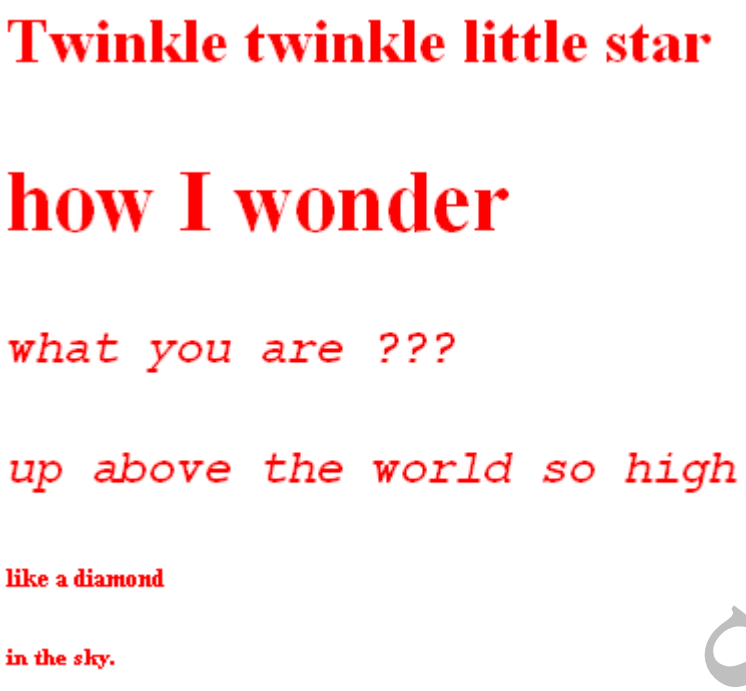
<h2 id =“section14”> Alice in wonderland</h2>

Universal selector

The universal selector, denoted by an asterisk(*), which applies style to all elements in the document. For example:

{color: red;}

makes all elements in the document red.



Pseudo Classes

Pseudo classes are styles that apply when something happens, rather than because the target element simply exists. Names of pseudo classes begin with colons hover classes apply when the mouse cursor is over the element focus classes apply when an element has focus i.e. the mouse cursor is over the element and the left mouse button is clicked. These two pseudo classes are supported by FX2 but IE7 supports only hover.

```
<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>

3.5 Property Values Forms

CSS1 includes 60 different properties in 7 categories(list can be found in W3C website) Fonts, Lists, Alignment of text, Margins, Colors, Backgrounds, Borders. Keywords property values are used when there are only a few possible values and they are predefined

Eg: small, large, medium.

Keyword values are not case sensitive, so Small, SmAlL, and SMALL are all the same as small. Number values can be integer or sequence of digits with decimal points and a + or – sign. Length value are specified as number values that are followed immediately by a two character abbreviation of a unit name. There can be no space between the number and the unit name. The possible unit names are px for pixels, in for inches, cm for centimeters, mm for millimeters, pt for points, pc for picas (12 points),em for value of current font size in pixels, ex for 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: eg: font size set to 85% means new font size will be 85% of the previous font size value.

URL values: URL property values use a form that is slightly different from references to URLs in links. The actual URL, which can be either absolute or relative, is placed in parentheses and preceded by url, as in the following:

url(protocol://server/pathname)

No space should be left between URL and the left parenthesis.

Colors : Color name rgb(n1, n2, n3). Hex form: #B0E0E6 stands for powder blue color. Property values are inherited by all nested tags, unless overridden.

3.6 Font properties

Font-family

The font-family property is used to specify a list of font name. The browser will use the first font in the list that it supports. For example, the following could be specified. font-family: Arial, Helvetica, Courier

Generic fonts: They can be specified as the font family value for example :serif, sans-serif, cursive, fantasy, and monospace (defined in CSS). Browser has a specific font defined for each generic name. If a font name that has more than one word, it should be single-quoted Eg: font-family: 'Times New Roman'

Font-size

Possible values: a length number or a name, such as smaller, xx-large, medium , large etc. Different browsers can use different relative value for the font-size.

Font-variant

The default value of the font-variant property is normal, which specifies the usual character font. This property can be set to small-caps to specify small capital characters.

Font-style

The font-style property is most commonly used to specify italic, as in the following example. Eg: font-style: italic

Font-weights

The font-weight property is used to specify the degree of boldness. For example: font-weight: bold

Font Shorthands

If more than one font property is to be specified than the values may be stated in a list as the value of the font property . The browser will determine from the form of the values which properties to assign. For example, consider the following specification : Eg: font: bold 24pt 'Times New Roman' Palatino Helvetica The order which browser follows is last must be font name, second last font size and then the font style, font variant and font

weight can be in any order but before the font size and names. Only the font size and the font family are required in the font value list. Below example displays the fonts.html

```
<html xmlns = http://www.w3.org/1999/xhtml>
<head> <title> Font properties </title>
<style type = "text/css">
p.big {font-size: 14pt;
font-style: italic;
font-family: 'Times New Roman';
}
p.small {font: 10pt bold 'Courier New';}
h2 {font-family: 'Times New Roman';
font-size: 24pt; font-weight: bold}
h3 {font-family: 'Courier New'; font-size: 18pt}
</style>
</head>
<body>
<p class = "big">
Where there is will there is a way.
</p>
<p class = "small">
Practise makes a man perfect.
</p>
<h2> Chapter 1 Introduction </h2>
<h3> 1.1 The Basics of Web programming </h3>
<h4> Book by Robert Sebesta
</body>
</html>
```

Where there is will there is a way.

Practise makes a man perfect.

Chapter 1 Introduction

1.1 The Basics of Web programming

Book by Robert Sebesta

Text Decoration

The text-decoration property is used to specify some special features of the text. The available values are line-through, overline, underline, and none, which is the default. Many browsers underline links. Text decoration is not inherited Eg:line-through, overline, underline, none

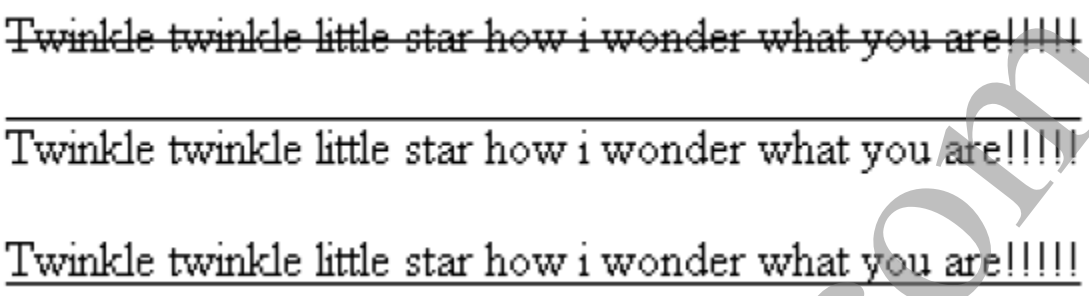
Letter-spacing – value is any length property value controls amount of space between characters in text

• Eg: 3px

```
<html xmlns = "http://www.w3.org/1999/xhtml">
<head> <title> Text decoration </title>
<style type = "text/css">
p.through {text-decoration: line-through}
p.over {text-decoration: overline}
p.under {text-decoration: underline}
</style> </head>
<body>
<p class = "through">
Twinkle twinkle little star how i wonder what you are!!!!! </p>
<p class= "over">
Twinkle twinkle little star how i wonder what you are!!!!! </p>
```

Twinkle twinkle little star how i wonder what you are!!!!!

</body></html>



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

<head>

<title>CSS Example</title>

<link rel="stylesheet" type="text/css" href="ch03_eg01.css" />

</head>

<body>

<h1>Simple CSS Example</h1>

This simple page demonstrates how CSS can be used to control the presentation of an XHTML document.

This paragraph demonstrates the use of the <code>class</code> attribute.

</body>

</html>

/* CSS Document for ch03_eg01.html */

body {

font-family:arial, verdana, sans-serif;

background-color:#efefef;}

h1 {

color:#666666;

font-size:22pt;}

p {

color:#999999;

font-size:10pt;}

p.important {

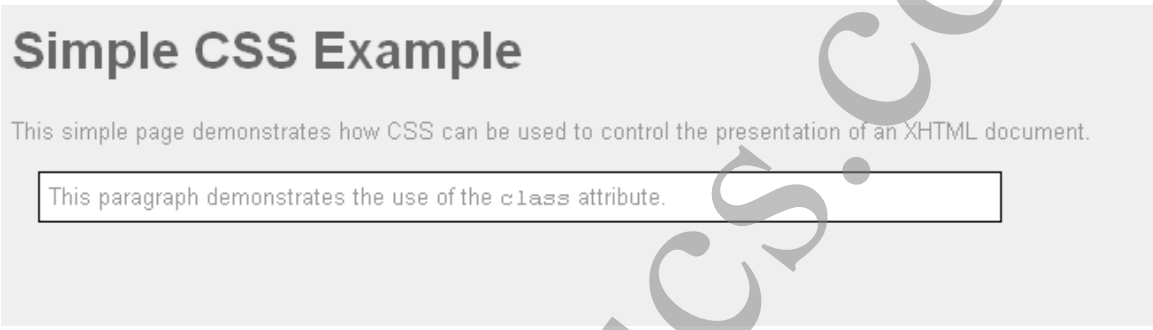
border:solid black 1px;

background-color:#ffffff;

padding:5px;

margin:15px;

width:40em;}



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

<head>

<title>CSS Example</title>

<link rel="stylesheet" type="text/css" href="ch03_eg02.css" /> </head> <body>

The first paragraph of text should be displayed in a sans-serif font.

The second paragraph of text should be displayed in a serif font.

The third paragraph of text should be displayed in a monospaced font.

The fourth paragraph of text should be displayed in a cursive font.

<p

class="five">The fifth paragraph of text should be displayed in a fantasy font.

</body>

</html>

CSS Document for ch03_eg02.html */

p.one {font-family:arial, verdana, sans-serif;}

p.two {font-family:times, "times new roman", serif;}

p.three {font-family:courier, "courier new", monospace;}

p.four {font-family: Zapf-Chancery, Santivo, cursive;}

p.five {font-family:Cottonwood, Studz, fantasy;}

The first paragraph of text should be displayed in a sans-serif font.

The second paragraph of text should be displayed in a serif font.

The third paragraph of text should be displayed in a monospaced font.

The fourth paragraph of text should be displayed in a cursive font.

The fifth paragraph of text should be displayed in a fantasy font.

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

<head>

<title>CSS Example</title>

<link rel="stylesheet" type="text/css" href="ch03_eg08.css" /> </head>

<body>

<h1>Lengths</h1>

The length used here is 12 px The length used here is 12

pt

The length used here is 2 pc The length used here is 0.5in

The length used here is 1cm The length used here is 12mm

The length used here is 1.5em The length used here is 1.5ex

</body> </html>

***** CSS Document for ch03_eg08.html */

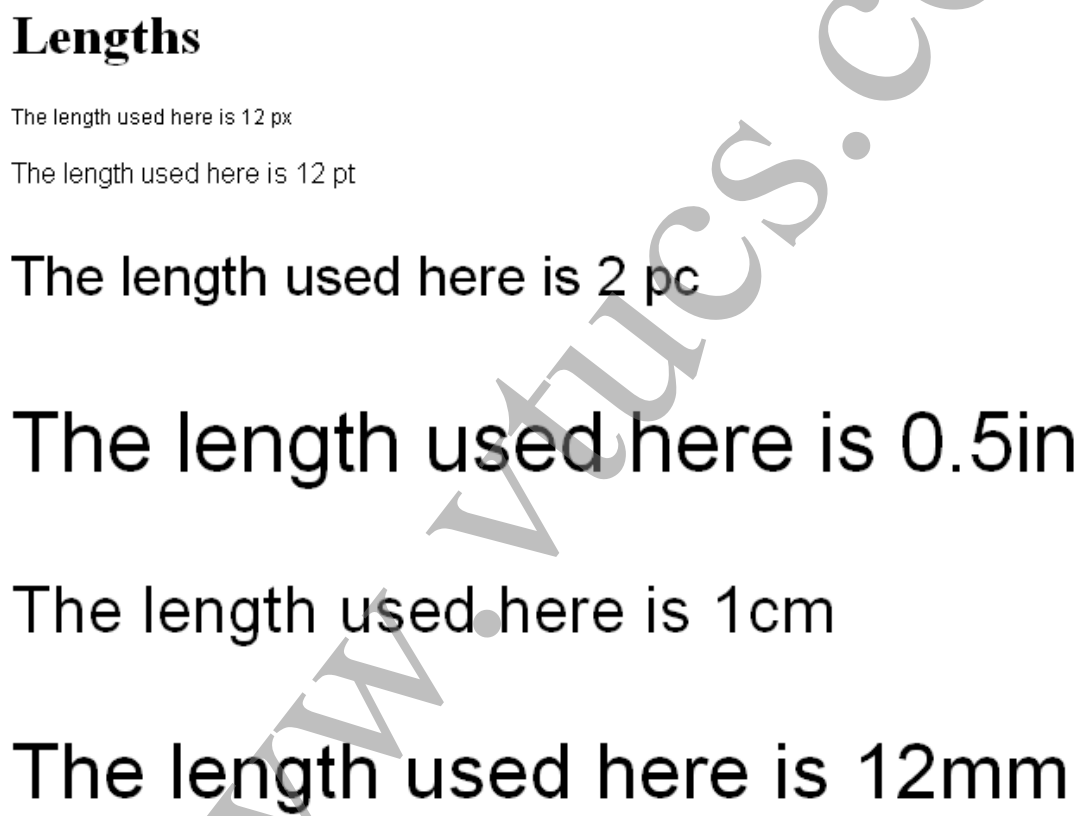
p {font-family:arial; font-size:12pt;}

/* lengths */

p.px {font-size:12px;}

p.pt {font-size:12pt;}

```
p.pc {font-size:2pc;}
p.in {font-size:0.5in;}
p.cm {font-size:1cm;}
p.mm {font-size:12mm;}
p.em {font-size:1.5em;}
p.ex {font-size:1.5ex;}
```



3.7 List properties

It is used to specify style of bullets or sequencing values in list items. The list-style-type of Unordered lists can be set to disc,circle,square or none. 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">
```

 Cessna Skyhawk

 Beechcraft Bonanza

 Piper Cherokee

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

<h3> Some Common Single-Engine Aircraft </h3>

<li style = "list-style-type: disc">

Cessna Skyhawk

<li style = "list-style-type: square">

Beechcraft Bonanza

<li style = "list-style-type: circle">

Piper Cherokee

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

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

<html>

</head><body>

<h3> Name of subjects offered</h3>

<ul style = "list-style-type: square">

 web programming

 Data structures

 Compilers design

<h3> Name of subjects offered</h3>

<li style = "list-style-type: disc">

web programming

<li style = "list-style-type: square">

Data structures

<li style = "list-style-type: circle">

Compilers design

</body>

</html>

Name of subjects offered

- web programming
- Data structures
- Compilers design

Name of subjects offered

- web programming
- Data structures
- Compilers design

When ordered lists are nested, it is best to use different kinds of sequence values for the different levels of nesting. The list-style-type can be used to change the sequence values. Below table lists the different possibilities defined by CSS1. Property value Sequence type first four values

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

CSS2 has more, like lower-greek and hebrew

```
<?xml version = "1.0"?>
<!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> Sequence types </title>
<style type = "text/css">
ol {list-style-type: upper-roman;}
ol ol {list-style-type: upper-alpha;}
ol ol ol {list-style-type: decimal;}
```

```
</style>
</head>body>
<h3> Aircraft Types </h3>
<ol>
<li> General Aviation (piston-driven engines)
<ol>
li> Single-Engine Aircraft
<ol>
<li> Tail wheel </li>
<li> Tricycle </li>
</ol>
</li>
<li> Dual-Engine Aircraft
<ol>
<li> Wing-mounted engines </li>
<li> Push-pull fuselage-mounted engines </li>
</ol>
</li>
</ol>
</li>
```

Aircraft Types

I. General Aviation (piston-driven engines)
   A. Single-Engine Aircraft
      1. Tail wheel
      2. Tricycle
   B. Dual-Engine Aircraft
      1. Wing-mounted engines
      2. Push-pull fuselage-mounted engines
II. Commercial Aviation (jet engines)
   A. Dual-Engine
      1. Wing-mounted engines
      2. Fuselage-mounted engines
   B. Tri-Engine
      1. Third engine in vertical stabilizer
      2. Third engine in fuselage

3.8 Colors

Colors are 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 larger set, the Web Palette 216 colors.Use hex color values of 00, 33, 66, 99, CC, and FF

2. Any one of 16 million different colors due to 24 bit color rep

3. 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

Color properties

The color property specifies the foreground color of XHTML elements. For example, consider the following small table

```
<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.

3.9 Alignment of Text

The text-indent property allows indentation. Takes either a length or a % value. The textalign 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.

