Internet & Intranet Applications

Binnur Kurt, PhD
bkurt@itu.edu.tr

Istanbul Technical University
Computer Engineering Department

Grading Scheme

- 5 Homework (30%)
- A midterm exam (30%)
  November 16th 2007
- A final exam (40%)
- Academic dishonesty including but not limited to cheating, plagiarism, collaboration is unacceptable and subject to disciplinary actions. Any student found guilty will have grade -100. Assignments are due in class on the due date. Late assignments will NEVER be accepted.

About the Lecturer

- BSc, MSc, PhD

Areas of Interest

- Digital Image/Video Analysis
- Real-Time Computer Vision Systems
- Multimedia: Indexing and Retrieval
- Software Engineering
- OO Analysis and Design
- Web Services

Welcome to the Course

- Important Course Information
  - 18:30-21:30, Friday
  - Course Web Page
    http://www.cs.itu.edu.tr/~kurt/Courses/bte550

Tell me and I forget.
Show me and I remember.
Let me do and I understand.
—Chinese Proverb

Course Outline

1. Networking Fundamentals
2. Introduction to Web Technologies
3. XML
4. XML using DTDs
5. HTML
6. XHTML
7. XPath and XSLT
8. CSS
10. Dynamic HTML: JavaScript
11. Web Applications: Forms, JSP, PHP
Networking Fundamentals

Objectives

► Define basic networking terms
► Describe some commonly used network applications
► Describe the main purposes and functions of computer networking
► Describe the history and purposes of the OSI reference model

Objectives (Cont.)

► Discuss the functions of each of the seven layers of the OSI reference model and provide examples of each
► Describe the basic process of communication between the layers of the OSI reference model
► Describe the functions of the TCP/protocol stack and provide examples of each layer’s function
► Compare the TCP/IP protocol stack to the OSI reference model

Basic Networking Terminology

► NIC
► Media
► Protocol
► NOS
► Connectivity devices
► LAN
► MAN
► WAN
► Physical topology
► Logical topology

How to Use the Icons

Demonstration
Discussion
Reference
Exercise

WWW Request-Response

Web Browser
Web Server

Copyright © 2005, Binnur Kurt
Sending E-Mail

E-mail Server

Client A

E-mail Server

Client B

The OSI Reference Model

7. Application
6. Presentation
5. Session
4. Transport
3. Network
2. Data Link
1. Physical

Why a Layered Network Model?

► Reduces complexity
► Standardizes interfaces
► Facilitates modular engineering
► Ensures interoperable technology
► Accelerates evolution
► Simplifies teaching and learning

Defining Components of the Network

Home Office

Mobile Users

Internet

Branch Office

Main Office

The Seven Layers of the OSI Model

7. Application
6. Presentation
5. Session
4. Transport
3. Network
2. Data Link
1. Physical

Network Processes to Applications
► Provides network services to application processes (such as electronic mail, file transfer, and terminal emulation)
The Seven Layers of the OSI Model

1. Application
2. Presentation
3. Session
4. Transport
5. Network
6. Data Link
7. Physical

Network Process to Applications
- Data Representation
  - Ensures data is readable by receiving system
  - Formats data
  - Structures data
  - Negotiates data transfer syntax for application layer

Interhost Communication
- Establishes, manages, and terminates sessions between applications

End-to-End Connections
- Handles transportation issues between hosts
- Ensures data transport reliability
- Establishes, maintains, and terminates virtual circuits
- Provides reliability through fault detection and recovery information flow control

Data Delivery
- Provides connectivity and path selection between two host systems
- Routes data packets
- Selects best path to deliver data

Access to Media
- Defines how data is formatted for transmission and how access to the network is controlled

Binary Transmission
- Defines the electrical, mechanical, procedural, and functional specifications for activating, maintaining, and deactivating the physical link.
Data Encapsulation

Sender

User Data

User Data

User Data

User Data

User Data

L7 HDR

L6 HDR

L5 HDR

L4 HDR

L3 HDR

L2 HDR

L1 HDR

BITE550 – Internet and Intranet Applications

TCP/IP Model

Application

Presentation

Internet

Network Access

BITE550 – Internet and Intranet Applications

Data De-Encapsulation

Receiver

User Data

User Data

User Data

User Data

User Data

L7 HDR

L6 HDR

L5 HDR

L4 HDR

L3 HDR

L2 HDR

L1 HDR

BITE550 – Internet and Intranet Applications

TCP/IP Protocol Stack vs. OSI Model

TCP/IP Protocol Stack

OSI Model

TCP/IP Protocol Stack

OSI Model

Application

Protocols

Networks

Data Flow Layers

Application

Presentation

Session

Transport

Network

Data Link

Physical

BITE550 – Internet and Intranet Applications

Peer-to-Peer Communication

Sender

Application

Presentation

Session

Transport

Network

Data Link

Physical

Receiver

Application

Presentation

Session

Transport

Network

Data Link

Physical

BITE550 – Internet and Intranet Applications

Summary

► There are a number of basic computer network terms, including NIC, media, protocol, physical topology, and logical topology, that are fundamental to an understanding of networks.

► Network applications are software programs that run between computers that are connected together on a network.

► Some common network applications include HTTP, POP3, FTP, Telnet, and SNMP.
Summary (Cont.)

► There are many different ways in which a computer network can be constructed to meet the requirements of an organization, but user components are generally grouped into the categories of main office, remote locations, and mobile users.

► The ISO created and released the OSI reference model in 1984 to provide vendors with a set of standards to ensure greater compatibility and interoperability between various types of network technologies.

► The OSI reference model reduces complexity, standardizes interfaces, facilitates modular engineering, ensures interoperable technology, accelerates evolution, and simplifies teaching and learning.

► Each layer of the OSI model has a set of unique functions. The seven layers of the OSI model are the application, presentation, session, transport, network, data-link, and physical layers.