```
<html xmlns = "http://www.w3.org/1999/xhtml">
<head> <title> The float property </title>
<style type = "text/css">
img {float: right}
```

The Box Model

```
</style>
</head>
```

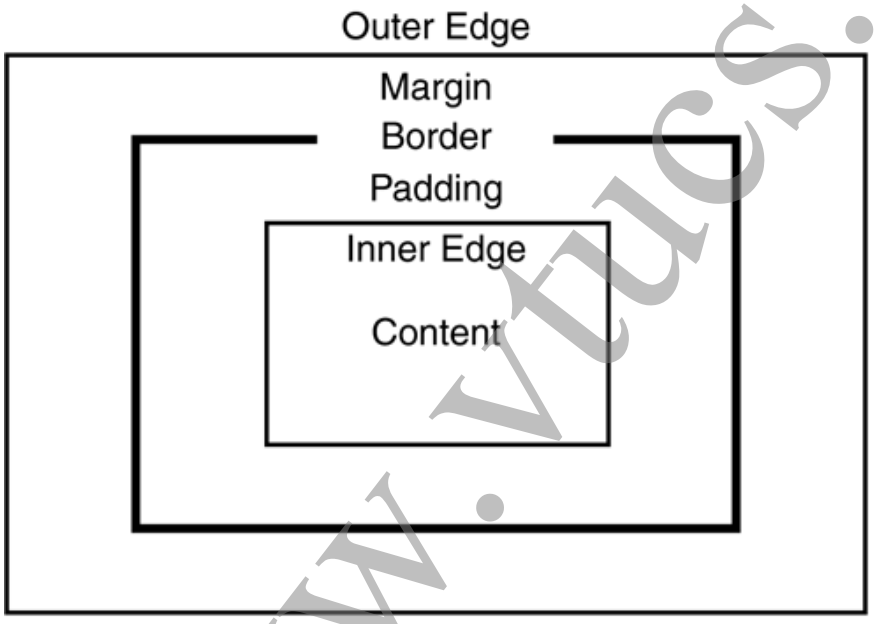
<body>

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. …. ……..

 </body>

</html>

3.10 The Box Model



Borders

Every element has a border-style property. It Controls whether the element has a border and if so, the style of the border. The styles of one of the four sides of an element can be set with 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) bordercolor – any color. Border color can be specified for any of the four borders (e.g., border-topcolor)

```
<html xmlns = "http://www.w3.org/1999/xhtml">
<head> <title> Table borders </title>
<style type = "text/css">
table {border-top-width: medium;
border-bottom-width: thick;
border-top-color: red;
border-bottom-color: green;
border-top-style: dotted;
border-bottom-style: dashed;
}
p {border-style: dashed; border-width: thin;
border-color: green
} </style> </head>
<body> <table border = "5">
<caption> Diet chart </caption>
<tr>
<th> </th>
<th> Fruits </th>
<th> vegetables </th>
<th> Carbohydrates </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> <tr>
<th> Dinner </th>
```

<td> 0 </td>

<td> 0 </td>

<td> 1 </td>

</tr>

</table>

If you strictly follow the chart you can easily lose weight.

 </body>

</html>

Diet chart

	Fruits	vegetables	Carbohydrates
Breakfast	0	1	0
Lunch	1	0	0
Dinner	0	0	1

If you strictly follow the chart you can easily lose weight.

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" />

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

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

<head><title> Margins and Padding

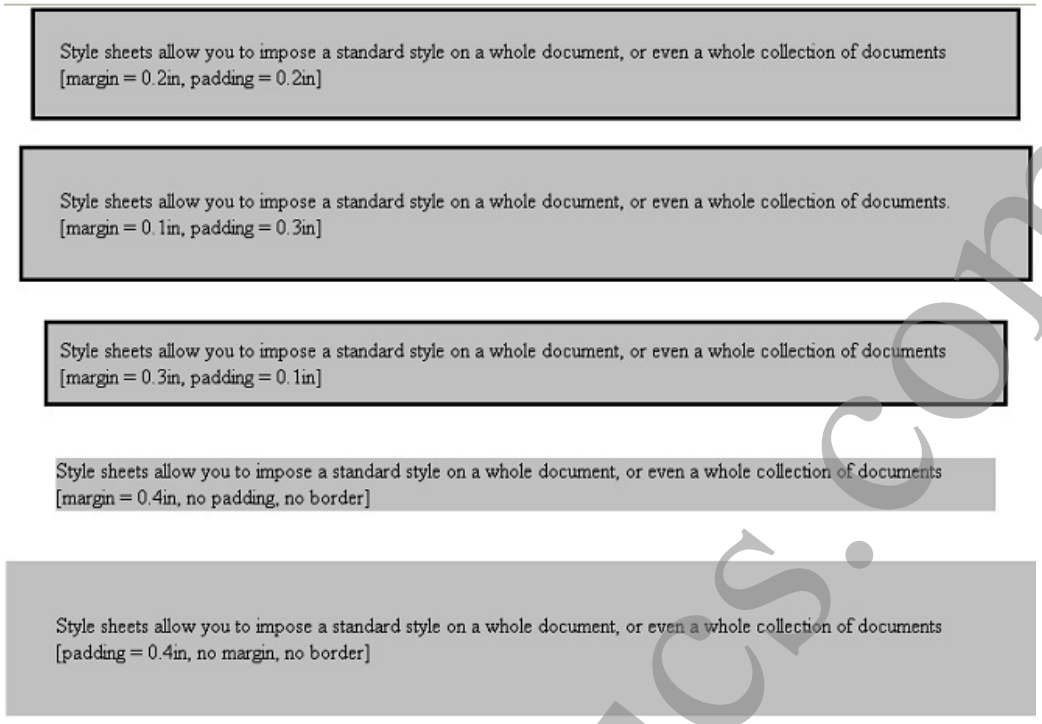
</title><style type = "text/css">

p.one {margin: 0.2in;

padding: 0.2in;

```
background-color: #C0C0C0;
border-style: solid;
}
p.two {margin: 0.1in;
padding: 0.3in;
background-color: #C0C0C0;
border-style: solid;
}
p.three {margin: 0.3in; padding: 0.1in;
background-color: #C0C0C0;
border-style: solid; }
p.four {margin:0.4in;
background-color: #C0C0C0;}
p.five {padding: 0.4in;
background-color: #C0C0C0;
}
</style> </head>
</style> </head> <body>
<p> Here is the first line. </p>
<p class = "one">
Style sheets allow you to impose a standard style on a whole document, or even a whole collection of documents <br /> [margin = 0.2in, padding = 0.2in]
</p>
<p class = "two">
Style sheets allow you to impose a standard style on a whole document, or even a whole collection of documents. <br /> [margin = 0.1in,
padding = 0.3in]
</p>
<p class = "three">
Style sheets allow you to impose a standard style on a whole document, or even a whole collection of documents <br /> [margin = 0.3in,
```

padding = 0.1in]



3.11 Background Images

The background-image property is used to place an image in the background of an element Repetition can be controlled. Background image can be replicated to fill the area of the element. This is known as tiling 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.

```
<html xmlns = "http://www.w3.org/1999/xhtml">
<head> <title> Background images </title>
<style type = "text/css">
body {background-image: url(c172.gif);}
p {margin-left: 30px; margin-right: 30px;
margin-top: 50px; font-size: 14pt;}
</style>
</head>
```

<body>

The Cessna 172 is the most common general aviation airplane in the world. It is an allmetal, single-engine piston, ……..at sea level is 720 feet per minute.

 </body>

</html>

3.12 Conflict Resolution

When two or more rules apply to the same tag there are resolutions for deciding which rule applies. 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

h3{color:red;} body h3 {color: green;}

A tag may inherit a property and also be used as a selector. Style sheets can have different sources:

The browser itself may set some style

eg: In FX2 min font size can be set in Tools-Options-

Advanced window

The author of a document may specify styles. The user, through browser settings, may specify styles. Individual properties can be specified as important or normal.

Eg: p.special{font-style: italic !important; font-size :14}

This property is known as weight of a specification. Conflict resolution is a multistage sorting process. The first step in the process is to gather the style specifications from the three possible levels of style sheets. These specifications are sorted into order by the relative precedence of the style sheet levels. This is done according to the following rules, in which the first has the highest precedence. 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

Conflict resolution by Specificity (high to low)

1. id selectors
2. Class and pseudo-class selectors
3. Contextual selectors
4. General selectors

Position

Essentially, later has precedence over earlier. Most recently seen specification is the one which gets more precedence. Sorting process to resolve the style specification is known as cascade.

UNIT - 4

JAVASCRIPT

4.1 Overview of Javascript

4.2 Object orientation and Javascript

4.3 General syntactic characteristics

4.4 Primitives, operations, and expressions

4.5 Screen output and keyboard input

4.6 Control statements

4.7 Object creation and modification

4.8 Arrays

4.9 Functions

4.10 Constructor

4.11 Pattern matching using regular expressions

4.12 Errors in scripts

4.13 Examples

UNIT - 4

JAVASCRIPT

4.1 Overview of Javascript

JavaScript is a sequence of statements to be executed by the browser. It is most popular scripting language on the internet, and works in all major browsers, such as IE, FireFox, chrome, opera safari. Prequisite –HTML/XHTML

Origins

It is originally known as LiveScript, developed by Netscape. It become a joint venture of Netscape and Sun in 1995, and was renamed as JavaScript. It was standardized by the European computer Manufacturers Association as ECMA-262. ISO-16262. Current standard specifications can be found at

http://www.ecma-international.org/publications/standardsEcma-262.htm

Collections of JavaScript code scripts and not programs.

What is JavaScript?

1. JavaScript was designed to add interactivity to HTML pages.
2. JavaScript is a scripting language.
3. A scripting language is a lightweight programming language.
4. It is usually embedded directly into HTML pages.
5. JavaScript is an interpreted language (Scripts are executed without preliminary compilations)

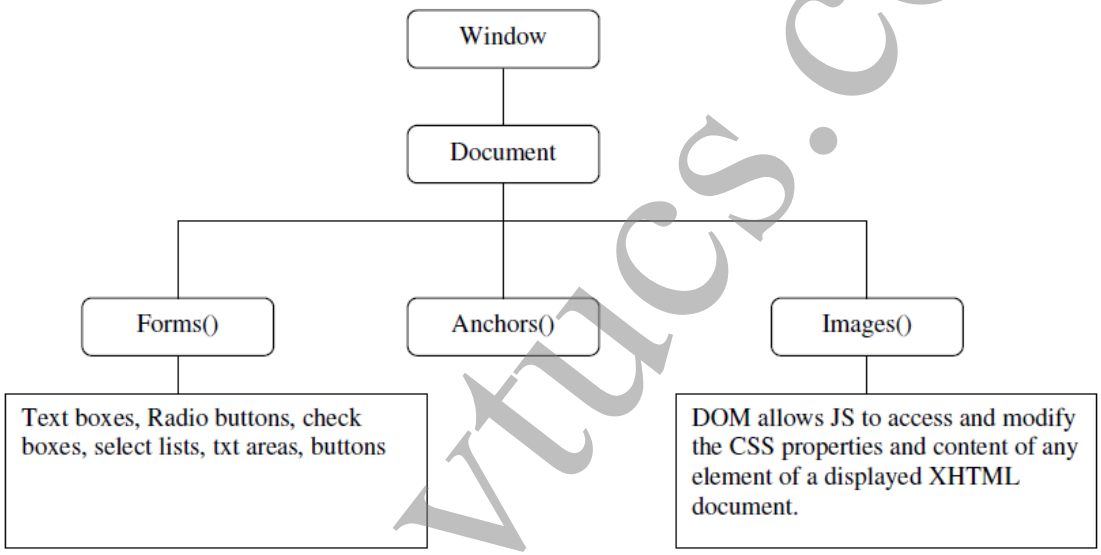
JavaScript can be divided into three parts.

1. The Core: It is a heart of the language, including its operators, expressions, statements and subprograms.

2. Client Side: It is a collection of objects that support control of a browser and interactions with users. Eg. With JavaScript an XHTML document can be made to be responsible to user inputs. Such as mouse clicks and keyboard use.

3. Server side: It is a collection of objects that make the language useful on a Web server. Eg. To support communication with a DBMS. Client side JavaScript is an XHTML embedded scripting language. We refer to every collection of JavaScript code as a script. An XHTML document can include any number of embedded scripts. The HTML Document Object Model(DOM) is the browsers view of an HTML page as an object hierarchy, starting with the browser window itself and moving deeper into the page, including of the elements on the page and their attribute.

Fig: The HTML DOM



The top level object is window. The document object is a child of window and all the objects that appear on the page are descendants of the document object. These objects can have children of their own. Eg. Form objects generally have several child objects , including textboxes, radio buttons and select menus.

JavaScript and java

Document Forms() Anchors() Images()

Window Text boxes, Radio buttons, check boxes, select lists, txt areas, buttons

DOM allows JS to access and modify the CSS properties and content of any element of a displayed XHTML document.

JavaScript and java is only related through syntax.

JavaScript support for OOP is different from that of Java.

JavaScript is dynamically typed.

Java is strongly typed language. Types are all known at compile time and operand types are checked for compatibility. But variables in JavaScript need not be declared and are dynamically typed, making compile time type checking impossible.

Objects in Java are static -> their collection of data number and methods is fixed at compile time.

JavaScript objects are dynamic: The number of data members and methods of an object can change during execution.

Uses of JavaScript

Goal of JavaScript is to provide programming capability at both server and the client ends of a Web connection. Client-side JavaScript is embedded in XHTML documents and is interpreted by the browser. This transfer of load from the often overloaded server to the normally under loaded client can obviously benefit all other clients. It cannot replace server side computations like file operations, database access, and networking.

JavaScript can be used as an alternative to Java applets. Java applets are downloaded separately from the XHTML documents that call them but JavaScript are integral part of XHTML document, so no secondary downloading is necessary. Java applets far better for graphics files scripts.

Interactions with users through form elements, such as buttons and menus, can be conveniently described in JavaScript. Because events such as button clicks and mouse movements are easily detected with JavaScript they can be used to trigger computations and provide feedback to the users.

Eg. When user moves the mouse cursor from a textbox, JavaScript can detect that movement and check the appropriateness of the text box's value. Even without forms, user interactions are both possible and simple to program. These interactions which take place in dialog windows include getting input from the user and allowing the user to make choices through buttons. It is also easy to generate new content in the browser display dynamically.

Event driven computation

Event driven computation means that the actions often are executed in response to actions often are executed in response to actions of the users of doc, actions like mouse clicks and form submissions. This type of computation supports user interactions through XHTML form elements on the client display. One of the common uses of JS is client end input data validation values entered by users will be checked before sending them to server for further processing. This becomes more efficient to perform input data checks and carry on this user dialog entirely on the client. This saves both server time and internet time.

Browsers and XHTML/JS documents.

It is an XHTML document does not include embedded scripts, the browser reads the lines of the document and renders its window according to the tags, attributes and content it finds when a JavaScript script is encountered in the doc, the browser uses its JS interpreter to execute the script. When the end of script reached, the browser goes back to reading the XHTML document and displaying its content.

JS scripts can appear in either part of an XHTML document, the head or the body, depending on the purpose of the script. Scripts that produce content only when requested or that react to user interactions are placed in the head of the document. -> Function definition and code associated with form elements such as buttons. Scripts that are to be interpreted just once, when the interpreter finds them are placed in the document body. Accordingly, the interpreter notes the existence of scripts that appear in the head of a document, but it does not interpret them while processing the head. Scripts that are found in the body of a document are interpreted as they are found.

4.2 Object orientation and Javascript

JavaScript is object based language. It doesn't have classes. Its objects serve both as objects and as models of objects. JavaScript does not support class based inheritance as is supported in OO language. CTT-Java. But it supports prototype based inheritance i.e a

technique that can be used to simulate some of the aspects of inheritance. JavaScript does not support polymorphism. A polymorphic variable can reference related objects of different classes within the same class hierarchy. A method call through such a polymorphic variable can be dynamically bound to the method in the objects class.

JavaScript Objects

JavaScript objects are collection of prospectus, which corresponds to the members of classes in Java & C++. Each property is either a data property or a function or method property.

1. Data Properties

a. Primitive Values (Non object Types)

b. Reference to other objects

2. Method Properties –methods.

Primitives are non object types and are used as they can be implemented directly in hardware resulting in faster operations on their values. These are accessed directly-like scalar types in java & C++ called value types. All objects in a JavaScript programs are directly accessed through variables. Such a variable is like a reference in java. The properties of an object are referenced by attaching the name of the property to the variable that references the object. Eg. If myCar variable referencing an object that has the property engine, the engine property can be referenced with myCar.engine.

The root object in JavaScript is object. It is ancestor through prototype inheritance, of all objects. Object is most generic of all objects, having some methods but no data properties. All other objects are specializations of object, and all inherit its methods.

JavaScript object appears both internally and externally as a list of property/value pairs. Properties are names values are data values of functions. All functions are objects and are referenced through variables. The collection of properties of JavaScript is dynamic – Properties can be added or deleted at any time.

4.3 General syntactic Characteristics

1. JavaScript are embedded either directly or indirectly in XHTML documents.
2. Scripts can appear directly as the content of a <script> tag.
3. The type attribute of <script> must be set to “text/JavaScript”.

4. The JavaScript can be indirectly embedded in an XHTML document using the src attribute of a <script> tag, whose value is name of a file that contains the script.

Eg. <script type=”text/JavaScript” src=”tst_number.js”>

</script>

Closing tag is required even if script element has src attribute included.

The indirect method of embedding JavaScript in XHTML has advantages of

1) Hiding the script from the browser user.

2) It also avoids the problem of hiding scripts from older browsers.

3) It is good to separate the computation provided by JavaScript from the layout and presentation provided by XHTML and CSS respectively. But it is sometimes not convenient and cumbersome to place all JavaScript code in separate file JavaScript identifiers or names are similar to programming languages.

1. must begin with (-), or a letter. Subsequent characters may be letters, underscores or digits.

2. No length limitations for identifiers.

3. Case sensitive

4. No uppercase letters.

Reserved words are break delete function return typeof case do if switch var catch else in this void continue finally instanceof throw while default for new try with

JavaScript has large collection of predefined words

alert

open

java

self

Comments in JavaScript

// - Single line

/* */ -Multiple line

Two issues regarding embedding JavaScript in XHTML documents.

1) There are some browsers still in use that recognize the <script> tag but do not have JS interpreters. These browsers will ignore the contents of the script element and cause no problems.

2) There are still a few browsers in use that are so old they do not recognize <script> tag. These browsers will display the contents of the script elements as if it were just text. Therefore it has been customary to enclose the contents of all script elements in XHTML comments to avoid this problem. XHTML validator also has a problem with embedded JS. When embedded JS happens to include recognizable tags.

For eg
 in output of JS-they often cause validation errors.

Therefore we have to enclose embedded JS in XHTML comments. XHTML comment introduction (<! - -) works as a hiding prelude to JS code. Syntax for closing a comment that encloses JS code is different. It is usual XHTML comment closer but it must be on its own line and preceeded by two slashes.

Eg. <!—

-- JS ---

//-->

Many more problem are associated with putting embedded JavaScript in comments in XHTML document.

Solution : Put JavaScript scripts of significant style in separate files.

Use of ; in JS is unusual

When EOL coincides with end of statement, the interpreter effectively insects a semicolon there, but this leads to problems.

Eg. return x;

Interpreter puts; after return making x an illegal orphan.

Therefore put JS statements on its own line when possible and terminate each statement with a semicolon. If stmt does not fit in one line, break the stmt at a place that will ensure that the first line does not have the form of a complete statement.

<?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>

<! - -hello.html

8

A trivial hello world example of XHTML/JavaScript

- ->