► Encapsulation is the process in which data is wrapped in a particular protocol header before network transit.

► The TCP/IP protocol stack has four layers: the application layer, transport layer, internet layer, and network access layer.

► There are both similarities and differences between the TCP/IP protocol stack and the OSI reference model.

Summary (Cont.)

► The ISO created and released the OSI reference model in 1984 to provide vendors with a set of standards to ensure greater compatibility and interoperability between various types of network technologies.

► Each layer of the OSI model has a set of unique functions. The seven layers of the OSI model are the application, presentation, session, transport, network, data-link, and physical layers.

► Encapsulation is the process in which data is wrapped in a particular protocol header before network transit.

► The TCP/IP protocol stack has four layers: the application layer, transport layer, internet layer, and network access layer.

► There are both similarities and differences between the TCP/IP protocol stack and the OSI reference model.
Networking devices are products used to connect networks. Hubs, switches, and routers interconnect devices within LANs, MANs, and WANs. Networking devices function at different layers of the OSI model, primarily Layers 1, 2, and 3. Repeaters reshape, amplify, and retiming signals before sending them along the network.

The term "hub" (also called a multiport repeater) is used instead of "repeater" when referring to the device that serves as a connection point in a network. Hubs work Layer 1 only and make no filtering decisions. Layer 2 LAN switches work at Layer 2, and they make limited MAC hardware address decisions. Routers can make decisions as to the best path for delivery of data on the network.

Working at Layers 2, 3, and 4, multilayer switches enable implementation of Layer 3 QoS and security functionality and perform many of the same functions as routers do, but in hardware. Voice gateways, DSLAMs, and optical devices are newer types of network connectivity devices. Firewalls and AAA servers provide security to the network.
Physical Topologies
- Bus Topology
- Ring Topology
- Star Topology
- Extended Star Topology
- Mesh Topology

Logical Topologies
- Main Switch
- Main Server
- Workgroup Switch
- Bridge
- Repeater

Extended-Star Topology

Bus Topology

Ring Topology
Dual-Ring Topology

Summary

► A physical topology describes the plan for wiring the physical devices, while a logical topology describes how information flows through a network.
► In a physical bus topology, a single cable connects all the devices.
► The most commonly used architecture in Ethernet LANs is the physical star topology, with each host in the network connected to the central device with its own cable.

Full-Mesh Topology

Summary (Cont.)

► When a star network is expanded to include additional networking devices that are connected to the main networking device, it is called an extended-star topology.
► In a ring topology, all the hosts are connected in the form of a ring or circle.
► A full-mesh topology connects all devices to each other, while in a partial-mesh topology, at least one device has multiple connections to others.

Partial-Mesh Topology

Communications Protocol

► TCP
TCP/IP Protocol Stack

TCP Characteristics
- Connection-Oriented Protocol
- Full-Duplex Operation
- Error Checking
- Sequencing
- Acknowledgments
- Flow Control
- Packet Recovery

UDP Characteristics
- Minimal Service
- Unreliable
- Not-Guaranteed
- Direct Access to Datagrams

Application Layer Overview
- File Transfer
  - TFTP
  - FTP
- E-Mail
  - SMTP
- Remote Login
  - Telnet
  - rlogin
- Network Management
  - SNMP
- Name Management
  - DNS
*Used by the router

Summary
- Protocols define a standard set of rules for communicating between devices.
- TCP/IP is a suite of protocols arranged as a stack.
- TCP is a connection-oriented protocol that provides flow control and reliable data delivery services.
- UDP provides minimal, non-guaranteed, transport services.
- The transport layer supports multiple application protocols.

Network and Host Addresses

TCP/IP Protocol Stack

Application
- Transport
- Network
- Data Link
- Physical

OSI Reference Model

TCP/IP Protocol Stack

Network and Host Addresses

Networking Fundamentals

BTE550 – Internet and Intranet Applications
Dotted-Decimal Notation

- An IP address is a 32-bit binary number: 10101000000100001000000000100011
- The 32-bit binary number can be divided into four octets: 10101100 00010000 10000000 00010001
- Each octet (or byte) can be represented in decimal: 172 16 128 17
- The address can be written in dotted-decimal notation: 172.16.128.17

IP Address Classes

- Class A: 0 7 24
- Class B: 1 0 14 16
- Class C: 1 1 0 21 8
- Class D:* 1 1 1 1
- Class E:** 1 1 1 1

IP Address Class Components

- **Class E addresses are reserved for research use only.
- *Class D addresses are used for multicast groups. There is no need to allocate octets or bits to separate network and host addresses.

Network Addresses

- Network Address (host bits = all zeros)
- Broadcast Address (host bits = all ones)

Broadcast Address

- 127 (01111111) is a Class A address reserved for loopback testing and cannot be assigned to a network.
Private IP Addresses

<table>
<thead>
<tr>
<th>Class</th>
<th>RFC 1918 Internal Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10.0.0.0 to 10.255.255.255</td>
</tr>
<tr>
<td>B</td>
<td>172.16.0.0 to 172.31.255.255</td>
</tr>
<tr>
<td>C</td>
<td>192.168.0.0 to 192.168.255.255</td>
</tr>
</tbody>
</table>

The Internet

In the early days of computer networking, local area networks (LANs) were created to share resources among members of a particular institution. These networks were constrained to short distances.

The Internet is a Network of Networks

IPv4 Address Allocation

- With Class A and B addresses virtually exhausted, Class C addresses (12.5 percent of the total space) are left to assign to new networks.