```
<html xmlns = “http://www.w3.org/1999/xhtml”.>
<head>
<title> Hello World</title>
</head>
<body>
<script type = “text/javascript”>
<!- -
Document.write(“Hello, fellow Web programmers!”);
//- ->
</script>
</body>
</html>
```

4.4 Primitives, operations, and expressions

The primitive data types, operations and expressions of JavaScript.

Primitive Types:

Pure primitive types : Number, String, Boolean, Undefined and null. JavaScript includes predefined objects that are closely related to the number, string and Boolean types named number, string and Boolean. These are wrapper objects. Each contains a property that stores a value of the corresponding primitive type. The purpose of the wrapper object is to provide properties and methods that are convenient for use with values of the primitive types.

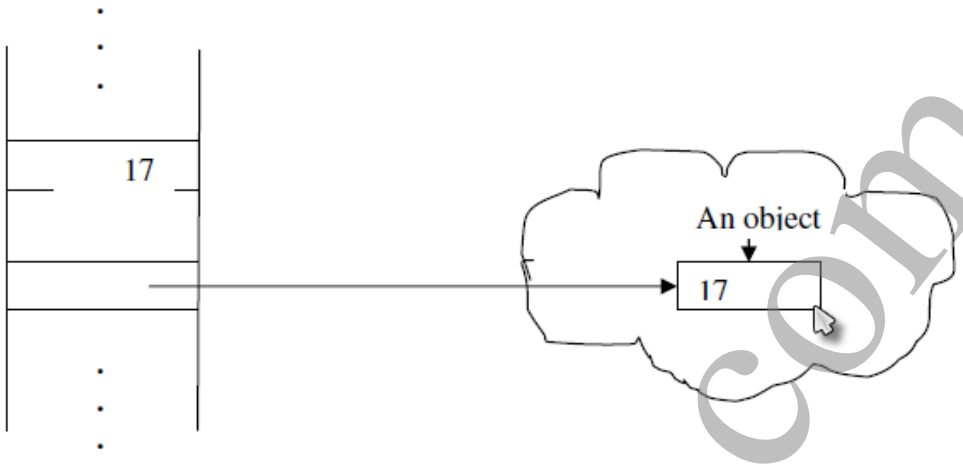
In case of numbers : Properties are more useful.

In case of string : Methods are more useful.

Because JavaScript coerces values between the number type and number objects and between the string type and objects, the methods of number and string can be used on variables of the corresponding primitive types.

Difference between primitives and objects :

Fig:



Prim is a primitive variable with value 17 and obj is a number object whose property value is 17. Fig shows how they are stored.

Numeric and String literals:

All numeric literals are values of type number. The numeric values of JavaScript are represented internally in double precision floating point form, Numeric values in JavaScript are called numbers because of single numeric data type. Literal numbers in a script can have forms of either integers or floating point values. Integer literals are strings of digits.

Floating point literals can have decimal points or exponents or both.

Legal numeric literals: 72, 7.2, .72, 72, 7E2, 7e2, .7e2, 7.e2, 7.2E-2.

Integers in Hexadecimal form 0x or 0X.String Literal: Sequence of 0 or more characters delimited by either single quotes or double quotes. They can include characters specified with escape sequences, such as \n and \t. If you want an actual single quote character in a string literal that is delimited by single quotes, embedded single quote must be preceded by a backslash.

'You\' re the most freckly person I\'ve ever met'

"D:\\bookfiles" ->Jo embed\

' ' or " " -> Null string

4.5 Screen output and keyboard input

A JavaScript is interpreted when the browser finds the script in the body of the XHTML document. Thus the normal screen for the JavaScript is the same as the screen in which the content of the host XHTML document is displayed. JS models the XHTML document with the document object. The window in which the browser displays an XHTML document is modeled with the window object. It includes two properties document and window.

Document -> Refers to document object.

Window -> self referential and refers to window object.

Methods -> write

Write is used to create script o/p, which is dynamically created XHTML document content.

Eg. document.write("The result is : ", result,"
");

Because write is used to create XHTML code, the only useful punctuations in its parameter is in the form of XHTL tags. Therefore the parameter to write often includes
 writeln methods implementing adds "\n" to its parameter. As browsers neglects line breaks when displaying XHTML, it has no effect on the output. Window object is JS model for the browser window. It includes three methods that create dialog boxes for three specific kinds of user interactions. The default object for JS is window object currently being displayed, so calls to these methods need not include an object reference.

Alert method opens a dialog window and displays its parameter in that window. It also displays an OK button. The parameter string to alert is not XHTML code, it is plain text. There fore the string parameter to alert may include \n but never should include
.

Confirm method opens a dialog window in which it displays its string parameter, along with two buttons OK and Cancel. Confirm returns a Boolean value that indicates the users button input

True -> for OK

False-> for cancel.

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

After the user responds to one of the button in the confirm dialog window script can test the variable, question and react accordingly. Prompt method creates a dialog window that contains a text box which is used to collect a string of input from the user, which prompt returns as its value. The window also includes two buttons, OK and Cancel, prompt takes two parameters the string that prompts the user for input and a default string in case the user does not type a string before pressing one of the two buttons. In many cases an empty string is used for the default input.

Eg. name name=prompt("What is your name"."");

Alert, prompt and confirm cause the browser to wait for a user response.

Alert – OK

Prompt-OK, Cancel

Confirm-OK, Cancel.

Eg.

```
<html>
<head>
</title> roots.html</title>
</head>
<body>
<script type ="text/javascript" src="roots.js">
</script>
</body>
</html>
```

```
//roots.js
// Compute the real roots of a given quadratic equation. If the roots are imaginary,
//this script displays NaN, because that is what results from taking the square root of
//a negative number.
//Get the coefficients of the equation of the equation from the user
var a = prompt("What is the value of 'a'?\n", "");
var b = prompt("What is the value of 'b'?\n", "");
var c = prompt("What is the value of 'c'?\n", "");
```

```
// Compute Square root
var root_part = Math.sqrt(b * b – 4.0 * a * c);
var denom = 2.0 * a;
//Compute and Display
Var root1 = (-b + root_part) / denom;
Var root2 = (-b - root_part) / denom;
document.write(“The first root is :”,root1, “<br/>”);
document.write(“The second root is :”,root2, “<br/>”);
```

4.7 Control Statements

Control expression control the order of execution of statements. Compound statements is JavaScript are syntactic contains for sequences of statements whose execution they control. Compound statement sequence of statements deleted by braces.

Control construct is a control statement whose execution it controls. Compound statements are not allowed to create local variables.

Control Expressions.

Control statements flow.

Eg. primitive values, relational expression and compound expressions. Result of evaluating a control expression is Boolean value true or false.

For strings.

true - string

false-null string

For number

True- any number

False-0

Operation	Operator
Is equal to	= =
Is not equal to	!=
Is less than	<
Is greater than	>
Is less than or equal to	<=
Is greater than or equal to	>=
Is strictly equal to	= = =
Is strictly not equal to	! = =

If two operands are not of the same type and operator is neither = = = or ! = =, JS will convert to a single type. Eg. If one is string and other is number, JS will convert string to a number. If one operand is Boolean and other is not, then Boolean is converted to a number.(1 for true, 0 for false)

4.7 Object creation and modification

Create objects with the new keyword followed by a constructor. For example, the following program creates a TwoDPoint object and prints its fields:

```
class OriginPrinter {

 public static void main(String[] args) {

   TwoDPoint origin; // only declares, does not allocate

   // The constructor allocates and usually initializes the object
   origin = new TwoDPoint();

   // set the fields
   origin.x = 0.0;
   origin.y = 0.0;
```

```
    // print the two-d point
    System.out.println(
     "The origin is at " + origin.x + ", " + origin.y);

  }  // end main

} // end OriginPrinter
```

The '.' is the member access separator.

A constructor invocation with new is required to allocate an object. There is no C++ like static allocation.

To compile this class, put it in a file called OriginPrinter.java in the same directory as TwoDPoint.java and type:

```
$ javac OriginPrinter.java
```

Multiple Objects

In general there will be more than one object in any given class. Reference variables are used to distinguish between different objects of the same class.

For example, the following program creates two two-d point objects and prints their fields:

```
class TwoPointPrinter {
  public static void main(String[] args) {

    TwoDPoint origin; // only declares, does not allocate
    TwoDPoint one; // only declares, does not allocate

    // The constructor allocates and usually initializes the object
```

```
    origin = new TwoDPoint();
    one = new TwoDPoint();

    // set the fields
    origin.x = 0.0;
    origin.y = 0.0;
    one.x = 1.0;
    one.y = 0.0;

    // print the 2D points
    System.out.println(
     "The origin is at " + origin.x + ", " + origin.y);
    System.out.println("One is at " + one.x + ", " + one.y);

  }  // end main

} // end TwoPointPrinter
```

one and origin are two different reference variables pointing to two different point objects. It's not enough to identify a variable as a member of a class like x or y in the example above. You have to specify which object in the class you're referring to.

It is possible for two different reference variables to point to the same object.

When an object is no longer pointed to by any reference variable (including references stored deep inside the runtime or class library) it will be marked for garbage collection.

For example, the following program declares two TwoDPoint reference variables, creates one two-d point object, and assigns that object to both variables. The two variables are equal.

```
class EqualPointPrinter {
  public static void main(String[] args) {
```

```
    TwoDPoint origin1; // only declares, does not allocate
    TwoDPoint origin2; // only declares, does not allocate

    // The constructor allocates and usually initializes the object
    origin1 = new TwoDPoint();
    origin2 = origin1;

    // set the fields
    origin1.x = 0.0;
    origin1.y = 0.0;

    // print
    System.out.println(
     "origin1 is at " + origin1.x + ", " + origin1.y);
    System.out.println(
     "origin2 is at " + origin2.x + ", " + origin2.y);

  }  // end main

} // end EqualPointPrinter
```

origin1 and origin2 are two different reference variables referring to the same point object.

4.8 Arrays

An array is a collection of variables of the same type. The args[] array of a main() method is an array of Strings.

Consider a class which counts the occurrences of the digits 0-9. For example you might wish to test the randomness of a random number generator. If a random number generator is truly random, all digits should occur with equal frequency over a sufficiently long period of time.

You will do this by creating an array of ten ints called ndigit. The zeroth component of ndigit will track the number of zeros; the first component will track the numbers of ones and so forth. The RandomTest program below tests Java's random number generator to see if it produces apparently random numbers.

```
import java.util.Random;

class RandomTest {

 public static void main (String args[]) {

  int[] ndigits = new int[10];

  Random myRandom = new Random();

  // Initialize the array
  for (int i = 0; i < 10; i++) {
   ndigits[i] = 0;
  }
  // Test the random number generator a whole lot
  for (long i=0; i < 100000; i++) {
   // generate a new random number between 0 and 9
   double x = myRandom.nextDouble() * 10.0;
   int n = (int) x;
   //count the digits in the random number
   ndigits[n]++;
  }
```

```
    // Print the results
    for (int i = 0; i < 10; i++) {
      System.out.println(i+": " + ndigits[i]);
    }
  }

}
```

Below is one possible output from this program. If you run it your results should be slightly different. After all this is supposed to be random. These results are pretty much what you would expect from a reasonably random generator. If you have a fast CPU and some time to spare, try bringing the number of tests up to a billion or so, and see if the counts for the different digits get any closer to each other.

```
$ javac RandomTest.java
$ java RandomTest
0: 10171
1: 9724
2: 9966
3: 10065
4: 9989
5: 10132
6: 10001
7: 10158
8: 9887
9: 9907
```

There are three for loops in this program, one to initialize the array, one to perform the desired calculation, and a final one to print out the results. This is quite common in code that uses arrays.

4.9 Functions

Methods say what an object does.

```
class TwoDPoint {
   double x;
   double y;

   void print() {
     System.out.println(this.x + "," + this.y);
   }

}
```

Notice that you use the Java keyword this to reference a field from inside the same class.

```
TwoDPoint origin = new TwoDPoint();
origin.x = 0.0;
origin.y = 0.0;
origin.print();
```

noun-verb instead of verb-noun; that is subject-verb instead of verb-direct object.

subject-verb-direct object(s) is also possible.

4.10 Constructor

Constructors create new instances of a class, that is objects. Constructors are special methods that have the same name as their class and no return type. For example,

```
class TwoDPoint {
   double x;
   double y;

   TwoDPoint(double xvalue, double yvalue) {
    this.x = xvalue;
    this.y = yvalue;
   }

   String getAsString() {
    return "(" + this.x + "," + this.y + ")";
   }

   void setX(double value) {
    this.x = value;
   }

   void setY(double value) {
    this.y = value;
   }

   double getX() {
    return this.x;
   }

   double getY() {
    return this.y;
   }

}
```

Constructors are used along with the new keyword to produce an object in the class (also called an instance of the class):

```
TwoDPoint origin = new TwoDPoint(0.0, 0.0);
System.out.println("The x coordinate is " + origin.getX());
```

4.11 Errors in scripts

Web browsers are such an hostile environment that it is almost guaranteed that we will constantly deal with runtime errors. Users provide invalid input in ways you didn't think of. New browser versions change their behavior. An AJAX call fails for a number of reasons.

Many times we can't prevent runtime errors from happening, but at least we can deal with them in a manner that makes the user experience less traumatic.

Completely unhandled errors

Look at this seemingly trivial code snippet.

```
function getInput() {
 var name = window.prompt('Type your name', '');
 alert('Your name has ' + name.length + ' letters.');
}
```

It may not be obvious, but this code has a bug waiting to break free. If the user clicks Cancel or presses Esc the prompt() function will return null, which will cause the next line to fail with a null reference error.

If you as a programmer don't take any step to deal with this error, it will simply be delivered directly to the end user, in the form of a utterly useless browser error message like the one below.

Depending on the user's browser or settings, the error message may be suppressed and only an inconspicuous icon shows up in the status bar. This can be worse than the error message, leaving the users thinking the application is unresponsive.

Globally handled errors

The window object has an event called onerror that is invoked whenever there's an unhandled error on the page.

```
window.onerror = function (message, url, lineNo) {
 alert(
  'Error: ' + message +
  '\n Url: ' + url +
  '\n Line Number: ' + lineNo);
 return true;
}
```

As you can see, the event will pass 3 arguments to the invoked function. The first one is the actual error message. The second one is the URL of the file containing the error (useful if the error is in an external .js file.) The last argument is the line number in that file where the error happened.

Returning true tells the browser that you have taken care of the problem. If you return false instead, the browser will proceed to treat the error as unhandled, showing the error message and the status bar icon.

Here's the message box that we will be showing to the user.

Structured Error Handling

The best way to deal with errors is to detect them the closest possible to where they happen. This will increase the chances that we know what to do with the error. To that effect JavaScript implements structured error handling, via the try...catch...finally block, also present in many other languages.

Syntax

```
try {
   statements;
} catch (error) {
   statements;
} finally {
   statements;
}
```

The idea is simple. If anything goes wrong in the statements that are inside the try block's statements then the statements in the catch block will be executed and the error will be passed in the error variable. The finally block is optional and, if present, is always executed last, regardless if there was an error caught or not.

Let's fix our example to catch that error.

```
function getInput(){
 try {
  var name = window.prompt('Type your name', '');
  alert('Your name has ' + name.length + ' letters.');
 } catch (error) {
  alert('The error was: ' + error.name +
  '\n The error message was: ' + error.message);
 } finally {
  //do cleanup
 }
}
```

The error object has two important properties: name and message. The message property contains the same error message that we have seen before. The name property contains the kind of error that happened and we can use that to decide if we know what to do with that error.

It's a good programming practice to only handle the error on the spot if you are certain of what it is and if you actually have a way to take care of it (other than just suppressing it altogether.) To better target our error handling code, we will change it to only handle errors named "TypeError", which is the error name that we have identified for this bug.

```
function getInput(){
 try {
  var name = window.prompt('Type your name', '');
  alert('Your name has ' + name.length + ' letters.');
 } catch (error) {
  if (error.name == 'TypeError') {
   alert('Please try again.');
  } else {
   throw error;
  }
 } finally {
  //do cleanup
 }
}
```

Now if a different error happens, which is admittedly unlikely in this simple example, that error will not be handled. The throw statement will forward the error as if we never had this try...catch...finally block. It is said that the error will bubble up.

Throwing custom errors

We can use the throw statement to throw our own types of errors. The only recommendation is that our error object also has a name and message properties to be consistent in error handling.

```
throw {
 name: 'InvalidColorError',
 message: 'The given color is not a valid color value.'
};
```

Debugging

One of the most important activities in software development is debugging. It can also be one of the most costly. That's why we need to do our best to reduce the amount of time spent in debugging.

One way to reduce this time is to create automated unit tests, which we will see in the lesson Production Grade JavaScript.

Another way is to use the best tools available and try to remove the pain associated with debugging. It used to be the case that debugging tools for JavaScript were archaic or close to non-existent. This situation has improved a lot and now we can confidently say we have feasible ways to debug JavaScript without resorting to horrendous tactics, such as sprinkling alert() calls across our code.

We won't waste your time discussing all the existing tools for debugging. Instead we will focus on the tool that singlehandedly re-wrote the JavaScript debugging history.

UNIT - 5

JAVASCRIPT AND HTML DOCUMENTS

5.1 The Javascript execution environment

5.2 The Document Object Model

5.3 Element access in Javascript

5.4 Events and event handling

5.5 Handling events from the Body elements

5.6 Button elements

5.7 Text box and Password elements

5.8 The DOM 2 event model

5.9 The navigator object

5.10 DOM tree traversal and modification.

UNIT - 5

JAVASCRIPT AND HTML DOCUMENTS

5.1 The Javascript execution environment

Features

The following features are common to all conforming ECMAScript implementations, unless explicitly specified otherwise.

Imperative and structured

JavaScript supports all the structured programming syntax in C (e.g., if statements, while loops, switch statements, etc.). One partial exception is scoping: C-style block-level scoping is not supported (instead, JavaScript has function-level scoping). JavaScript 1.7, however, supports block-level scoping with the let keyword. Like C, JavaScript makes a distinction between expressions and statements. One syntactic difference from C is automatic semicolon insertion, in which the semicolons that terminate statements can be omitted.[18]

Dynamic

dynamic typing

> As in most scripting languages, types are associated with values, not variables. For example, a variable x could be bound to a number, then later rebound to a string. JavaScript supports various ways to test the type of an object, including duck typing.[19]

object based

> JavaScript is almost entirely object-based. JavaScript objects are associative arrays, augmented with prototypes (see below). Object property names are string keys: obj.x = 10 and obj["x"] = 10 are equivalent, the dot notation being syntactic

sugar. Properties and their values can be added, changed, or deleted at run-time. Most properties of an object (and those on its prototype inheritance chain) can be enumerated using a for...in loop. JavaScript has a small number of built-in objects such as Function and Date.

run-time evaluation

JavaScript includes an eval function that can execute statements provided as strings at run-time.

Functional

first-class functions

Functions are first-class; they are objects themselves. As such, they have properties and can be passed around and interacted with like any other object.

inner functions and closures

Inner functions (functions defined within other functions) are created each time the outer function is invoked, and variables of the outer functions for that invocation continue to exist as long as the inner functions still exist, even after that invocation is finished (e.g. if the inner function was returned, it still has access to the outer function's variables) — this is the mechanism behind closures within JavaScript.

Prototype-based

prototypes

JavaScript uses prototypes instead of classes for inheritance. It is possible to simulate many class-based features with prototypes in JavaScript.

functions as object constructors

Functions double as object constructors along with their typical role. Prefixing a function call with new creates a new object and calls that function with its local this keyword bound to that object for that invocation. The constructor's prototype property determines the object used for the new object's internal prototype. JavaScript's built-in constructors, such as Array, also have prototypes that can be modified.

functions as methods

Unlike many object-oriented languages, there is no distinction between a function definition and a method definition. Rather, the distinction occurs during function calling; a function can be called as a method. When a function is called as a method of an object, the function's local this keyword is bound to that object for that invocation.

Miscellaneous

run-time environment

JavaScript typically relies on a run-time environment (e.g. in a web browser) to provide objects and methods by which scripts can interact with "the outside world". In fact, it relies on the environment to provide the ability to include/import scripts (e.g. HTML <script> elements). (This is not a language feature per se, but it is common in most JavaScript implementations.)

variadic functions

An indefinite number of parameters can be passed to a function. The function can access them through formal parameters and also through the local arguments object.

array and object literals

Like many scripting languages, arrays and objects (associative arrays in other languages) can each be created with a succinct shortcut syntax. In fact, these literals form the basis of the JSON data format.

regular expressions

JavaScript also supports regular expressions in a manner similar to Perl, which provide a concise and powerful syntax for text manipulation that is more sophisticated than the built-in string functions.

5.2 The Document Object Model

The **Document Object Model** (**DOM**) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents. Aspects of the DOM (such as its "Elements") may be addressed and manipulated within the syntax of the programming language in use. The public interface of a DOM is specified in its Application Programming Interface (API).

Applications

DOM is likely to be best suited for applications where the document must be accessed repeatedly or out of sequence order. If the application is strictly sequential and one-pass, the SAX model is likely to be faster and use less memory. In addition, non-extractive XML parsing models such as VTD-XML, provide a new memory-efficient option.

Web browsers

A web browser is not obliged to use DOM in order to render an HTML document. However, the DOM is required by JavaScript scripts that wish to inspect or modify a web page dynamically. In other words, the Document Object Model is the way JavaScript sees its containing HTML page and browser state.

Implementations

Because DOM supports navigation in any direction (e.g., parent and previous sibling) and allows for arbitrary modifications, an implementation must at least buffer the document that has been read so far (or some parsed form of it).

Layout engines

Web browsers rely on layout engines to parse HTML into a DOM. Some layout engines such as Trident/MSHTML and Presto are associated primarily or exclusively with a particular browser such as Internet Explorer and Opera respectively. Others, such as WebKit and Gecko, are shared by a number of browsers, such as Safari, Google Chrome,

Firefox or Flock. The different layout engines implement the DOM standards to varying degrees of compliance.

See also: Comparison of layout engines (Document Object Model)

Libraries

DOM implementations:

- libxml2
- MSXML
- Xerces is a C++ DOM implementation with Java and Perl bindings

APIs that expose DOM implementations:

- JAXP (Java API for XML Processing) is an API for accessing DOM providers

Alternative non-DOM tree-based XML libraries:

- VTD-XML is a Java library that offers alternative tree-based view to XML documents

5.3 Element access in Javascript

Javascript provides the ability for getting the value of an element on a webpage as well as dynamically changing the content within an element.

Getting the value of an element

To get the value of an element, the **getElementById** method of the **document** object is used. For this method to get the value of an element, that element has to have an id given to it through the **id** attribute.

Example:

<script type="text/javascript"> function getText(){ //access the element with the id 'textOne' and get its value //assign this value to the variable theText var theText = document.getElementById('textOne').value; alert("The text in the textbox is " + theText); } </script> <input type="text" id="textOne" /> <input type="button" value="Get text" onclick="getText()" />

Changing the content within an element

To change the content within an element, use the innerHTML property. Using this property, you could replace the text in paragraphs, headings and other elements based on several things such as a value the user enters in a textbox. For this property to change the content within an element, that element has to have an 'id' given to it through the **id** attribute.

Example:

<script type="text/javascript"> function changeTheText(){ //change the innerHTML property of the element whose id is 'text' to 'So is HTML!' document.getElementById('text').innerHTML = 'So is HTML!'; } </script> Javascript is cool! <input type='button' onclick='changeTheText()' value='Change the text' />

You can also change the text of elements based on user input:

Example:

<script type="text/javascript"> function changeInputText(){ /* change the innerHTML property of the element whose id is 'theText' to the value from the variable usersText which will take the value from the element whose id is 'usersText' */ var usersText = document.getElementById('usersText').value;
document.getElementById('theText').innerHTML = usersText; } </script> Enter some text into the textbox and click the buttton <input type="text" id="usersText" />
 <input type="button" onclick="changeInputText()" value="Change the text" />

5.4 Events and event handling

HTML Events

Not all events are significant to a program. We can simplify situations to determine which events are significant, and which events we can leave out. For example, a leaf hitting the bus is an event, but is not a significant event if we wish to model the cost effectiveness of the bus route.

As we know, HTML provides elements to allow us to create web documents. We can fill these elements with our own data and display unique web documents. We can position the elements with careful planning, and create aesthetic web sites.

This is the object-orientated part of web documents. DHTML, or Dynamic HTML supplies the event-driven side of things.

To begin using DHTML, and therefore event-driven HTML, we need to look at what events happen to our web document.

There are over 50 events in all. Some can happen to a lot of HTML elements, some only happen to specific HTML elements.

For example, moving the mouse pointer and clicking the mouse buttons create the following events:

·onmousedown
·onmousemove
·onmouseout
·onmouseover
·onmouseup
·onclick
·ondblclick

The event happens to the object. So, if the mouse pointer is over an image, say Image1, and we click it, we create an (object, event) pair, in this case (Image1, onclick).

If we move the mouse into a select box, Select1, we create (Select1, onmouseover), and so on.

We need technical definitions of the events to be able to appreciate what is actually happening.

Mouse Event	Description
onmousedown	A mouse button has been pressed
onmousemove	The mouse has been moved
onmouseout	The mouse pointer has left an element
onmouseover	The mouse pointer has entered an element
onmouseup	A mouse button has been released
onclick	A mouse button has been clicked
ondblclick	A mouse button has been double-clicked (clicked twice rapidly)

The idea behind this is to represent all the possibilities we may encounter in computer events. This doesn't allow for events like the mouse is dirty, or the mouse is upside-down - these are not relevant.

You may also notice that some actions will fire two events. Moving the mouse into or out of an element will fire both a mousemove event and a mouseover (or mouseout) event. Clicking a mouse button fires a mousedown, click and mouseup events.

The other user-type event is generated by the keyboard.

Keyboard Event	Description
onkeydown	A key has been pressed

onkeypress	onkeydown followed by onkeyup
onkeyup	A key has been released

Remaining events have a more conceptual taste. There are a few too many to list here, but they are well documented. Commonly used events include:

Event	Description
onblur	An element loses focus
onerror	An error occurs
onfocus	An element gains focus
onload	The document has completely loaded
onreset	A form reset command is issued
onscroll	The document is scrolled
onselect	The selection of element has changed
onsubmit	A form submit command is issued

We will look at the onfocus event at the end of this article, which may help your understanding of the subtler events.

Attaching Events to HTML elements

The first thing you need to know is that when onblur or onmousover or any other event is used within your HTML page, it is commonly referred to as an event handler.

Now we need to write some JavaScript to enhance the current functionality of the element, and this is where it can get complicated.

We all know what a typical HTML element looks like, and hopefully we know what an HTML element with the STYLE attribute looks like.

As a re-cap, an HTML element is:

<P>Hello from the 60's</P>

and with style:

<P STYLE="position:absolute;top:10;left:10;color:black">Hello from the 60's</P>

To attach the event handler of our choice to this <P> element, we need to notify the element of the type of event to watch out for, and also we have to tell the element what to do when it does receive the event.

To create the psychedelic 60's effect I have in mind, we need to monitor for the onmousemove event. The JavaScript to create this code is fairly simple, we change the color to a random one every time the mouse pointer moves over the <P> element.

The whole code for the <P> element therefore looks like:

<P onmousemove="style.color=Math.floor(Math.random()*16777215);" STYLE="position:absolute;top:10;left:10;color:black">Hello from the 60's</P>

Placed in an HTML document, we get:

```
<HTML>
<HEAD>
<TITLE>Hello from the 60's</TITLE></HEAD>
<BODY>
<P onmousemove="style.color=Math.floor(Math.random()*16777216);"
STYLE="position:absolute;top:10;left:10;color:black">Hello from the 60's</P>
</BODY>
</HTML>
```

When we move the mouse pointer over the text, we get a very psychedelic effect. Note that the mouse pointer must be moved, it's mere presence over the text is not an event.

The strange number, 16777216, is the total number of colors we get. We can use any of 256 red shades, 256 green shades and 256 blue shades, so the total is 256^3 = 16777216.

This is slightly obscured by the fact that the first number is 0, and so we only want the range (0 - 16777215). This is created by the Math.floor() method with the Math.random() method. Math.random() returns a number between greater than or equal to zero, but less than 1. When we floor the random function, we will never get 16777216, but will get 0 - 16777215.

Other Event Handling Techniques

This is not the only method we have for attaching events to elements, and nor is it the only method we have for implementing JavaScript.

Function Calls

We could house the JavaScript in a function, and so the 60's program becomes:

```
<HTML>
<HEAD>
<TITLE>Hello from the 60's Again</TITLE>
</HEAD>
<SCRIPT>
function randomcolor() {
event.srcElement.style.color=Math.floor(Math.random()*16777216);
}
</SCRIPT>
<BODY>

<Ponmousemove="randomcolor();"
STYLE="position:absolute;top:10;left:10;color:black">Hello from the 60's</P>
<P
```

onmousemove="randomcolor();"STYLE="position:absolute;top:50;left:20;color:black">
Hello from the 70's</P></BODY></HTML>

Now our event handler calls the randomcolor() function.

The randomcolor() function is slightly more complicated the method used before. It makes use of the Event object, which I will cover in detail in a later article.

One of the properties of the event object is the srcElement property, which contains the element that initiated the event.

We can use this style of event handler to make our code terser and more efficient. We can even use the same function twice, and independently.

Attaching Events

We also have several different techniques for attaching the event to an object. We have met one of them already, which is direct attachment of the event to an object.

We can also indirectly attach the event to the object. We can achieve the same event handling effect as before from within a script tag.

```
<HTML>
<HEAD>
<TITLE>Hello from the 60's Part 3</TITLE>
</HEAD>
<BODY>
<P ID="sixties" STYLE="position:absolute;top:10;left:10;color:black">Hello from the 60's</P>
</BODY>
</HTML>
<SCRIPT>
function sixties.onmousemove() {
event.srcElement.style.color=Math.floor(Math.random()*16777216);
```

```
}
</SCRIPT>
```

This code is fairly stylish, but causes a few more problems. As the script contains a reference to the sixties object, this object must have been created before the script can use it. If we place the <SCRIPT> element where we usually do, inside the <HEAD> element, we get an error message, because the sixties object does not yet exist. So we must place if after the sixties object has been created.

We can also create little script elements, which we assign to an object.

```
<HTML>
<HEAD>
<TITLE>Hello from the 60's IV</TITLE>
</HEAD>
<BODY>
<P ID="sixties" STYLE="position:absolute;top:10;left:10;color:black">Hello from the 60's</P>
</BODY>
</HTML>
<SCRIPT FOR="sixties" EVENT="onmousemove">
event.srcElement.style.color=Math.floor(Math.random()*16777216);
</SCRIPT>
```

To finish the article, we shall look at one of the more conceptual events, the focus event.

5.6 Button elements

Syntax **<BUTTON>...</BUTTON>**

Attribute Specifications

- NAME=CDATA (key in submitted form)
- VALUE=CDATA (value in submitted form)
- TYPE=[submit | reset | button] (type of button)
- DISABLED (disable button)

- ACCESSKEY=Character (shortcut key)
- TABINDEX=Number (position in tabbing order)
- ONFOCUS=Script (element received focus)
- ONBLUR=Script (element lost focus)
- common attributes

Contents

- Inline elements except A, INPUT, SELECT, TEXTAREA, LABEL, BUTTON, and IFRAME
- Block-level elements except FORM, ISINDEX, and FIELDSET

Contained in Block-level elements, inline elements except BUTTON

The **BUTTON** element defines a submit button, reset button, or push button. Authors can also use **INPUT** to specify these buttons, but the **BUTTON** element allows richer labels, including images and emphasis. However, **BUTTON** is new in HTML 4.0 and not as widely supported as **INPUT**. For compatibility with old browsers, **INPUT** should generally be used instead of **BUTTON**.

The **TYPE** attribute of **BUTTON** specifies the kind of button and takes the value **submit** (the default), **reset**, or **button**. The **NAME** and **VALUE** attributes determine the name/value pair sent to the server when a submit button is pushed. These attributes allow authors to provide multiple submit buttons and have the form handler take a different action depending on the submit button used.

Some examples of **BUTTON** follow:

- <BUTTON NAME=submit VALUE=modify ACCESSKEY=M>Modify information</BUTTON>
  <BUTTON NAME=submit VALUE=continue ACCESSKEY=C>Continue with application</BUTTON>
- <BUTTON ACCESSKEY=S>Submit <IMG SRC="checkmark.gif" ALT=""></BUTTON>

<BUTTON TYPE=reset ACCESSKEY=R>Reset <IMG SRC="x.gif" ALT=""></BUTTON>

- <BUTTON TYPE=button ID=toggler ONCLICK="toggle()" ACCESSKEY=H>Hide non-strict attributes</BUTTON>

The **ACCESSKEY** attribute, used throughout the preceding examples, specifies a single Unicode character as a shortcut key for pressing the button. Entities (e.g. &**eacute;**) may be used as the **ACCESSKEY** value.

The boolean **DISABLED** attribute makes the **BUTTON** element unavailable. The user is unable to push the button, the button cannot receive focus, and the button is skipped when navigating the document by tabbing.

The **TABINDEX** attribute specifies a number between 0 and 32767 to indicate the tabbing order of the button. A **BUTTON** element with **TABINDEX=0** or no **TABINDEX** attribute will be visited after any elements with a positive **TABINDEX**. Among positive **TABINDEX** values, the lower number receives focus first. In the case of a tie, the element appearing first in the HTML document takes precedence.

The **BUTTON** element also takes a number of attributes to specify client-side scripting actions for various events. In addition to the core events common to most elements, **BUTTON** accepts the following event attributes:

- **ONFOCUS**, when the element receives focus;
- **ONBLUR**, when the element loses focus.

5.7 Text box and Password elements

Introduction to forms

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

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