Introduction to Web Application Technologies

Client-Server Architecture

Hypertext Transfer Protocol

- The Hypertext Transfer Protocol (HTTP) is similar to FTP because it is a protocol to transfer files from the server to the client.
- HTTP was created in conjunction with the related Hypertext Markup Language (HTML) standard.
- There is one fundamental difference between FTP and HTTP: HTTP supports only one request per connection.
- This means that the client connects to the server to retrieve one file and then disconnects. This mechanism allows more users to connect to a given server over a period of time.
HTML
- HTML is a document display language that allows users to link from one document to another.
- HTML also permits images and other media objects to be embedded in an HTML document. The media objects are stored in files on a server.
- HTTP also retrieves these files. HTTP can therefore be used to transmit any file that conforms to the Multipurpose Internet Mail Extensions (MIME) specification.

HTTP Client-Server Architecture
- For every exchange over the Web using HTTP, there is a request and a response.

Web Browsers and Web Servers
- To view an HTML document with rich media content, a graphical user interface (GUI) was built on top of the client-side HTTP. This GUI is called a Web browser.
- The server-side HTTP component is called a Web server.
- Several companies have developed Web browsers and Web servers; some have developed both. The first Web server was a process called httpd; the first widely used browser was Mosaic, created by National Center for Supercomputing Applications (NCSA).

Uniform Resource Locator
- A URL is a canonical name that locates a specific resource on the Internet:
  protocol://host:port/path/file
- Example:
- The path element includes the complete directory structure path to find the file. The port number is used to identify the TCP port that is used by the protocol on the server. If the port number is the standard port for the given protocol, then that number can be ignored in the URL.
- For example, port 80 is the default HTTP port.

Web Applications
- Very early in the development of HTML, the designers created a mechanism to permit a user to invoke a program on the Web server.
- This was called the Common Gateway Interface (CGI). When a Web site includes CGI processing, this is called a Web application.
**CGI Programs on the Web Server**

- Usually, the browser needs to send data to the CGI program on the server.
- The CGI specification defines how the data is packaged and sent in the HTTP request to the server. This data is usually typed into the Web browser in an HTML form.
- The URL determines which CGI program to execute. This might be a script or an executable file.
- The CGI program parses the CGI data in the request, processes the data, and generates a response (usually an HTML page). The CGI response is sent back to the Web server, which wraps the response in an HTTP response. The HTTP response is sent back to the Web browser.

**Execution of CGI Programs**

- At runtime, a CGI program is launched by the Web server as a separate operating system (OS) shell.
- The shell includes an OS environment and process to execute the code of the CGI program, which resides within the server’s file system.

**3 Overview of XML**

- Define XML
- Compare and contrast HTML and XML
- Identify characteristics of XML documents
What is XML?
► the eXtensible Markup Language
► W3C-endorsed standard for document markup
► A generic syntax used to mark up data with simple, human-readable tags
► Provides a standard format for computer documents
► Flexible enough to be customized for different domains as diverse as web sites, electronic data interchange, vector graphics, real-estate listings, object serialization, remote procedure calls, voice-mail systems,...

What is not XML?
► XML is not a programming language
  – There’s no such thing as an XML compiler that reads XML files and produces executable code
► XML is not a network transport protocol
► XML is not a database
  – You’re not going to replace an Oracle or MySQL server with XML
  – A database can contain XML data, but the database itself is not an XML document

HTML Document (Good for Formatting)

<?xml version = "1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE employees SYSTEM "employees.dtd">
<employees>
  <company-name>Sun Microsystems, Inc.</company-name>
  <employee>
    <id>3005790</id>
    <name><first>Adrian</first><last>Gonzalez</last></name>
    <email>adrian@sun.com</email>
    <address>901 San Antonio Road, …</address>
    <phonenumber>408-5551212</phonenumber>
  </employee>
</employees>

What do we need for Web Services & B2B?
► Portable Data
► Portable Code
Service Driven Approach

Java and XML: Symbiotic Relationship
- It’s a “Match made in Heaven”
  - Java enables Portable Code
  - XML enables Portable Data
- XML tools and programs are mostly written in the Java programming language
- Better API support for Java platform than any other language
- Two great tastes that taste great together

XML: The Missing Link For Web Applications
- Portable code
- Portable data
- Ubiquitous communication

Two Viewpoints of XML
- Presentation Oriented Publishing (POP)
  - Useful for Browsers and Editors
  - Usually used for data that will be consumed by Humans
- Message Oriented Middleware (MOM)
  - Useful for Machine-to-Machine data exchange
  - Business-to-Business communication an excellent example
Standardization Activities

► XML Standards
  – Through Standard organizations
  – W3C, IETF, OASIS, UN/CEFACT

XML

► The eXtensible Markup Language (XML) is the universal format for structured documents and data on the Web
► XML is a text-based markup language.
► As with HTML, you identify data using tags (identifiers enclosed in angle brackets, like this: <...>).
► Collectively, the tags are known as "markup".
► But unlike HTML, XML tags tell you what the data means, rather than how to display it.

W3C

► World Wide Web Consortium (W3C) creates Web standards.
► W3C’s mission is to lead the Web to its full potential, which it does by developing technologies (specifications, guidelines, software, and tools) that will create a forum for information, commerce, inspiration, independent thought, and collective understanding.
► W3C defines the Web as the universe of network-accessible information
► W3C languages RDF, XML, and digital signatures are the building blocks of the Semantic Web.

How XML Works

<?xml version="1.0"?>
<invoice>
  <orderDate>2005-01-01</orderDate>
  <shipDate>2005-01-05</shipDate>
  <billingAddress>
    <name>Paul Biron</name>
    <street>123 IBM Avenue</street>
    <city>Hawthorne</city>
    <state>NY</state>
    <zip>10532</zip>
  </billingAddress>
  <voice>555-1234</voice>
  <fax>555-4321</fax>
</invoice>

This document is text and might well be stored in a text file. You can edit this file with any standard text editor

XML Parser

► An XML parser is responsible for dividing the document into individual elements, attributes, and other pieces.
► It passes the contents of the XML document to an application piece by piece.
► If at any point the parser detects a violation of the well-formedness rules of XML, it then reports the error to the application and stops parsing.
Individual XML applications normally dictate more precise rules about exactly which elements and attributes are allowed where.