```
<FORM action="http://somesite.com/prog/adduser" method="post">
  <P>
  <LABEL for="firstname">First name: </LABEL>
        <INPUT type="text" id="firstname"><BR>
  <LABEL for="lastname">Last name: </LABEL>
        <INPUT type="text" id="lastname"><BR>
  <LABEL for="email">email: </LABEL>
        <INPUT type="text" id="email"><BR>
  <INPUT type="radio" name="sex" value="Male"> Male<BR>
  <INPUT type="radio" name="sex" value="Female"> Female<BR>
  <INPUT type="submit" value="Send"> <INPUT type="reset">
  </P>
</FORM>
```

5.8 The DOM 2 event model

The DOM2 Event model specification (**http://www.w3.org/TR/DOM-Level-2-Events/**) describes a standard way to create, capture, handle, and cancel events in a tree-like structure such as an (X)HTML document's object hierarchy. It also describes event propagation behavior, that is, how an event arrives at its target and what happens to it afterward.

The DOM2 approach to events accommodates the basic event model and marries important concepts from the proprietary models. This essentially means that the basic event model works exactly as advertised in a DOM2-supporting browser. Also, everything that you can do in Netscape 4 and Internet Explorer you can do in a DOM2 browser, but the syntax is different.

The hybridization of the proprietary models is evident in how events propagate in a DOM2-supporting browser. Events begin their lifecycle at the top of the hierarchy (at the **Document**) and make their way down through containing objects to the target. This is known as the capture phase because it mimics the behavior of Netscape 4. During its descent, an event may be pre-processed, handled, or redirected by any intervening object. Once the event reaches its target and the handler there has executed, the event proceeds back up the hierarchy to the top. This is known as the bubbling phase because of its obvious connections to the model of Internet Explorer 4+.

Mozilla-based browsers were the first major browsers to implement the DOM2 Events standard. These browsers include Mozilla itself, Netscape 6+, Firefox, Camino, and others. Opera 7 has nearly complete support, as does Safari (a popular MacOS browser). In fact, most browsers (with the exception of those from Microsoft) support or will soon support DOM2 Events. This is as it should be; uniformity of interface is the reason we have standards in the first place.

The fly in the ointment is that, as of version 6, Internet Explorer doesn't support DOM2 Events. Microsoft does not appear to have plans to add support in the near

future, and it's unclear whether they plan to in the medium- or long-term. So, unfortunately, Web programmers aren't likely to be free of the headache of cross-browser scripting for events any time soon and should be wary of focusing solely on the DOM2 Event specification.

Binding Handlers to Objects

The easiest way to bind event handlers to elements under DOM Level 2 is to use the (X)HTML attributes like **onclick** that you should be familiar with by now. Nothing changes for DOM2-supporting browsers when you bind events in this way, except that only support for events in the (X)HTML standard is guaranteed (though some browsers support more events).

Because there is no official DOM2 way for script in the text of event handler attributes to access an **Event** object, the preferred binding technique is to use JavaScript. The same syntax is used as with the basic event model:

```
<<p id="myElement">>Click on me<</p>>
<<p>>Not on me<</p>>

<<script type="text/javascript">>
<<!--
function handleClick(e)
{
  alert("Got a click: " + e);
 // IE5&6 will show an undefined in alert since they are not DOM2
}
document.getElementById("myElement").onclick = handleClick;
//-->>
<</script>>
```

Notice in this example how the handler accepts an argument. DOM2 browsers pass an **Event** object containing extra information about the event to handlers. The name of the argument is arbitrary, but "event," "e," and "evt" are most commonly used. We'll discuss the **Event** object in more detail in an upcoming section.

DOM2 Event Binding Methods

You can also use the new **addEventListener()** method introduced by DOM2 to engage an event handler in a page. There are three reasons you might wish to use this function instead of directly setting an object's event handler property. The first is that it enables you to bind multiple handlers to an object for the same event. When handlers are bound in this fashion, each handler is invoked when the specified event occurs, though the order in which they are invoked is arbitrary. The second reason to use **addEventListener()** is that it enables you to handle events during the capture phase (when an event "trickles down" to its target). Event handlers bound to event handler attributes like **onclick** and **onsubmit** are only invoked during the bubbling phase. The third reason is that this method enables you to bind handlers to text nodes, an impossible task prior to DOM2.

The syntax of the **addEventListener()** method is as follows:

object.addEventListener("event", handler, capturePhase);

- object is the node to which the listener is to be bound.
- "event" is a string indicating the event it is to listen for.
- handler is the function that should be invoked when the event occurs.
- capturePhase is a Boolean indicating whether the handler should be invoked during the capture phase (**true**) or bubbling phase (**false**).

For example, to register a function changeColor() as the capture-phase **mouseover** handler for a paragraph with **id** of myText you might write

```
document.getElementById('myText').addEventListener("mouseover",
changeColor, true);
```

To add a bubble phase handler swapImage():

```
document.getElementById('myText').addEventListener("mouseover",
swapImage, false);
```

Handlers are removed using **removeEventListener()** with the same arguments as given when the event was added. So to remove the first handler in the previous example (but keep the second) you would invoke

```
document.getElementById('myText').removeEventListener("mouseover",
```

changeColor, true);

We'll see some specific examples using the **addEventListener()** later on in the chapter.

Event Objects

As previously mentioned, browsers supporting DOM2 Events pass an **Event** object as an argument to handlers. This object contains extra information about the event that occurred, and is in fact quite similar to the **Event** objects of the proprietary models. The exact properties of this object depend on the event that occurred, but all **Event** objects have the read-only properties listed in Table 11-10.

Table 11-10: Properties Common to All Event Objects

Read-Only Property	Description
>bubbles	Boolean indicating whether the event bubbles
>cancelable	Boolean indicating whether the event can be canceled
>currentTarget	Node whose handler is currently executing (i.e., the node the handler is bound to)
>eventPhase	Numeric value indicating the current phase of the event flow (1 for capture, 2 if at the target, 3 for bubble)
>type	String indicating the type of the event (such as "click")
>target	Node to which the event was originally dispatched (i.e., the node at which the event occurred)

We list the properties specific to each event in the following sections as we discuss the different kinds of events DOM2-supporting browsers enable you to handle.

Mouse Events

The mouse events defined by DOM2 are those from (X)HTML. They're listed in Table 11-11. Since, under DOM2, not all events include a bubbling phase and all default actions can be canceled, Table also lists these behaviors for each event.

Table : Mouse-Related Events Supported Under DOM2 Events

Event	Bubbles?	Cancelable?
click	Yes	Yes
mousedown	Yes	Yes
mouseup	Yes	Yes
mouseover	Yes	Yes
mousemove	Yes	No
mouseout	Yes	Yes

When a mouse event occurs, the browser fills the **Event** object with the extra information shown in Table

Table : Additional Properties of the Event Object When the Event Is Mouse-Related

Property	Description
>altKey	Boolean indicating if the ALT key was depressed
>button	Numeric value indicating which mouse button was used (typically 0 for left,

Table : Additional Properties of the Event Object When the Event Is Mouse-Related

Property	Description
	1 for middle, 2 for right)
>clientX	Horizontal coordinate of the event relative to the browser's content pane
>clientY	Vertical coordinate of the event relative to the browser's content pane
>ctrlKey	Boolean indicating if the CTRL key was depressed during event
>detail	Indicating the number of times the mouse button was clicked (if at all)
>metaKey	Boolean indicating if the META key was depressed during event
>relatedTarget	Reference to a node related to the event—for example, on a mouseover it references the node the mouse is leaving; on mouseout it references the node to which the mouse is moving
>screenX	Horizontal coordinate of the event relative to the whole screen
>screenY	Vertical coordinate of the event relative to the whole screen
>shiftKey	Boolean indicating if the SHIFT key was depressed during event

The following example illustrates their use:

```
<<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">>
<<html xmlns="http://www.w3.org/1999/xhtml">>
<<head>>
<<title>>DOM2 Mouse Events<</title>>
<<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />>
<</head>>
<<body>>
<<h2>>DOM2 Mouse Events<</h2>>
<<form id="mouseform" name="mouseform" action="#" method="get">>
Alt Key down?
<<input id="altkey" type="text" />><<br />>
Control Key down?
<<input id="controlkey" type="text" />><<br />>

Meta Key down?
<<input id="metakey" type="text" />><<br />>
Shift Key down?
<<input id="shiftkey" type="text" />><<br />>

Browser coordinates of click: <<input id="clientx" type="text" />>,
                    <<input id="clienty" type="text" />> <<br />>
Screen coordinates of click: <<input id="screenx" type="text" />>,
                  <<input id="screeny" type="text" />> <<br />>
Button used: <<input id="buttonused" type="text" />><<br />><<br />>
<</form>>
<<hr />>
Click anywhere on the document...
<<script type="text/javascript">>
<<!--
function showMouseDetails(event)
```

```
{
  var theForm = document.mouseform;

  theForm.altkey.value = event.altKey;
  theForm.controlkey.value = event.ctrlKey;
  theForm.shiftkey.value = event.shiftKey;
  theForm.metakey.value = event.metaKey;
  theForm.clientx.value = event.clientX;
  theForm.clienty.value = event.clientY;
  theForm.screenx.value = event.screenX;
  theForm.screeny.value = event.screenY;
  if (event.button == 0)
    theForm.buttonused.value = "left";
  else if (event.button ==  1)
    theForm.buttonused.value = "middle";
  else
    theForm.buttonused.value = "right";
}
document.addEventListener("click", showMouseDetails, true);
//-->>
<</script>>
<</body>>
<</html>>
```

The result of a click is shown in Figure .

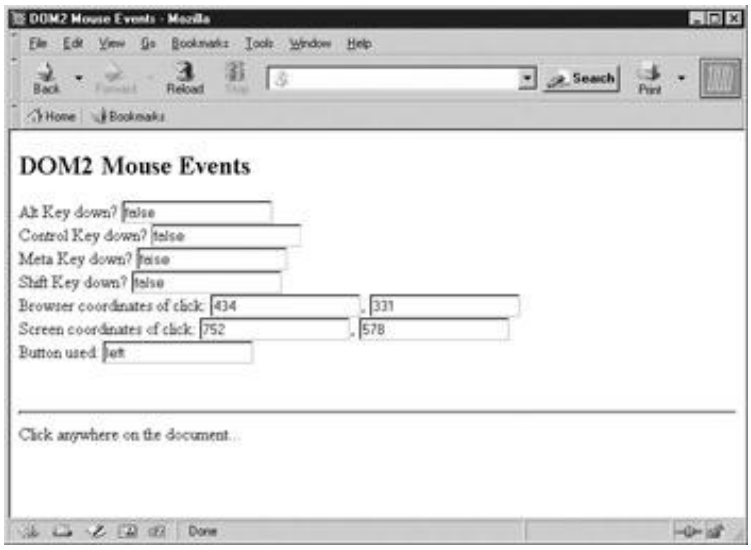


Figure: Contextual information is passed in through the **Event** object.

Keyboard Events

Surprisingly, DOM Level 2 does not define keyboard events. They will be specified in a future standard, the genesis of which you can see in DOM Level 3. Fortunately, because (X)HTML allows **keyup**, **keydown**, and **keypress** events for many elements, you'll find that browsers support them. Furthermore, with IE dominating the picture, you still have that event model to fall back on. Table lists the keyboard-related events for DOM2-compliant browsers, as well as their behaviors.

Table: Keyboard Events Supported by Most Browsers

Event	Bubbles?	Cancelable?
keyup	Yes	Yes
keydown	Yes	Yes
keypress	Yes	Yes

The Mozilla-specific key-related properties of the **Event** object are listed in Table .

Table: Additional Properties of the Event Object for Key-Related Events in Mozilla

Property	Description

Table: Additional Properties of the Event Object for Key-Related Events in Mozilla

Property	Description
>altKey	Boolean indicating if the ALT key was depressed
>charCode	For printable characters, a numeric value indicating the Unicode value of the key depressed
>ctrlKey	Boolean indicating if the CTRL key was depressed during event
>isChar	Boolean indicating whether the keypress produced a character (useful because some key sequences such as CTRL-ALT do not)
>keyCode	For non-printable characters, a numeric value indicating the Unicode value of the key depressed
>metaKey	Boolean indicating if the META key was depressed during event
>shiftKey	Boolean indicating if the SHIFT key was depressed during event

Browser Events

DOM2 browsers support the familiar browser and form-related events found in all major browsers. The list of these events is found in Table .

Table: Browser- and Form-Related DOM2 Events and Their Behaviors

Event	Bubbles?	Cancelable?
load	No	No

Table: Browser- and Form-Related DOM2 Events and Their Behaviors

Event	Bubbles?	Cancelable?
unload	No	No
abort	Yes	No
error	Yes	No
select	Yes	No
change	Yes	No
submit	Yes	Yes
reset	Yes	No
focus	No	No
Blur	No	No
resize	Yes	No
scroll	Yes	No

UI Events

Although DOM Level 2 builds primarily on those events found in the (X)HTML specification (and DOM Level 0), it adds a few new User Interface (UI) events to round out the field. These events are prefixed with "DOM" to distinguish them from "normal" events. These events are listed in Table.

Table: UI-Related DOM2 Events and Their Behaviors

Event	Bubbles?	Cancelable?
DOMFocusIn	Yes	No
DOMFocusOut	Yes	No
DOMActivate	Yes	Yes

The need for and meaning of these events is not necessarily obvious. **DOMFocusIn** and **DOMFocusOut** are very similar to the traditional **focus** and **blur** events, but can be applied to any element, not just form fields. The **DOMActivate** event is fired when an object is receiving activity from the user. For example, it fires on a link when it is clicked and on a select menu when the user activates the pull-down menu. This event is useful when you don't care how the user invokes the element's functionality, just that it is being used. For example, instead of using both an **onclick** and **onkeypress** handler to trap link activation (via the mouse or keyboard) you could register to receive the **DOMActivate** event. While these new events are rarely used, it is helpful to be aware of them should you encounter them in new scripts.

Event Creation

The last DOM 2 Event topic we mention is not often used nor implemented in browsers, but is interesting nonetheless—event creation. The DOM2 Event specification allows for synthetic events to be created by the user using **document.createEvent()**. You first create the type of event you want, say an HTML-related event:

```
evt = document.createEvent("HTMLEvents");
```

Then once your event is created you pass it various attributes related to the event type. Here, for example, we pass the type of event "click" and Boolean values indicating it is bubble-able and cancelable:

```
evt.initEvent("click","true","true");
```

Finally, we find a node in the document tree and dispatch the event to it:

```
currentNode.dispatchEvent(evt);
```

The event then is triggered and reacts as any other event.

The following example shows DOM2 event creation in action and allows you to move around the tree and fire clicks at various locations. **The addEventListener()** and **removeEventListener()** are added into the example so you do not have to observe click events until you are ready.

```
<<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">>
<<html xmlns="http://www.w3.org/1999/xhtml">>
<<head>>
<<title>>DOM2 Event Creation<</title>>
<<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />>
<</head>>
<<body>>
<<h2>>DOM2 Event Creation<</h2>>
<<form id="mouseform" name="mouseform" action="#" method="get">>
Browser coordinates of click: <<input id="clientx" type="text" />>,
                    <<input id="clienty" type="text" />> <<br />>
<</form>>
<<br />><<hr />><<br />>
<<script type="text/javascript">>
<<!--
// DOM 2 Only - no IE6 support

function showMouseDetails(event)
{
 document.mouseform.clientx.value = event.clientX;
 document.mouseform.clienty.value = event.clientY;
}
function makeEvent()
{
 evt = document.createEvent("HTMLEvents");
 evt.initEvent("click","true","true");
 currentNode.dispatchEvent(evt);
}
function startListen()
{
```

```
 document.addEventListener("click", showMouseDetails, true);
}
function stopListen()
{
 document.removeEventListener("click", showMouseDetails, true);
}

startListen();
//-->>
<</script>>
<<form action="#" method="get" id="myForm" name="myForm">>
  Current Node: <<input type="text" name="statusField" value="" />>
    <<br />>
  <<input type="button" value="parent" onclick="if
(currentNode.parentNode) currentNode = currentNode.parentNode;
document.myForm.statusField.value = currentNode.nodeName;" />>

  <<input type="button" value="First Child" onclick="if
(currentNode.firstChild) currentNode = currentNode.firstChild;
document.myForm.statusField.value = currentNode.nodeName;" />>

  <<input type="button" value="Next Sibling" onclick="if
(currentNode.nextSibling) currentNode = currentNode.nextSibling;
document.myForm.statusField.value = currentNode.nodeName;" />>

  <<input type="button" value="Previous Sibling" onclick="if
(currentNode.previousSibling) currentNode = currentNode.previousSibling;
document.myForm.statusField.value = currentNode.nodeName;" />>
    <<br />><<br />>
  <<input type="button" value="Start Event Listener"
onclick="startListen();" />>
```