Some XML parsers compare the document to its schema as they read it to see if the document satisfies the constraints specified there.

Such a parser is called a validating parser.

Checking a document against a schema is called validation.

Not all parsers are validating parsers. Some merely check for well-formedness.

XML is a descendant of SGML, the Standard Generalized Markup Language.

SGML was invented by Charles F. Goldfarb, Ed Mosher, and Ray Lorie at IBM in the 1970s.


It is a semantic and structural markup language for text documents.

Achieved some success in the U.S. military and government, in the aerospace sector.

SGML's biggest success was HTML, which is an SGML application.

The problem: SGML is complicated—very, very complicated.

It is so complex that almost no software has ever implemented it fully.

In 1996, J. Bosak, T. Bray, C. M. Sperberg, J. Clark, and several others began work on a "lite" version of SGML.

The result, in February of 1998, was XML 1.0.

The next standard out of the gate was Namespaces in XML.

Next was the Extensible Stylesheet Language (XSL).

Development of extensions to the core XML specification continues.

- XML Namespaces
- XML DTDs, XML Schema
- XSL (Extensible Style Sheet Language)
- XPath (= XSLT/ XPointer), XLink
- XQL (XML Query Language)
- eXcelon

XML is a family of standards comprising XML, DTD, XSL, XPointer, XLink, XQuery, XPointer, XSLT, XML Schema, XML DTDs, and XML Namespaces.
XML Fundamentals

XML Documents and XML Files

- An XML document contains text, never binary data.
- It can be opened with any program that knows how to read a text file.

```xml
<person>
  Alan Turing
</person>
```

A very simple yet complete XML document.

Your operating system may or may not like these names,
but an XML parser won’t care.

Elements, Tags, and Character Data

- Example is composed of a single element named person.
- The element is delimited by the start-tag `<person>` and the end-tag `</person>`.
- Everything between the start-tag and the end-tag of the element is called the element’s content.
- The whitespace is part of the content, though many applications will choose to ignore it.
- The string "Alan Turing" and its surrounding whitespace are character data.

XML Characteristics

- Elements
- Text
- Attributes

```xml
<purchaseOrder>
  <description>Battery</description>
  <quantity>9</quantity>
  <price>60</price>
</purchaseOrder>
```

```xml
<shoeOrder id="4040458">
  <color>Brown</color>
  <size>9</size>
  <width>AA</width>
</shoeOrder>
```
An XML Document

```
<?xml version="1.0"?>
<!DOCTYPE order SYSTEM "order.dtd">
<order>
  <book isbn="0-201-34285-5">
    <title>The XML Companion</title>
    <author>Neil Bradley</author>
    <publisher>Addison-Wesley</publisher>
  </book>
<!-- This is a comment -->
</order>
```

Processing Instruction

```
Processing Instruction
```

Document Type Definition (DTD)

```
Document Type Definition (DTD)
```

Root Element

```
Root Element
```

Attribute

```
Attribute
```

Element

```
Element
```

Comment

```
Comment
```

XML Elements

- Basic components of XML documents
- Elements must start with a letter, underscore or colon
- Encapsulate element content, usually composed of:
  - Other elements
  - Character data
  - Entity references
- Delimited using tags
- All elements must have a start-tag and an end-tag
- Elements can optionally have attributes
- Empty elements can use an abbreviated element form

XML Namespaces

- XML Namespaces allow a prefix to be associated with an element to avoid name collisions
- XML Namespaces are a W3C specification
- A unique URI must be used with a prefix to denote elements in this namespace from other namespaces
- The URI is only for distinguishing prefixes, it is not actually resolved
- Namespaces use the reserve word xmlns
  ```
  <CC:LunchMenu xmlns:Camp="http://catering.com/CC">
    . . .
  </CC:LunchMenu>
  ```

Case Sensitivity

- XML, unlike HTML, is case sensitive
- `<person>` is not the same as `<PERSON>`
- If you open an element with a `<person>` tag, you can't close it with a `<PERSON>` tag

XML Trees

- Every XML document has one element that does not have a parent: root element
- `invoice`
  ```
  orderDate=2005-01-01
  shipDate=2005-01-05
  billingAddress:
    name=Paul Biron
    street=123 IBM Avenue
    city=Hawthorne
    state=NY
    zip=10532
  billingAddress:
    voice=555-1234
    fax=555-4321
  ```
- Root Element is `invoice`

XML Trees (Con't)

- The URI is only for distinguishing prefixes, it is not actually resolved
Attributes
► Elements can contain attributes to provide information about the element
► Attributes are not considered part of an element’s content
► Attributes are not part of the presentation to an end user, though they may be used to affect the presentation
► An attribute is a name-value pair attached to the element’s start-tag
► Names are separated from values by an equals sign and optional whitespace
► Values are enclosed in single or double quotation marks

Attributes (Con’t)

Attributes (Con’t)

<person born="1912-06-23" died="1954-06-07">
  Alan Turing
</person>

or

<person born='1912-06-23' died='1954-06-07'>
  Alan Turing
</person>

Attributes (Con’t)

When and whether one should use child elements or attributes to hold information? This is a subject of heated debate

XML Names (Con’t)

XML Names
► XML names may only start with letters, ideograms, and the underscore character.
► They may not start with a number, hyphen, or period.
► There is no limit to the length of an element or other XML name.
► Thus these are all well-formed elements:
  – <Drivers_License_Number>98 NY 32</Drivers_License_Number>
  – <month-day-year>7/23/2001</month-day-year>
  – <first_name>Alan</first_name>
  – <_4-lane>I-610</_4-lane>
  – <téléphone>011 33 91 55 27 55 27</téléphone>

XML Names
► Element and other XML names may contain essentially any alphanumeric character.
► This includes the standard English letters A through Z and a through z as well as the digits 0 through 9.
► XML names may also include non-English letters, numbers, and ideograms such as ö, ç, Ω
► They may also include these three punctuation characters:
  – _ the underscore
  – - the hyphen
  – . the period

White Space
► XML defines white space as any of these 4 characters
  – Horizontal tab
  – Line feed
  – Carriage return
  – Space
► An XML parser must pass all white space contained within content to the application
► An XML parser may remove white space in element tags and attribute values
► All end of line characters are converted to line feed characters by parsers

White Space

White Space

White Space

White Space

White Space

XML Names

XML Names

XML Names

XML Names

White Space

XML Names (Con’t)
XML Names (Con’t)

- `<permittedNames>`
- `<name/>`
- `<xsl:copy-of>`
- `<A_long_element_name/>`
- `<A.name.separated.with.full.stops/>`
- `<a123323123-231/>`

- `<forbiddenNames>`
- `<A;name/>`
- `<last@name>`
- `<@#$%^()+=/>`
- `<A*2/>`
- `<1ex/>`

Character References

- Character references represent displayable characters that cannot otherwise be displayed
- Character references are either decimal or hexadecimal numbers
  - Decimals are preceded by `&#`;
  - Hexadecimals are preceded by `&#x`
- All character references end with a semicolon
  - `&#169` or `&#xA9` will display as ©

Entity References

- The character data inside an element may not contain a raw unescaped opening angle bracket (`<`).
- This character is always interpreted as beginning a tag
- If you need to use this character in your text, you can escape it using the `&lt;` entity reference with the actual `<` character
- `&lt;publisher&gt;O'Reilly &amp; Associates&lt;/publisher&gt;`

Entity References (Con’t)

- `&lt;` - The less-than sign; a.k.a. the opening angle bracket (`<`)
- `&amp;` - The ampersand (`&`)
- `&gt;` - The greater-than sign; a.k.a. the closing angle bracket (`>`)
- `&quot;` - The straight, double quotation marks (`"`)
- `&apos;` - The apostrophe; a.k.a. the straight single quote (`'`)

CDATA Sections

- When an XML document includes samples of XML or HTML source code, the `<` and `&` characters in those samples must be encoded as `&lt;` and `&amp;`.
- The more sections of literal code a document includes and the longer they are, the more tedious this encoding becomes
- Instead you can enclose each sample of literal code in a CDATA section. CDATA sections exist for the convenience of human authors, not for programs.
- An XML parser will not attempt to process any data in a CDATA section

CDATA Sections: Example

```xml
<p>You can use a default <code>xmlns</code> attribute to avoid having to add the svg prefix to all your elements:
</p>

<![CDATA[
  <svg xmlns="http://www.w3.org/2000/svg" width="12cm" height="10cm">
    <ellipse rx="110" ry="130" />
    <rect x="4cm" y="1cm" width="3cm" height="6cm" />
  </svg>]]>
```
XML documents can be commented so that coauthors can leave notes for each other and themselves, documenting why they've done what they've done or items that remain to be done.

<!-- I need to verify and update these links when I get a chance. -->

Comments may appear anywhere in the character data of a document. They may also appear before or after the root element.

XML comments are used to provide information about the XML document. Comments are not considered part of the content. Comment have the following syntax:

<!-- comment text -->

Comments can appear anywhere except inside markup tags and attribute values. XML comments should not be used to transmit information. Comments should contain no entity or character references.

XML provides the processing instruction as a mean of passing information to particular applications that may read the document. A processing instruction begins with <? and ends with ?>. Immediately following the <? is an XML name called the target. Processing instructions are markup, but they're not elements. Consequently, like comments, processing instructions may appear anywhere in an XML document outside of a tag, including before or after the root element.

The XML declaration should (but do not have to) begin with an XML declaration. The XML declaration looks like a processing instruction with the name xml and version, standalone, and encoding attributes. Technically, it’s not a processing instruction though, just the XML declaration:

<?xml version="1.0" encoding="ASCII" standalone="yes"?>

XML documents are composed of pure text. Which encoding? Is it ASCII? Latin-1? Unicode? Something else? By default XML documents are assumed to be encoded in the UTF-8 variable-length encoding of the Unicode character set. However, most XML processors, especially those written in Java, can handle a much broader range of character sets. All you have to do is tell the parser which character encoding the document uses.
Encoding (Con’t)

► An XML document encoded in Latin-1 which includes letters like ö and ç needed for many non-English Western European languages.

```xml
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<person>
  Erwin Schrödinger
</person>
```

Standalone

► If the standalone attribute has the value no, then an application may be required to read an external DTD to determine the proper values for parts of the document.

► For instance, a DTD may provide default values for attributes that a parser is required to report even though they aren’t actually present in the document.

Well-formed XML Examples

► A well formed document with one element:

```xml
<text>This is an XML document</text>
```

Well-formed XML Examples: Match start & end Tag

► The name in an element’s end-tag must match the element type in the start-tag.

► In HTML some elements do not have to have a closing tag. The following code is legal in HTML:

```html
<p>This is a paragraph
  <p>This is another paragraph</p>
</p>
```

Well-Formedness

► Every XML document must be well-formed. This means it must adhere to a number of rules, including the following:
  1. Every start-tag must have a matching end-tag.
  2. Elements may nest, but may not overlap.
  3. There must be exactly one root element.
  4. Attribute values must be quoted.
  5. An element may not have two attributes with the same name.
  6. Comments and processing instructions may not appear inside tags.
  7. No unescaped < or & signs may occur in the character data of an element or attribute.

Well-formed XML Examples: One root element

► There is exactly one element, called the root, or document element, no part of which appears in the content of any other element.

```xml
<name>Binnur Kurt</name>
```

```xml
<name>
  <first>Binnur</first>
  <last>Kurt</last>
</name>
```
Well-formed XML Examples: Element end tag

► Each element has either the end tag or takes the special form.
► There is no difference between <AAA/></AAA> and <AAA/> in XML.