```
    <<input type="button" value="Stop Event Listener"
 onclick="stopListen();" />>
    <<br />><<br />>
    <<input type="button" value="Create Event" onclick="makeEvent();" />>
<</form>>
<<script type="text/javascript">>
<<!--
 var currentNode = document.body;
 document.myForm.statusField.value = currentNode.nodeName;
//-->>
<</script>>
<</body>>
<</html>>
```

There are a number of details to DOM2 Event creation that we forgo primarily because it is not widely implemented in browsers. In fact with much of this section it should always be kept in mind that the DOM2 currently is probably not the best approach to making events work across browsers since it is not supported by Internet Explorer. We review the sorry state of affairs for event support in browsers briefly now.

5.9 The navigator object

The JavaScript navigator object is the object representation of the client internet browser or web navigator program that is being used. This object is the top level object to all others.

Navigator Properties

- appCodeName - The name of the browser's code such as "Mozilla".
- appMinorVersion - The minor version number of the browser.
- appName - The name of the browser such as "Microsoft Internet Explorer" or

"Netscape Navigator".

- appVersion - The version of the browser which may include a compatability value and operating system name.
- cookieEnabled - A boolean value of true or false depending on whether cookies are enabled in the browser.
- cpuClass - The type of CPU which may be "x86"
- mimeTypes - An array of MIME type descriptive strings that are supported by the browser.
- onLine - A boolean value of true or false.
- opsProfile
- platform - A description of the operating system platform. In my case it is "Win32" for Windows 95.
- plugins - An array of plug-ins supported by the browser and installed on the browser.
- systemLanguage - The language being used such as "en-us".
- userAgent - In my case it is "Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)" which describes the browser associated user agent header.
- userLanguage - The languge the user is using such as "en-us".
- userProfile

Methods

- javaEnabled() - Returns a boolean telling if the browser has JavaScript enabled.
- taintEnabled() - Returns a boolean telling if the browser has tainting enabled. Tainting is a security protection mechanism for data.

Navigator Objects

Most Navigator objects have corresponding HTML tags. Refer to the appropriate object section for more information.

- MimeType - Allows access to information about MIME types supported by the

browser.

- plugin - Access to information about all plugins the browser supports. <EMBED>
- window - The browser frame or window.
    - frame- Allows access to all frames in the window. <FRAME>
    - history - The URLs previously accessed in a window.
    - location - Represents a URL.
    - document - The HTML document loaded in the window. <BODY>
        - anchor object anchors array - Allows access to all anchors in the document. <A NAME="">
        - applet - Allows access to all applets in the document.
        - area - Allows access to an area in a client image map. <MAP>
        - image object images array - Allows access to all images in the document. <IMG>
        - link object links array - Allows access to all links in the document. <A HREF="">
        - layer - Allows access to HTML layers.
        - forms - Allows access to all forms in the document. <FORM>
            - button - Allows access to a form button exclusive of a submit or reset button. <INPUT TYPE="button">
            - checkbox - Allows access to a formcheckbox. <INPUT TYPE="checkbox">
            - element - Allows access to buttons or fields in a form.
            - FileUpLoad - Allows access to a form file upload element. <INPUT TYPE="file">
            - hidden - Allows access to a form hidden field. <INPUT TYPE="hidden">
            - option
            - password - Allows access to a form password field. <INPUT TYPE="password">
            - radio - Allows access to a form radio button set. <INPUT TYPE="radio">

- reset - Allows access to a form reset button.. <INPUT TYPE="reset">
- select - Allows access to a form selected list with the option allowing access to selected elements in the select list. <SELECT>
- submit - Allows access to a form submit button. <INPUT TYPE="submit">
- text - Allows access to a form text field. <INPUT TYPE="text">
- textarea - Allows access to a form text area field. <TEXTAREA>

- embeds - Allows access to embedded plug ins.

5.10 DOM tree traversal and modification.

Its TreeWalker, NodeIterator, and NodeFilter interfaces provide easy-to-use, robust, selective traversal of a document's contents.

The interfaces found within this section are not mandatory. A DOM application may use the hasFeature(feature, version) method of the DOMImplementation interface with parameter values "Traversal" and "2.0" (respectively) to determine whether or not this module is supported by the implementation. In order to fully support this module, an implementation must also support the "Core" feature defined defined in the DOM Level 2 Core specification [DOM Level 2 Core]. Please refer to additional information about conformance in the DOM Level 2 Core specification [DOM Level 2 Core].

NodeIterators and TreeWalkers are two different ways of representing the nodes of a document subtree and a position within the nodes they present. A NodeIterator presents a flattened view of the subtree as an ordered sequence of nodes, presented in document

order. Because this view is presented without respect to hierarchy, iterators have methods to move forward and backward, but not to move up and down. Conversely, a TreeWalker maintains the hierarchical relationships of the subtree, allowing navigation of this hierarchy. In general, TreeWalkers are better for tasks in which the structure of the document around selected nodes will be manipulated, while NodeIterators are better for tasks that focus on the content of each selected node.

NodeIterators and TreeWalkers each present a view of a document subtree that may not contain all nodes found in the subtree. In this specification, we refer to this as the logical view to distinguish it from the physical view, which corresponds to the document subtree per se. When an iterator or TreeWalker is created, it may be associated with a NodeFilter, which examines each node and determines whether it should appear in the logical view. In addition, flags may be used to specify which node types should occur in the logical view.

NodeIterators and TreeWalkers are dynamic - the logical view changes to reflect changes made to the underlying document. However, they differ in how they respond to those changes. NodeIterators, which present the nodes sequentially, attempt to maintain their location relative to a position in that sequence when the sequence's contents change. TreeWalkers, which present the nodes as a filtered tree, maintain their location relative to their current node and remain attached to that node if it is moved to a new context. We will discuss these behaviors in greater detail below.

NodeIterators

A NodeIterator allows the members of a list of nodes to be returned sequentially. In the current DOM interfaces, this list will always consist of the nodes of a subtree, presented in document order. When an iterator is first created, calling its nextNode() method returns the first node in the logical view of the subtree; in most cases, this is the root of the subtree. Each successive call advances the NodeIterator through the list, returning the next node available in the logical view. When no more nodes are visible, nextNode() returns null.

NodeIterators are created using the createNodeIterator method found in the DocumentTraversal interface. When a NodeIterator is created, flags can be used to determine which node types will be "visible" and which nodes will be "invisible" while traversing the tree; these flags can be combined using the OR operator. Nodes that are "invisible" are skipped over by the iterator as though they did not exist.

The following code creates an iterator, then calls a function to print the name of each element:

```
NodeIterator iter=
 ((DocumentTraversal)document).createNodeIterator(
     root, NodeFilter.SHOW_ELEMENT, null);

while (Node n = iter.nextNode())
   printMe(n);
```

Moving Forward and Backward

NodeIterators present nodes as an ordered list, and move forward and backward within this list. The iterator's position is always either between two nodes, before the first node, or after the last node. When an iterator is first created, the position is set before the first item. The following diagram shows the list view that an iterator might provide for a particular subtree, with the position indicated by an asterisk '*' :

```
* A B C D E F G H I
```

Each call to nextNode() returns the next node and advances the position. For instance, if we start with the above position, the first call to nextNode() returns "A" and advances the iterator:

```
[A] * B C D E F G H I
```

The position of a NodeIterator can best be described with respect to the last node returned, which we will call the reference node. When an iterator is created, the first node is the reference node, and the iterator is positioned before the reference node. In these diagrams, we use square brackets to indicate the reference node.

A call to previousNode() returns the previous node and moves the position backward. For instance, if we start with the NodeIterator between "A" and "B", it would return "A" and move to the position shown below:

* [A] B C D E F G H I

If nextNode() is called at the end of a list, or previousNode() is called at the beginning of a list, it returns null and does not change the position of the iterator. When a NodeIterator is first created, the reference node is the first node:

* [A] B C D E F G H I

Robustness

A NodeIterator may be active while the data structure it navigates is being edited, so an iterator must behave gracefully in the face of change. Additions and removals in the underlying data structure do not invalidate a NodeIterator; in fact, a NodeIterator is never invalidated unless its detach() method is invoked. To make this possible, the iterator uses the reference node to maintain its position. The state of an iterator also depends on whether the iterator is positioned before or after the reference node.

If changes to the iterated list do not remove the reference node, they do not affect the state of the NodeIterator. For instance, the iterator's state is not affected by inserting new nodes in the vicinity of the iterator or removing nodes other than the reference node. Suppose we start from the following position:

A B C [D] * E F G H I

Now let's remove "E". The resulting state is:

A B C [D] * F G H I

If a new node is inserted, the NodeIterator stays close to the reference node, so if a node is inserted between "D" and "F", it will occur between the iterator and "F":

A B C [D] * X F G H I

Moving a node is equivalent to a removal followed by an insertion. If we move "I" to the position before "X" the result is:

A B C [D] * I X F G H

If the reference node is removed from the list being iterated over, a different node is selected as the reference node. If the reference node's position is before that of the NodeIterator, which is usually the case after nextNode() has been called, the nearest node before the iterator is chosen as the new reference node. Suppose we remove the "D" node, starting from the following state:

A B C [D] * F G H I

The "C" node becomes the new reference node, since it is the nearest node to the NodeIterator that is before the iterator:

A B [C] * F G H I

If the reference node is after the NodeIterator, which is usually the case after previousNode() has been called, the nearest node after the iterator is chosen as the new reference node. Suppose we remove "E", starting from the following state:

A B C D * [E] F G H I

The "F" node becomes the new reference node, since it is the nearest node to the NodeIterator that is after the iterator:

A B C D * [F] G H I

As noted above, moving a node is equivalent to a removal followed by an insertion. Suppose we wish to move the "D" node to the end of the list, starting from the following state:

A B C [D] * F G H I C

The resulting state is as follows:

A B [C] * F G H I D

One special case arises when the reference node is the last node in the list and the reference node is removed. Suppose we remove node "C", starting from the following state:

A B * [C]

According to the rules we have given, the new reference node should be the nearest node after the NodeIterator, but there are no further nodes after "C". The same situation can arise when previousNode() has just returned the first node in the list, which is then removed. Hence: If there is no node in the original direction of the reference node, the nearest node in the opposite direction is selected as the reference node:

A [B] *

If the NodeIterator is positioned within a block of nodes that is removed, the above rules clearly indicate what is to be done. For instance, suppose "C" is the parent node of "D", "E", and "F", and we remove "C", starting with the following state:

A B C [D] * E F G H I D

The resulting state is as follows:

A [B] * G H I D

Finally, note that removing a NodeIterator's root node from its parent does not alter the list being iterated over, and thus does not change the iterator's state.

Visibility of Nodes

The underlying data structure that is being iterated may contain nodes that are not part of the logical view, and therefore will not be returned by the NodeIterator. If nodes that are to be excluded because of the value of the whatToShow flag, nextNode() returns the next visible node, skipping over the excluded "invisible" nodes. If a NodeFilter is present, it is applied before returning a node; if the filter does not accept the node, the process is repeated until a node is accepted by the filter and is returned. If no visible nodes are encountered, a null is returned and the iterator is positioned at the end of the list. In this case, the reference node is the last node in the list, whether or not it is visible. The same approach is taken, in the opposite direction, for previousNode().

In the following examples, we will use lowercase letters to represent nodes that are in the data structure, but which are not in the logical view. For instance, consider the following list:

A [B] * c d E F G

A call to nextNode() returns E and advances to the following position:

A B c d [E] * F G

Nodes that are not visible may nevertheless be used as reference nodes if a reference node is removed. Suppose node "E" is removed, started from the state given above. The resulting state is:

A B c [d] * F G

Suppose a new node "X", which is visible, is inserted before "d". The resulting state is:

A B c X [d] * F G

Note that a call to previousNode() now returns node X. It is important not to skip over invisible nodes when the reference node is removed, because there are cases, like the one just given above, where the wrong results will be returned. When "E" was removed, if the new reference node had been "B" rather than "d", calling previousNode() would not return "X".

UNIT - 6

DYNAMIC DOCUMENTS WITH JAVASCRIPT

UNIT - 6

DYNAMIC DOCUMENTS WITH JAVASCRIPT

6.1 Introduction to dynamic documents

Dynamic HTML is not a new markup language

- A dynamic HTML document is one whose tag attributes, tag contents, or element style properties can be changed after the document has been and is still being displayed by a browser

- We will discuss only W3C standard approaches

- All examples in this chapter, except the last, use the DOM 0 event model and work with both IE6 and NS6

- To make changes in a document, a script must be able to address the elements of the document using the DOM addresses of those elements

6.2 Positioning elements

For DHTML content developers, the most important feature of CSS is the ability to use ordinary CSS style attributes to specify the visibility, size, and precise position of individual elements of a document. In order to do DHTML programming, it is important to understand how these style attributes work. They are summarized in Table and documented in more detail in the sections that follow.

Table. CSS positioning and visibility attributes

Attribute(s)	Description
position	Specifies the type of positioning applied to an element

top, left	Specifies the position of the top and left edges of an element
bottom, right	Specifies the position of the bottom and right edges of an element
width, height	Specifies the size of an element
z-index	Specifies the "stacking order" of an element relative to any overlapping elements; defines a third dimension of element positioning
display	Specifies how and whether an element is displayed
visibility	Specifies whether an element is visible
clip	Defines a "clipping region" for an element; only portions of the element within this region are displayed
overflow	Specifies what to do if an element is bigger than the space allotted for it

The Key to DHTML: The position Attribute

The CSS position attribute specifies the type of positioning applied to an element. The four possible values for this attribute are:

static

This is the default value and specifies that the element is positioned according to the normal flow of document content (for most Western languages, this is left to right and top to bottom.) Statically positioned elements are not DHTML elements and cannot be positioned with the top, left, and other attributes. To use DHTML positioning techniques with a document element, you must first set its position attribute to one of the other three values.

absolute

This value allows you to specify the position of an element relative to its containing element. Absolutely positioned elements are positioned independently of all other elements and are not part of the flow of statically positioned elements.

An absolutely positioned element is positioned either relative to the <body> of the document or, if it is nested within another absolutely positioned element, relative to that element. This is the most commonly used positioning type for DHTML.

fixed

This value allows you to specify an element's position with respect to the browser window. Elements with fixed positioning do not scroll with the rest of the document and thus can be used to achieve frame-like effects. Like absolutely positioned elements, fixed-position elements are independent of all others and are not part of the document flow. Fixed positioning is a CSS2 feature and is not supported by fourth-generation browsers. (It is supported in Netscape 6 and IE 5 for the Macintosh, but it is not supported by IE 5 or IE 6 for Windows).

relative

When the position attribute is set to relative, an element is laid out according to the normal flow, and its position is then adjusted relative to its position in the normal flow. The space allocated for the element in the normal document flow remains allocated for it, and the elements on either side of it do not close up to fill in that space, nor are they "pushed away" from the new position of the element. Relative positioning can be useful for some static graphic design purposes, but it is not commonly used for DHTML effects.

Specifying the Position and Size of Elements

Once you have set the position attribute of an element to something other than static, you can specify the position of that element with some combination of the left , top, right, and bottom attributes. The most common positioning technique is to specify the left and top attributes, which specify the distance from the left edge of the containing element (usually the document itself) to the left edge of the element, and the distance from the top edge of the container to the top edge of the element. For example, to place an element 100 pixels from the left and 100 pixels from the top of the document, you can specify CSS styles in a style attribute as follows:

The containing element relative to which a dynamic element is positioned is not necessarily the same as the containing element within which the element is defined in the document source. Since dynamic elements are not part of normal element flow, their positions are not specified relative to the static container element within which they are defined. Most dynamic elements are positioned relative to the document (the <body> tag) itself. The exception is dynamic elements that are defined within other dynamic elements. In this case, the nested dynamic element is positioned relative to its nearest dynamic ancestor.

Although it is most common to specify the position of the upper-left corner of an element with left and top, you can also use right and bottom to specify the position of the bottom and right edges of an element relative to the bottom and right edges of the containing element. For example, to position an element so that its bottom-right corner is at the bottom-right of the document (assuming it is not nested within another dynamic element), use the following styles:

position: absolute; right: 0px; bottom: 0px;

To position an element so that its top edge is 10 pixels from the top of the window and its right edge is 10 pixels from the right of the window, you can use these styles:

position: fixed; right: 10px; top: 10px;

Note that the right and bottom attributes are newer additions to the CSS standard and are not supported by fourth-generation browsers, as top and left are.

In addition to the position of elements, CSS allows you to specify their size. This is most commonly done by providing values for the width and height style attributes. For example, the following HTML creates an absolutely positioned element with no content. Its width, height, and background-color attributes make it appear as a small blue square:

```
<div style="position: absolute; left: 10px; right: 10px;
        width: 10px; height: 10px; background-color: blue">
```

```
</div>
```

Another way to specify the width of an element is to specify a value for both the left and right attributes. Similarly, you can specify the height of an element by specifying both top and bottom. If you specify a value for left, right, and width, however, the width attribute overrides the right attribute; if the height of an element is over-constrained, height takes priority over bottom.

Bear in mind that it is not necessary to specify the size of every dynamic element. Some elements, such as images, have an intrinsic size. Furthermore, for dynamic elements that contain text or other flowed content, it is often sufficient to specify the desired width of the element and allow the height to be determined automatically by the layout of the element's content.

In the previous positioning examples, values for the position and size attributes were specified with the suffix "px". This stands for pixels. The CSS standard allows measurements to be done in a number of other units, including inches ("in"), centimeters ("cm"), points ("pt"), and ems ("em" -- a measure of the line height for the current font). Pixel units are most commonly used with DHTML programming. Note that the CSS standard requires a unit to be specified. Some browsers may assume pixels if you omit the unit specification, but you should not rely on this behavior.

Instead of specifying absolute positions and sizes using the units shown above, CSS also allows you to specify the position and size of an element as a percentage of the size of the containing element. For example, the following HTML creates an empty element with a black border that is half as wide and half as high as the containing element (or the browser window) and centered within that element:

```
<div style="position: absolute; left: 25%; top: 25%; width: 50%; height: 50%;
        border: 2px solid black">
</div>
```

Element size and position details

It is important to understand some details about how the left , right, width, top, bottom, and height attributes work. First, width and height specify the size of an element's content area only; they do not include any additional space required for the element's padding, border, or margins. To determine the full onscreen size of an element with a border, you must add the left and right padding and left and right border widths to the element width, and you must add the top and bottom padding and top and bottom border widths to the element's height.

Since width and height specify the element content area only, you might think that left and top (and right and bottom) would be measured relative to the content area of the containing element. In fact, the CSS standard specifies that these values are measured relative to the outside edge of the containing element's padding (which is the same as the inside edge of the element's border).

Let's consider an example to make this clearer. Suppose you've created a dynamically positioned container element that has 10 pixels of padding all the way around its content area and a 5 pixel border all the way around the padding. Now suppose you dynamically position a child element inside this container. If you set the left attribute of the child to "0 px", you'll discover that the child is positioned with its left edge right up against the inner edge of the container's border. With this setting, the child overlaps the container's padding, which presumably was supposed to remain empty (since that is the purpose of padding). If you want to position the child element in the upper left corner of the container's content area, you should set both the left and top attributes to "10px". Figure helps to clarify this.

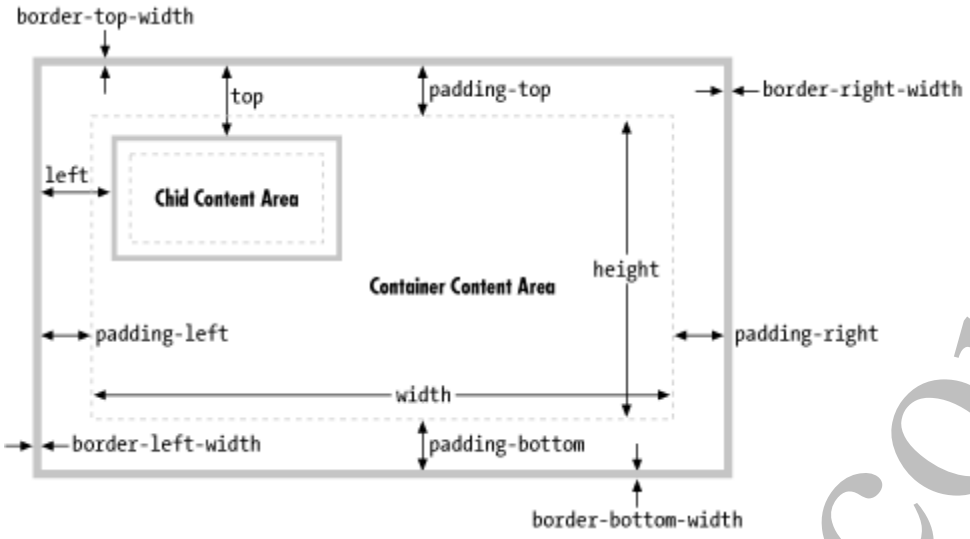


Figure Dynamically positioned container and child elements with some CSS attributes

Now that you understand that width and height specify the size of an element's content area only and that the left, top, right, and bottom attributes are measured relative to the containing element's padding, there is one more detail you must be aware of: Internet Explorer Versions 4 through 5.5 for Windows (but not IE 5 for the Mac) implement the width and height attributes incorrectly and include an element's border and padding (but not its margins). For example, if you set the width of an element to 100 pixels and place a 10-pixel margin and a 5-pixel border on the left and right, the content area of the element ends up being only 70 pixels wide in these buggy versions of Internet Explorer.

In IE 6, the CSS position and size attributes work correctly when the browser is in standards mode and incorrectly (but compatibly with earlier versions) when the browser is in compatibility mode. Standards mode, and hence correct implementation of the CSS "box model," is triggered by the presence of a <!DOCTYPE> tag at the start of the document, declaring that the document adheres to the HTML 4.0 (or later) standard or some version of the XHTML standards. For example, any of the following three HTML document type declarations cause IE 6 to display documents in standards mode:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Strict//EN">

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
```

Netscape 6 and the Mozilla browser handle the width and height attributes correctly. But these browsers also have standards and compatibility modes, just as IE does. The absence of a <!DOCTYPE> declaration puts the Netscape browser in quirks mode, in which it mimics certain (relatively minor) nonstandard layout behaviors of Netscape 4. The presence of <!DOCTYPE> causes the browser to break compatibility with Netscape 4 and correctly implement the standards.