<listOfTags>
  <AAA/></AAA>
  <BBB/></BBB>
  <CCC/>
  <DDD/>
</listOfTags>

Well-formed XML Examples: Attributes

► XML elements can have attributes in name/value pairs.
► Attribute values must always be quoted
► With XML, it is illegal to omit quotation marks around attribute values.

<elements-with-attributes>
  <el Ok = "yes"/>
  <one attr = "a value"/>
  <several first = '1' second = '2' third = "333"/>
  <apos_quote case1 = "John's" case2 = 'He said: "Hello!"'/>
</elements-with-attributes>

XML Quiz 1

► Solution:
  <root>
  <e1 a*b = "23432"/>
  <e2 value = "12"/>
  <e3 value="aa"au"/>
  <e4 value="bbbh"/>
  <e5 xml-ID = "xml2"/>
  </root>

XML Quiz 2

► Find Errors:
  <root>
  <example>
  <![[CDATA[ <P>Q&R]]>]
  </example>
  <Name>
    Binnur Kurt
  </Name>
  <Address/>
  </root>

► Solution:
  No error
XML Quiz 3

Find Errors:

```xml
<root>
  <isLower>
    23 < 46
  </isLower>
  <Name>
    Willey & Sons
  </Name>
</root>
```

Solution:

```xml
<root>
  <isLower>
    23 < 46
  </isLower>
  <Name>
    Willey & Sons
  </Name>
</root>
```

Exercise: Create an XML document

Create an XML document that captures business card information.

Give appropriate tag names.

cd $Lab$/Mod1

Review card.txt – make appropriate changes and create card.xml
XML and Schema
► XML = Portable data
► XML = Unconstrained data
► DTDs and XML Schemas add Constraints to XML.

Valid XML documents
► A “Well Formed” XML document has correct XML syntax.
► A "Valid" XML document is not only well-formed, but conforms to a DTD.

Example: Unconstrained XML document
<?xml version="1.0"?>
<trade account="1234567" action="buy">
    <symbol>SUNW</symbol>
    <symbol>IBM</symbol>
    <symbol>CSCO</symbol>
    <quantity>200</quantity>
    <quantity>100</quantity>
</trade>

XML DTD – Document Type Definition
► A DTD defines the legal elements of an XML document.
► DTD is described in XML 1.0 standard.
► A DTD can be declared inline in your XML document, or as an external reference.
► Internal DOCTYPE declaration:
    <!DOCTYPE root-element [element-declarations]>
► External DOCTYPE declaration:
    <!DOCTYPE root-element SYSTEM "filename">

Example: XML document with a DTD
<?xml version="1.0"?>
<!DOCTYPE note [>
    <!ELEMENT note (to,from,heading,body)>
    <!ELEMENT to (#PCDATA)>
    <!ELEMENT from (#PCDATA)>
    <!ELEMENT heading (#PCDATA)>
    <!ELEMENT body (#PCDATA)>
    ]>
<note>
    <to>J. Canny</to>
    <from>Binnur Kurt</from>
    <heading>Tebrikler</heading>
    <body>Merhaba!!</body>
</note>
**Rules for DTD**

- The document type declaration must appear before the first element in the document.
- The name following the word DOCTYPE in the document type declaration must match the name of root element.
- When an element type has element content, that type must contain only child elements (no character data), optionally separated by white space.

**Example**

- *name.dtd:*
  ```xml
  <!ELEMENT name (first, last)>
  <!ELEMENT first (PCDATA)>
  <!ELEMENT last (PCDATA)>
  <!DOCTYPE name SYSTEM "name.dtd">
  <name>
    <first>Sridhar</first>
    <last>Reddy</last>
  </name>
  ```

**Example: XML document with a DTD**

```xml
<?xml version="1.0"?>
<!DOCTYPE trade [
  <!ELEMENT trade (symbol, quantity)>
  <!ATTLIST trade account CDATA #REQUIRED
  action (buy | sell) #REQUIRED >
  <!ELEMENT symbol (PCDATA)>
  <!ELEMENT quantity (PCDATA)>
  <trade account="1234567" action="buy">
    <symbol>SUNW</symbol>
    <quantity>100</quantity>
  </trade>
```
More Rules for DTD

Here are the qualifiers you can add to an element definition:

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>None</td>
<td>Must occur once</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>One or more</td>
</tr>
<tr>
<td></td>
<td>Plus</td>
<td>Zero or more</td>
</tr>
<tr>
<td></td>
<td>Sub</td>
<td>One element or zero or more</td>
</tr>
<tr>
<td></td>
<td>Question Mark</td>
<td>Optional (zero or one)</td>
</tr>
</tbody>
</table>

Example using *, ?, and + in DTD
tutorial.dtd:

```xml
<!ELEMENT XXX (AAA?, BBB+)>  
<!ELEMENT AAA (CCC?, DDD*)>  
<!ELEMENT BBB (CCC, DDD)>  
<!ELEMENT CCC (#PCDATA)>  
<!ELEMENT DDD (#PCDATA)>  
<!DOCTYPE tutorial SYSTEM "tutorial.dtd">  
```

- The root element XXX can contain one element AAA followed by one or more elements BBB.
- Element AAA can contain one element CCC and several elements DDD.
- Element BBB must contain precisely one element CCC and one element DDD.
More DTD Rules

- With character [ ] you can select one from several elements.
  - `<ELEMENT XXX (AAA , BBB)>`
  - `<ELEMENT AAA (CCC , DDD)>`
  - `<ELEMENT BBB (CCC | DDD)>`
  - `<ELEMENT CCC (#PCDATA)>`
  - `<ELEMENT DDD (#PCDATA)>`
- The root element XXX must contain one element AAA followed by one element BBB.
- Element AAA must contain one element CCC followed by element DDD.
- Element BBB must contain either one element CCC or one element DDD.