6.3 Moving elements

```
id = window.setInterval("animate();", 100);
```

A rather rare, but still used, DHTML scenario is not only positioning an element, but also moving and therefore animating an element. To do so, window.setTimeout() and window.setInterval() come in handy. The following code animates an ad banner diagonally over the page. The only potential issue is how to animate the position. For the Internet Explorer properties (posLeft, posTop), just adding a value suffices. For left and top, you first have to determine the old position and then add a value to it. The JavaScript function parseInt() extracts the numeric content from a string like "123px". However, parseInt() returns NaN if no value is found in left or top. Therefore, the following helper function takes care of this situation; in this case, 0 is returned instead:

```
function myParseInt(s) {
  var ret = parseInt(s);
  return (isNaN(ret) ? 0 : ret);
}
```

Then, the following code animates the banner and stops after 50 iterations:

Animating an Element (animate.html; excerpt)

```
<script language="JavaScript"
  type="text/javascript">
var nr = 0;
var id = null;
function animate() {
  nr++;
  if (nr > 50) {
    window.clearInterval(id);
    document.getElementById("element").style.visibility = "hidden";
  } else {
    var el = document.getElementById("element");
    el.style.left =
      (myParseInt(el.style.left) + 5) + "px";
    el.style.posLeft += 5;
    el.style.top =
      (myParseInt(el.style.top) + 5) + "px";
    el.style.posTop += 5;
  }
}
window.onload = function() {
  id = window.setInterval("animate();", 100);
};
</script>
<h1>My Portal</h1>
<div id="element" style="position: absolute;
background-color: #eee; border: 1px solid">
JavaScript Phrasebook
</div>
```

6.4 Element visibility

The visibility property of an element controls whether it is displayed

- The values are visible and hidden

- Suppose we want to toggle between hidden and visible, and the element's DOM address is dom

```
if (dom.visibility == "visible"
dom.visibility = "hidden";
else
dom.visibility = "visible";
```

--> SHOW showHide.html

6.5 Changing colors and fonts

In order to change text colors, you will need two things:

1. A command to change the text.

2. A color (hex) code.

Section 1: Changing Full-Page Text Colors

You have the ability to change full-page text colors over four levels:

<TEXT="######"> -- This denotes the full-page text color.

<LINK="######"> -- This denotes the color of the links on your page.

<ALINK="######"> -- This denotes the color the link will flash when clicked upon.

<VLINK="######"> -- This denotes the colors of the links after they have been visited.

These commands come right after the <TITLE> commands. Again, in that position they affect everything on the page. **Also...** place them all together inside the same command along with any background commands. Something like this:

< BODY BGCOLOR="######" TEXT="######" LINK="######" VLINK="######">

<VLINK="#FFFFFF">

Section 2: Changing Specific Word Color

But I only want to change one word's color!

You'll use a color (hex) code to do the trick. Follow this formula:

<FONT COLOR="######">text text text text text</FONT>

It's a pain in the you-know-where, but it gets the job done. It works with all H commands and text size commands. Basically, if it's text, it will work.

Using Cascading Style Sheets (CSS) to Change Text Colors

There isn't enough space to fully describe what CSS is capable of in this article, but we have several articles here that can get you up to speed in no time! For a great tutorial on using CSS to change color properties, check out this article by Vincent Wright.

A quick intro to CSS is in order, so let's describe it a bit. CSS is used to define different elements on your web page. These elements include text colors, link colors, page background, tables, forms--just about every aspect of the style of the web page. You can use CSS inline, much like the HTML above, or you can, more preferably, include theh style sheet within the HEAD tags on your page, as in this example:

```
<STYLE type=text/css>
A:link {
 COLOR: red /*The color of the link*/
}
```

```
A:visited {
 COLOR: #800080 /*The color of the visited link*/
}
A:hover {
 COLOR: green /*The color of the mouseover or 'hover' link*/
}
BODY { COLOR: #800080 /*The color of all the other text within the body of the page*/
{
</STYLE>
```

Alternately, you can include the CSS that is between the STYLE tags above, and save it in a file that you could call "basic.css" which would be placed in the root directory of your website. You would then refer to that style sheet by using a link that goes between the HEAD tags in your web page, like this:

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

As you can see in the example above, you can refer to the colors using traditional color names, or hex codes as described above.

The use of CSS is vastly superior to using inline FONT tags and such, as it separates the content of your site from the style of your site, simplifying the process as you create more pages or change the style of elements. If you are using an external style sheet, you can make the change once in the style sheet, and it will be applied to your entire site. If you choose to include the style sheet itself within the HEAD tags as shown above, then you will have to make those changes on every page on your site.

CSS is such a useful tool in your web developer arsenal, you should definitely take the time to read more about it in our CSS Tutorials section.

6.5 Dynamic content

In the early days of the Web, once a Web page was loaded into a client's browser, changing the information displayed on the page required another call to the Web server.

The user interacted with the Web page by submitting forms or clicking on links to other Web pages, and the Web server delivered a new page to the client.

Any customization of information - such as building a Web page on the fly from a template - required that the Web server spent additional time processing the page request.

In short dynamic content is about changing the content of a Web page by inserting and deleting elements or the content inside elements before, or after, a Web page has been loaded into a client's browser.

UNIT - 7

XML

7.1 Introduction

7.2 Syntax

7.3 Document structure

7.4 Document Type definitions

7.5 Namespaces

7.5 XML schemas

7.6 Displaying raw XML documents

7.7 Displaying XML documents with CSS

7.8 XSLT style sheets

7.9 XML processors;

7.10 Web services.

UNIT - 7

XML

7.1 Introduction

SGML is a meta-markup language is a language for defining markup language it can describe a wide variety of document types.

_ Developed in the early 1980s; In 1986 SGML was approved by ISO std.

_ HTML was developed using SGML in the early 1990s - specifically for Web documents.

_ Two problems with HTML:

1. HTML is defined to describe the general form and layout of information without considering its meaning.

2. Fixed set of tags and attributes. Given tags must fit every kind of document. No way to find particular information

3. There are no restrictions on arrangement or order of tag appearance in document. For example, an opening tag can appear in the content of an element, but its corresponding closing tag can appear after the end of the element in which it is nested.

Eg : Now is the time

_ One solution to the first problems is to allow for group of users with common needs to define their own tags and attributes and then use the SGML standard to define a new markup language to meet those needs. Each application area would have its own markup language.

_ Use SGML to define a new markup language to meet those needs

_ Problem with using SGML:

1. It's too large and complex to use and it is very difficult to build a parser for it. SGML includes a large number of capabilities that are only rarely used.

2. A program capable of SGML documents would be very large and costly to develop.

3. SGML requires that a formal definition be provided with each new markup language. So having area-specific markup language is a good idea, basing them on SGML is not.

_ A better solution: Define a simplified version of SGML and allow users to define their own markup languages based on it. XML was designed to be that simplified version of SGML.

_ XML is not a replacement for HTML . Infact two have different goals

_ HTML is a markup language used to describe the layout of any kind of information

_ XML is a meta-markup language that provides framework for defining specialized markup languages

_ Instead of creating text file like

<html>

<head><title>name</title></head>……

Syntax

<name>

<first> nandini </first>

<last> sidnal </last>

</name>

XML is much larger then text file but makes easier to write software that accesses the information by giving structure to data.

_ XML is a very simple and universal way of storing and transferring any textual kind

_ XML does not predefine any tags

- XML tag and its content, together with closing tag _ element

_ XML has no hidden specifications

_ XML based markup language as _ tag set

_ Document that uses XML based markup language _ XML document

_ An XML processor is a program that parses XML documents and provides the parts to an application

_ Both IE7 and FX2 support basic XML XML is a meta language for describing mark-up languages. It provides a facility to define tags and the structural relationship between them

What is XML?

• XML stands for EXtensible Markup Language

• XML is a markup language much like HTML

• XML was designed to carry data, not to display data

• XML tags are not predefined. You must define your own tags

• XML is designed to be self-descriptive

• XML is a W3C Recommendation

XML is not a replacement for HTML.

XML syntax rules:

The syntax of XML can be thought of at two distinct levels.

1. There is the general low-level syntax of XML that imposes its rules on all XML documents.

2. The other syntactic level is specified by either document type definitions or XML schemas. These two kinds of specifications impose structural syntactic rules on documents written with specific XML tag sets.

_ DTDs and XML schema specify the set of tags and attributes that can appear in particular document or collection of documents, and also the orders and various arrangements in which they can appear.

DTD's and XML schema can be used to define a XML markup language.

XML document can include several different kinds of statements.

• Data elements

• Markup declarations - instructions to XML parser

• Processing instructions – instructions for an applications program that will process the data described in the document. The most common of these are the data elements of the document. XML document may also include markup declarations, which are instructions to the XML parser, and processing instructions, which are instructions for an application program that will process the data described in the document.

All XML document must begin with XML declaration. It identifies the document as being XML and provides the version no. of the XML standard being used. It will also include encoding standard. It is a first line of the XHTML document.

Comments are same as HTML. <!-- This is a comment -->

XML names are used to name elements and attributes.

XML names must begin with a letter or underscore and can include digits, hyphens, and periods.

XML names are case sensitive. , the tag <Letter> is different from the tag <letter>.

There is no length limitation for names.

White-space is Preserved in XML

HTML truncates multiple white-space characters to one single white-space:

HTML: Hello my name is Tove

Output: Hello my name is Tove.

With XML, the white-space in a document is not truncated.

Every XML document defines a single root element, whose opening tag must appear on the first line of XML code.

All other elements must be nested inside the root element. The root element of every XHTML document is html.

All XML Elements Must Have a Closing Tag

<element_name/> --- no content

In HTML, you will often see elements that don't have a closing tag:

This is a paragraph

This is another paragraph.

In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:

This is a paragraph

This is another paragraph

Opening and closing tags must be written with the same case:

<Message>This is incorrect</message>

<message>This is correct</message>

XML Elements Must be Properly Nested

In HTML, you might see improperly nested elements:

This text is bold and italic

In XML, all elements **must** be properly nested within each other:

This text is bold and italic

In the example above, "Properly nested" simply means that since the <i> element is opened inside the <b> element, it must be closed inside the <b> element.

XML Documents Must Have a Root Element

XML documents must contain one element that is the **parent** of all other elements. This element is called the **root** element.

<root>
<child>
<subchild>.....</subchild>
</child>
</root>

XML tags can have attributes, which are specified with name/value assignments. XML Attribute Values must be enclosed with single or double quotation marks. XML document that strictly adheres to these syntax rule is considered as well formed. An XML document that follows all of these rules is well formed

Example :

<?xml version = “1.0” encoding = “utf-8”?>
<ad>
<year>1960</year>
<make>Cessna</make>
<model>Centurian</moel>
<color>Yellow with white trim</color>
<location>
<city>Gulfport</city>
<state>Mississippi</state>
</location>
</ad>

None of this tag in the document is defined in XHTML-all are designed for the specific content of the document. When designing an XML document, the designer is often faced with the choice between adding a new attribute to an element or defining a nested element.

o In some cases there is no choices.

o In other cases, it may not matter whether an attribute or a nested element is used.

o Nested tags are used,

- when tags might need to grow in structural complexity in the future
- if the data is subdata of the parent element's content
- if the data has substructure of its own

o Attribute is used ,

- For identifying numbers/names of element
- If data in question is one value from a given possiblities
- The attribute is used if there is no substructure

```
<!-- A tag with one attribute -->
<patient name = "Maggie Dee Magpie">
...
</patient>
<!-- A tag with one nested tag -->
<patient>
<name> Maggie Dee Magpie </name>
...
</patient>
<!-- A tag with one nested tag, which contains
three nested tags -->
<patient>
<name>
<first> Maggie </first>
<middle> Dee </middle>
<last> Magpie </last>
</name>
...
</patient>
```

Here third one is a better choice because it provides easy access to all of the parts of data.

7.3 Document structure

XML document often uses two auxiliary files:

1. It specifies tag set and syntactic structural rules.

2. It contain the style sheet to describe how the content of the document to be printed.

XML documents (and HTML documents) are made up by the following building blocks: Elements, Tags, Attributes, Entities, PCDATA, and CDATA

Elements

Elements are the main building blocks of both XML and HTML documents.

Elements can contain text, other elements, or be empty.

Tags

Tags are used to markup elements.

A starting tag like <element_name> mark up the beginning of an element, and an ending tag like

</element_name> mark up the end of an element.

Examples:

A body element: <body>body text in between</body>.

A message element: <message>some message in between</message>

Attributes

Attributes provide extra information about elements.

Attributes are placed inside the start tag of an element. Attributes come in name/value pairs. The following "img" element has an additional information about a source file:

 The name of the element is "img". The name of the attribute is "src".

The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a "

/".

PCDATA

PCDATA means parsed character data.

Think of character data as the text found between the start tag and the end tag of an XML element.

PCDATA is text that will be parsed by a parser. Tags inside the text will be treated as markup and entities will be expanded.

CDATA

CDATA also means character data. CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

Most of you know the HTML entity reference: " " that is used to insert an extra space in an HTML document. Entities are expanded when a document is parsed by an XML parser.

_ An XML document often uses two auxiliary files:

- One to specify the structural syntactic rules (DTD / XML schema)
- One to provide a style specification (CSS /XSLT Style Sheets)

Entities

An XML document has a single root element, but often consists of one or more entities

An XML document consist of one or more entities that are logically related collection of information,

Entities range from a single special character to a book chapter

• An XML document has one document entity

* All other entities are referenced in the document entity

Reasons to break a document into multiple entities.

1. Good to define a Large documents as a smaller no. of parts easier to manage .

2. If the same data appears in more than one place in the document, defining it as a entity allows any no. of references to a single copy of data.

3. Many documents include information that cannot be represented as text, such as images. Such information units are usually stored as binary data. Binary entities can only be referenced in the document entities

Entity names:

• No length limitation

• Must begin with a letter, a dash, or a colon

• Can include letters, digits, periods, dashes, underscores, or colons

_ A reference to an entity has the form name with prepended ampersand and appended semicolon: &entity_name; Eg. &apple_image;

_ One common use of entities is for special characters that may be used fo markup delimiters to appear themselves in the document.

These are predefined (as in XHTML):

Character	Entity	Meaning
<	&lt	Less than
>	&gt	Greater than
&	&amp	Ampersand
“	&quot	Double quote
‘	&apos	Single quote

_ If several predefined entities must appear near each other in a document, it is better to avoid using entity references. Character data section can be used. The content of a character data section is not parsed by the XML parser, so it can include any tags.

_ The form of a character data section is as follows: <![CDATA[content]]> // no tags can be used since it is not parsed For example, instead of Start >>>> HERE <<<<

use

<![CDATA[Start >>>> HERE <<<<]]>

The opening keyword of a character data section is not just CDATA, it is in effect [CDATA[. There can be any spaces between [and C or between A and [.

_ Content of Character data section is not parsed by parser For example the content of the line <![CDATA[The form of a tag is <tag name>]]> is as follows The form of a tag is <tag name>

7.4 Document Type definitions

A DTD is a set of structural rules called declarations. These rules specify a set of elements, along with how and where they can appear in a document

• Purpose: provide a standard form for a collection of XML documents an define a markup language for them.

• DTD provide entity definition.

• With DTD, application development would be simpler.

• Not all XML documents have or need a DTD

External style sheets are used to impose a uniform style over a collection of documents.

When are DTDs used?

When same tag set definition are used by collection of documents , collection of users and documents must have a consistent and uniform structure.