Example: Attributes declaration in DTD

- `<ELEMENT attributes (#PCDATA)>`
- `<ATTLIST attributes
  aaa CDATA #REQUIRED
  bbb CDATA #IMPLIED
  ccc CDATA #FIXED "someData" >`
- An attribute of CDATA type can contain any character if it conforms to well formedness constraints.
- Required attribute must be always present.
- Implied attribute is optional.
- If an attribute has a default value declared with the #FIXED keyword, instances of that attribute must match the default value.

DTD Rules

- Text can be interspersed with elements
  - `<ELEMENT XXX (AAA+, BBB+)>`
  - `<ELEMENT AAA (BBB | CCC)>`
  - `<ELEMENT BBB (#PCDATA | CCC)*>`
  - `<ELEMENT CCC (#PCDATA)>`
- The element AAA can contain either BBB or CCC
- On the other hand the element BBB can contain any combination of text and CCC elements.

Attributes declaration in DTD

- Attributes are used to associate name-value pairs with elements.
- Attribute specifications may appear only within start-tags and empty element tags.
- The declaration starts with ATTLIST then follows the name of the element the attributes belong to and then follows the definition of the individual attributes.
- The order of attributes is not important.

Example: Attributes declaration in DTD

- `<DOCTYPE student [
  <!ELEMENT student (name)>
  <!ELEMENT name (#PCDATA)>
  <!ATTLIST student
  DOB CADAT #REQUIRED
  EDU CADAT #IMPLIED> ]`
- `<student DOB="Nov11,1990" EDU="4thGrade">
  <name>Curran Reddy</name>
</student>
- `<student EDU="1st Grade" DOB="May14,1994">
  <name>Kaavya Reddy</name>
</student>`

Rules in DTD: Attribute values

- Permitted attribute values can be defined in DTD.
  - `<ELEMENT SPORT (player+, practicemonth)>`
  - `<ELEMENT player (#PCDATA)>`
  - `<ELEMENT practiceMonth EMPTY)>`
  - `<ATTLIST player meetsHeightReq ( yes | no ) #REQUIRED>`
  - `<ATTLIST practiceMonth (1|2|3|4|5|6|7|8|9|10|11|12) #IMPLIED>`
- The attribute meetsHeightReq cannot have the value "maybe"
- The attribute practiceMonth cannot have the value "16"
**Rules in DTD: Attribute values**

- If an attribute is implied, a default value can be provided for the case when the attribute is not used.
  ```xml
  <!ELEMENT practiceMonth EMPTY>
  <!ATTLIST practiceMonth (1|2|3|4|5|6|7|8|9|10|11|12) "11">
  ```

- An element can be defined EMPTY. In such a case it can contain only attributes but no text.
  ```xml
  <!ELEMENT AAA EMPTY>
  <!ATTLIST AAA true (yes | no ) "yes">
  ```

**Rules in DTD: Parameter-entity**

- Parameter-entity references may only appear in the DTD.
- Parameter-entity references use percent-sign (%) and semicolon (;) as delimiters.
- Example:
  ```xml
  <!ENTITY % auction-req SYSTEM "Auction.dtd">
  ```

- Usage:
  - `%auction-req;`
- Errors:
  - Fatal Error: Missing whitespace after % in parameter entity declaration.
  - Fatal Error: Illegal parameter entity reference syntax. If space after % when referencing.

**DTD Validation**

- Validating with the XML Parser

**Designing an XML Data Structure**

- Whenever possible, use an existing DTD. It's usually a lot easier to ignore the things you don't need than to design your own from scratch.
- Using a standard DTD makes data interchange possible, and may make it possible to use data-aware tools developed by others.
- If an industry standard exists, consider referencing that DTD with an external parameter entity.
- Industry-standard DTDs:
  - [http://www.XML.org](http://www.XML.org)
  - [http://www.xmlx.com](http://www.xmlx.com)

**Attributes and Elements**

- When to model a given data item as a subelement or as an attribute of an existing element?
  - Use `element` if:
    - The data contains substructures
    - The data contains multiple lines
    - The data changes frequently
  - Use `attribute` if:
    - The data is a small, simple string that rarely changes
    - The data is confined to a small number of fixed choices

- Also consideration can be based on how you would like to parse the data.
  ```xml
  <employee>
    <id>12056</id>
    <grade>Manager</grade>
    <division>Engineering</division>
    <description>
      He is a manager of X product. Leads a team of 15 programmers. Also this person is in the special task force team to build next generation app.
    </description>
  </employee>
  ```
**Attributes and Elements**

- or as:

  ```xml
  <employee grade="Manager" id="12056"
  division="Engineering">
    <description>
      He is a manager … generation app
    </description>
  </employee>
  ```

**Limitations of DTDs**

- DTDs are not defined in XML.
- DTDs don't distinguish between different data types, such as dates, integers, and text strings as Data Types are not mentioned.

**Example DTDs and Schemas**

- Electronics Supply Chain (RosettaNet)
- Electronic Business (ebXML - UN/CEFACT & OASIS)
- Finance (IFX)
- Healthcare (HL7)
- Payment Processing (Visa XML)
- Hospitality (HMX-X)
- Trading Partners (tpXML - IBM)
- B2B (eHIS-XML - BASDA)
- Internet billing (Spectrum)
- WAP (WML)
- School Interoperability (SIF)
- Telephony (XML)
- Travel (OTA)
- and many, many, more...