A document can be tested against a DTD to determine weather it confirms to the rules the DTD describes.

Application programs that processes the data in the collection of XML documents can be written to assume the particular document form.

Without such structural restrictions, developing such applications would be difficult. If not impossible.

The DTD for a document can be internal (embedded in XML document) or external(separate file)- can be used with more than one document.

DTD with incorrect/inappropriate declaration can have wide-spread consequences.

DTD declarations have the form: <!keyword … >

There are four possible declaration keywords:

ELEMENT, ATTLIST, ENTITY, and NOTATION

1. Declaring Elements:

• Element declarations are similar to BNF(CFG)(used to define syntactic structure of Programming language) here DTD describes syntactic structure of particular set of doc so its rules are similar to BNF.

• An element declaration specifies the name of an element and its structure

• If the element is a leaf node of the document tree, its structure is in terms of characters

• If it is an internal node, its structure is a list of children elements (either leaf or internal nodes)

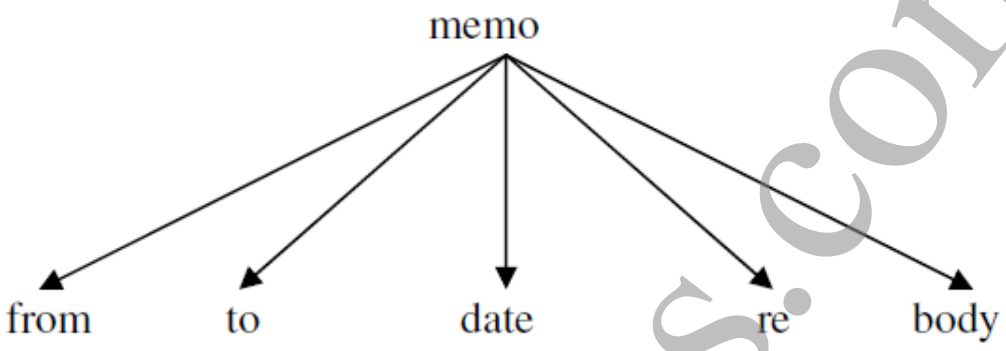
• General form:

<!ELEMENT element_name(list of child names)>

e.g.,

<!ELEMENT memo (from, to, date, re, body)>

This element structure can describe the document tree structure shown below.



• Child elements can have modifiers,

_ + -> One or more occurrences

_ * -> Zero or more occurrences

_ ? ->Zero or one occurrences

Ex: consider below DTD declaration

<!ELEMENT person (parent+, age, spouse?, sibling*)>

_ One or more parent elements

_ One age element

_ Possible a spouse element.

_ Zero or more sibling element.

• Leaf nodes specify data types of content of their parent nodes which are elements

1. PCDATA (parsable character data)

2. EMPTY (no content)

3. ANY (can have any content)

Example of a leaf declaration:

<!ELEMENT name (#PCDATA)>

2. **Declaring Attributes:**

• Attributes are declared separately from the element declarations

• General form:

<!ATTLIST element_name attribute_name attribute_type [default _value]>

More than one attribute

< !ATTLIST element_name attribute_name1 attribute_type default_value_1

attribute_name 2 attribute_type default_value_2

…>

_ Attribute type :There are ten different types, but we will consider only CDATA

_ Possible Default value for attributes:

Value - value ,which is used if none is specified

#Fixed value - value ,which every element have and can't be changed

Required - no default value is given ,every instance must specify a value

#Implied - no default value is given ,the value may or may not be specified

Example :

<!ATTLIST car doors CDATA "4">

<!ATTLIST car engine_type CDATA #REQUIRED>

<!ATTLIST car make CDATA #FIXED "Ford">

<!ATTLIST car price CDATA #IMPLIED>

<car doors = "2" engine_type = "V8">

...

</car>

Declaring Entities :

Two kinds:

• A general entity can be referenced anywhere in the content of an XML document

Ex: Predefined entities are all general entities.

• A parameter entity can be referenced only in DTD.

• General Form of entity declaration.

<!ENTITY [%] entity_name "entity_value">

% when present it specifies declaration parameter entity

Example :

<!ENTITY jfk "John Fitzgerald Kennedy">

_ A reference above declared entity: &jfk;

• If the entity value is longer than a line, define it in a separate file (an external text entity)

_ General Form of external entity declaration

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

SYSTEM specifies that the definition of the entity is in a different file.

• Example for parameter entity

<!ENTITY %pat “(USN, Name)”>

<!ELEMENT student %pat; >

4. 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">

5. Internal and External DTD's:

• Internal DTDs

<!DOCTYPE planes [

<!– The DTD for planes -->

]>

• External DTDs

<!DOCTYPE XML_doc_root_name SYSTEM

"DTD_file_name">

For examples,

<!DOCTYPE planes_for_sale SYSTEM

"planes.dtd">

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

<!-- planes.xml - A document that lists ads for used airplanes -->

<!DOCTYPE planes_for_sale SYSTEM "planes.dtd">

<planes_for_sale>

<ad>

<year> 1977 </year>

<make> &c; </make>

<model> Skyhawk </model>

<color> Light blue and white </color>

<description> New paint, nearly new interior,

685 hours SMOH, full IFR King avionics </description>

<price> 23,495 </price>

<seller phone = "555-222-3333"> Skyway Aircraft </seller>

<location>

<city> Rapid City, </city>

<state> South Dakota </state>

</location>

</ad>

</planes_for_sale>

7.5 Namespaces

XML provides benefits over bit formats and can now create well formed XML doc. But when applications become complex we need to combine elements from various doc types into one XML doc. This causes Pb?

Two documents will have elements with same name but with different meanings and semantics.

Namespaces – a means by which you can differentiate elements and attributes of different XML document types from each other when combining them together into other documents , or even when processing multiple documents simultaneously.

Why do we need Namespaces?

XML allows users to create their tags, naming collisions can occur For ex: element title

– <title>Resume of John</title>

– <title>Sir</title><Fname>John</Fname>

Namespaces provide a means for user to prevent naming collisions Element title

– <xhtml:title>Resume of John</xhtml:title>

– <person:title>Sir</person:title><Fname>John</Fname> xhtml and person --- two namespaces

Namespace is collection of elements and attribute names used in XML documents. It has a form of a URI

A Namespace for elements and attributes of the hierarchy rooted at particular element is declared as the value of the attribute xmlns. Namespace declaration for an element is

<element_name xmlns[:prefix] = URI>

• The square bracket indicates that what is within them is optional.

• prefix[optional] specify name to be attached to names in the declared namespace

Two reasons for prefix :

1. Shorthand for URI // URI is too long to be typed on every occurrence of every name from the namespace.

2. URI may includes characters that are illegal in XML

Usually the element for which a namespace is declared is usually the root of a document.

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

This declares the default namespace, from which names appear without prefixes.

As an example of a prefixed namespace declaration, consider

<birds xmlns:bd = http://www.audubon.org/names/species>

Within the birds element, including all of its children elements, the names from the namespace must be prefixed with bd, as in the following.

<bd:lark>

If an element has more than one namespace declaration, then

<birds xmlns:bd = “http://www.audubon.org/names/species”>

xmlns : html = “http://www.w3.org/1999/xhtml”>

Here we have added the standard XHTML namespace to the birds element. One namespace declaration in an element can be used to declare a default namespace. This can be done by not including the prefix in the declaration. The names from the default by omitting the prefix.

Consider the example in which two namespaces are declared.

The first is declared to be the default namespaces.

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> 75689 </population>

<capital>

<cap:name> Pierre </cap:name>

<cap:population>12429</cap:population>

</capital>

</state>

</states>

Each state element has name and population elements for both namespaces.

7.6 XML schemas

A schema is any type of model document that defines the structure of something, such as databases structure or documents. Here something is XML doc. Actually DTDs are a type of schema.

An XML schema is an XMl document so it can be parsed with an XML parser.

The term XML scheme is used to refer to specify W3C XML schema technology.

W3C XML Schemas like DTD allow you to describe the structure for an XML doc.

DTDs have several disadvantages

• Syntax is different from XML - cannot be parsed with an XML parser

• It is confusing to deal with two different syntactic forms

• DTDs do not allow restriction on the form of data that can be content of element ex: <quantity>5</quantity> and <quantity>5</quantity> are valid DTD can only specifies that could be anything. Eg time No datatype for integers all are treated as texts.

XML Schemas is one of the alternatives to DTD

• It is XML document, so it can be parsed with XML parser

• It also provides far more control over data types than do DTDs

• User can define new types with constraints on existing data types

1. Schema Fundamentals:

• Schema are related idea of class and an object in an OOP language

Schema class definition

XML document confirming to schema structure Object

• Schemas have two primary purposes

Specify the structure of its instance XML documents

pecify the data type of every element & attribute of its instance XML documents

2. Defining a schema:

Schemas are written from a namespace(schema of schemas):

http://www.w3.org/2001/XMLSchema element, schema, sequence and string are some names from this namespace

Every XML schema has a single root, schema.

• The schema element must specify the namespace for the schema of schemas from which the schema's elements and its attributes will be drawn.

• It often specifies a prefix that will be used for the names in the schema. This name space specs appears as

xmlns:xsd = http://www.w3.org/2001/XMLSchema

Every XML schema itself defines a tag set like DTD, which must be named with the targetNamespace attribute of schema element. The target namespace is specified by assigining a name space to the target namespace attribute as the following: targetNamespace = http://cs.uccs.edu/planeSchema

Every top-level element places its name in the target namespace If we want to include nested elements, we must set the elementFormDefault attribute to qualified. elementFormDefault = qualified.

The default namespace which is source of the unprefixed names in the schema is given with another xmlns specification xmlns = "http://cs.uccs.edu/planeSchema

A complete example of a schema element:

<xsd:schema

<!-- Namespace for the schema itself -->

xmlns:xsd = http://www.w3.org/2001/XMLSchema

<!-- Namespace where elements defined here will be placed -->

targetNamespace = “http://cs.uccs.edu/planeSchema”

<!-- Default namespace for this document -->

xmlns = “http://cs.uccs.edu/planeSchema”

<!-- Specify non-top-level elements to be in the target namespace-->

elementFormDefault = "qualified” >

Defining a schema instance:

• An instance of schema must specify the namespaces it uses

• These are given as attribute assignments in the tag for its root element

1. Define the default namespace

<planes

xmlns = http://cs.uccs.edu/planesScema

…>

2. It is root element of an instance document is for the schemaLocation attribute.

Specify the standard namespace for instances (XMLSchema-instance) xmlns:xsi =“http://www.w3.org/2001/XMLSchema-instance"

3. Specify location where the default namespace is defined, using the schemaLocation attribute, which is assigned two values namespace and filename.

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

4. Schema Data types: Two categories of data types

1.Simple (strings only, no attributes and no nested elements)

2. Complex (can have attributes and nested elements)

• XML Schema defines 44 data types

• Primitive: string, Boolean, float, …

• Derived: byte, decimal, positiveInteger, …

• User-defined (derived) data types – specify constraints on an existing type (then called as base type)

• Constraints are given in terms of facets of the base type

Ex: interget data type has *8 facets :totalDigits, maxInclusive….

Both simple and complex types can be either named or anonymous

DTDs define global elements (context of reference is irrelevant). But context of reference is essential in XML schema

Data declarations in an XML schema can be

1. Local ,which appears inside an element that is a child of schema
2. Global, which appears as a child of schema

5. Defining a simple type:

• Use the element tag and set the name and type attributes

<xsd:element name = "bird" type = "xsd:string"/>

The instance could be :

<bird> Yellow-bellied sap sucker </bird>

• An element can be given default value using default attribute

<xsd:element name = "bird" type = "xsd:string" default="Eagle" />

• An element can have constant value, using fixed attribute

<xsd:element name = "bird" type = "xsd:string" fixed="Eagle" />

Declaring User-Defined Types:

• User-Define type is described in a simpleType element, using facets

• facets must be specified in the content of restriction element

• facets values are specified with the value attribute

For example, the following declares a user-defined type , firstName

<xsd:simpleType name = "firstName" >

<xsd:restriction base = "xsd:string" >

<xsd:maxLength value = "20" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name = “phoneNumber" >

<xsd:restriction base = "xsd:decimal" >

<xsd:precision value = “10" />

</xsd:restriction>

</xsd:simpleType>

6. Declaring Complex Types:

• There are several categories of complex types, but we discuss just one, element-only elements

• Element-only elements are defined with the complex Type element

• Use the sequence tag for nested elements that must be in a particular order

• Use the all tag if the order is not important

• Nested elements can include attributes that give the allowed number of occurrences (minOccurs, maxOccurs, unbounded)

• For ex:

<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:string" />

</xsd:sequence>

</xsd:complexType>

7. Validating Instances of Schemas:

• An XML schema provides a definition of a category of XML documents.

• However, developing a schema is of limited value unless there is some mechanical way to determine whether a given XML instance document confirms to the schema.

• Several XML schema validation tools are available eg. xsv(XML schema validator) This can be used to validate online.

• Output of xsv is an XML document. When run from command line output appears without being formated.

Output of xsv when run on planes.xml

<?XML version = '1.0' encoding = 'utf-8?>

<xsv docElt = '{ http://cs.uccs.edu/planesSchema} planes'

instanceAccessed = 'true'

instanceErrors = '0'

schemaErrors = '0'

schemaLocs = 'http://cs.uccs.edu/planesSchema->planes.xsd'

Target = 'file: /c:/wbook2/xml/planes.xml'

Validation = 'strict'

Version = 'XSV 1.197/1.101 of 2001/07/07 12:01:19'

Xmlns='http:// www.w3.org/2000/05.xsv'>

<importAttempt URI ='file:/c:wbook2/xml/planes.xsd'

namespace = 'http://cs.uccs.edu/planesSchema'

outcome = 'success' />

</xsv>

If schema is not in the correct format, the validator will report that it could not find the specified schema.

7.7 Displaying RAW XML Documents

An XML enabled browser or any other system that can deal with XML documents cannot possibly know how to format the tags defined in the doc.

Without a style sheet that defines presentation styles for the doc tags the XML doc can not be displayed in a formatted manner.

Some browsers like FX2 have default style sheets that are used when style sheets are not defined.

Eg of planes.xml document.

7.8 Displaying XML Documents with CSS

Style sheet information can be provided to the browser for an xml document in two ways.

• First, a CSS file that has style information for the elements in the XML document can be developed.

• Second the XSLT style sheet technology can be used..

Using CSS is effective, XSLT provides far more power over the appearance of the documents display.

A CSS style sheet for an XML document is just a list of its tags and associated styles

The connection of an XML document and its style sheet is made through an xmlstylesheet processing instruction

Display– used to specify whether an element is to be displayed inline or in a separate block.

<?xml-stylesheet type = "text/css“ href = “planes.css"?>

For example: planes.css

<!-- planes.css - a style sheet for the planes.xml document -->

ad { display: block; margin-top: 15px; color: blue;}

year, make, model { color: red; font-size: 16pt;}

```
color {display: block; margin-left: 20px; font-size: 12pt;}
description {display: block; margin-left: 20px; font-size: 12pt;}
seller { display: block; margin-left: 15px; font-size: 14pt;}
location {display: block; margin-left: 40px; }
city {font-size: 12pt;}
state {font-size: 12pt;}
<?xml version = "1.0" encoding = "utf-8"?>
<!-- planes.xml - A document that lists ads for used airplanes -->
<planes_for_sale>
<ad>
<year> 1977 </year>
<make> Cessana </make>
<model> Skyhawk </model>
<color> Light blue and white </color>
<description> New interior
</description>
<seller phone = "555-222-3333">
Skyway Aircraft </seller>
<location>
<city> Rapid City, </city>
<state> South Dakota </state>
</location>
</ad>
</planes_for_sale>
```

With planes.css the display of planes.xml as following: 1977 Cessana Skyhawk Light blue and white New interior Skyway Aircraft Rapid City, South Dakota

Web Services:

The ultimate goal of Web services: Allow different software in different places, written in different languages and resident on different platforms, to connect and interoperate

The Web began as provider of markup documents, served through the HTTP methods, GET and POST

A Web service is closely related to an information service

- The server provides services, through server- resident software
- The same Web server can provide both documents and services

The original Web services were provided via Remote Procedure Call (RPC), through two technologies, DCOM and CORBA. DCOM and CORBA use different protocols, which defeats the goal of universal component interoperability

There are three roles required to provide and use Web services:

1. Service providers
2. Service requestors
3. A service registry

• Service providers

- Must develop & deploy software that provide service

- Service Description --> Web Serviced Definition Language (WSDL)

*Used to describe available services, as well as of message protocols for their use their use

* Such descriptions reside on the Web server

• Service requestors

- Uses WSDL to query a query a web registry

• Service registry

- Created using Universal Description, Discovery, and Integration Service (UDDI)

* UDDI also provides

- Create service registry

- Provides method to query a Web service registry

• Standard Object Access Protocol (SOAP)

- An XML-based specification that defines the forms of messages and RPCs

- The root element of SOAP is envelope

- Envelope contains SOAP messages – description of web services

- Supports the exchange of information among distributed systems

UNIT - 8

PERL, CGI PROGRAMMING

8.1 Origins and uses of Perl

8.2 Scalars and their operations

8.3 Assignment statements and simple input and output

8.4 Control statements

8.5 Fundamentals of arrays

8.6 Hashes

8.7 References

8.8 Functions

8.9 Pattern matching

8.10 File input and output

8.11 Examples

8.12 The Common Gateway Interface

8.13 CGI linkage

8.14 Query string format

8.15 CGI.pm module

8.16 A survey example

8.17 Cookies

UNIT - 8

PERL, CGI PROGRAMMING

8.1 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

8.2 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

8.3 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.

8.4 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"
```

8.5 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."

8.7 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.
";

}

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 . "
";

}

```
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

8.8 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.

8.9 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;
}
```

8.10 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.

8.11 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

<!-- 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

</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";

8.12 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.

8.13 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;

8.14 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).