**XML Schema**

- XML Schema facilities for describing the structure and constraining the contents of XML 1.0 documents
- The schema language, which is itself represented in XML 1.0
- XML Schema was a W3C Proposed Recommendation as of March 16, 2001
- XML Schema is now a W3C Recommendation as of May 2nd 2001.
  - http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

**How Will Agreements Be Shared?**

- We need to publish XML agreements on the Internet:
  - human search: what industry standard meets my needs?
  - machine resolution/namespaces
  - versioning: what is the latest version of a DTD or Schema?
  - archiving: what is the meaning of documents generated 20 years ago?
- Requirement for XML registry and repository:
  - developed/implemented to open specifications
  - accessible by everyone
  - unencumbered use of repository

**XML.ORG Registry**

- Central clearinghouse for accessing XML schemas, vocabularies and related documents
- Self-supporting, non-commercial resource created by OASIS for the community at large
- Model for a distributed web of repositories that will comply with OASIS Technical Committee specification
Exercise: Create a DTD

Part 1:
- Create a DTD for the XML document that captured the information about business card in lab 1.

Part 2:
- Compare your DTD to the person sitting next to you and see if you can combine into one which can still validate both the XML documents.

Atomic types

- string, integer, boolean, date, …
- enumeration types
- restriction and range [a-z]
- list: list of values of an atomic type, …

DTDs vs. schemas (types)

- By database (or programming language) standard, XML DTDs are rather weak specifications.
  - Only one base type -- PCDATA.
  - No useful "abstractions", e.g., unshared records.
  - No sub-typing or inheritance.
  - IDREFs are not typed or scoped -- you point to something, but you don’t know what!
- XML extensions to overcome the limitations.
  - Type systems: XML-Data, XML-Schema, SOX, DCD
  - Integrity Constraints

Atomic types: Examples

Example: define an element or an attribute
```xml
<xs:element name="car" type="carType"/>
<xs:attribute name="color" type="carType"/>
```

Define the type:
```xml
<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

XML Schema

- Official W3C Recommendation

- A rich type system
  - Simple (atomic, basic) types for both element and attributes
  - Complex types for elements
  - Inheritance
  - Constraints
    - key
    - keyref (foreign keys)
    - uniqueness: “more general” keys
  - Namespace

Complex types

- Sequence: “record type” – ordered
- All: record type – unordered
- Choice: variant type
- Occurrence constraint: maxOccurs, minOccurs
- Group: mimicking parameter type to facilitate complex type definition
- Any: “open” type – unrestricted
**Example**

A complex type for publications:

```xml
<xs:complexType name="publicationType">
  <xs:sequence>
    <xs:choice>
      <xs:group ref="journalType">
        <xs:element name="conference" type="xs:string"/>
      </xs:choice>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="xs:string" minOccur="0" maxOccur="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

**Inheritance — Restriction**

Supertype: restricting/removing certain fields of an existing type

```xml
<xs:complexType name="anotherPublicationType">
  <xs:complexContent>
    <xs:restriction base="publicationType">
      <xs:sequence>
        <xs:choice>
          <xs:group ref="journalType">
            <xs:element name="conference" type="xs:string"/>
          </xs:choice>
          <xs:element name="author" type="xs:string" minOccur="1" maxOccur="unbounded"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
```

**Example (cont’d)**

```xml
<xs:group name="journalType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="volume" type="xs:integer"/>
    <xs:element name="number" type="xs:integer"/>
  </xs:sequence>
</xs:group>
```

**Inheritance — Extension**

Subtype: extending an existing type by including additional fields

```xml
<xs:complexType name="datedPublicationType">
  <xs:complexContent>
    <xs:extension base="publicationType">
      <xs:sequence>
        <xs:choice>
          <xs:group ref="journalType">
            <xs:element name="conference" type="xs:string"/>
          </xs:choice>
          <xs:element name="author" type="xs:string" minOccur="0" maxOccur="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

**Content**

Creating Basic Documents

Use HTML to structure a basic Web page by manipulating text and graphics
Overview
► Hypertext Markup Language (HTML) is the foundation of all Web documents, or pages.
► This module will teach you the basic elements and architecture of a web page and how HTML brings it together.
► You will learn to identify tags, view code, create hierarchies, format text and pages, and use styles and style sheets.

Introduction
► It is important to know the elements involved with building Web pages. Upon completion of this lesson, you will be able to:
  – Summarize Web page creation
  – List the elements of HTML
  – Identify HTML tags
  – View and evaluate HTML code in a page

Basics
► The pages created with HTML are plain text.
► You can create, edit, or view the HTML code for a Web page in any text editor, such as Windows Notepad, on any computer platform.

Web-Authoring Tools
► Although creating simple Web pages in a text editor is easy, it can become a tedious task.
► Web-authoring tools, such as Microsoft FrontPage Express and Macromedia Dreamweaver, allow you to create HTML Web pages in the same way word documents are created.

Browsers
► When a Web page is open in a browser, the HTML code that creates the page is not visible.
► Instead, the browser interprets the HTML code and displays the page appropriately on the screen.

Edit Views
► If you are creating a Web page in a text editor and want to view the file, save your work and open the file in the browser.
► You can then continue to edit, save your work, and view the results, switching between the text editor and the browser to see the effects of the edits.
Browser Limitations
► The original intent of the HTML specification was to allow Web authors to describe the structure of a page without spending time on the look of a page.
► Traditionally, each browser had its own way of interpreting the look of the page, and the page created would appear differently on different browsers.

Accurate Page Descriptions
► HTML can now describe the look of a page accurately.
► More and more descriptive elements are being established in the HTML specification, such as font styles, sizes, colors, and style sheets that can maintain a consistent look throughout a Web site.

View Testing
► When a browser differentiates a non-specific message, one browser might display the first-level heading in a large font centered on the page, while another browser might display it in italics and left-aligned on the page.
► As a result, authors test pages in several of the more popular browsers.

Text Limitations
► An author can specify a paragraph of text be defined as one of the six heading levels, but the HTML heading code cannot describe what a heading should look like.
► The Web browser must differentiate each type of heading from the others.

Elements
► A Web page consists of elements, each of which is defined by an HTML code, or tag.
Tag Architecture

► A tag is always enclosed in angle brackets, and most tags come in pairs, with an opening and a closing tag.
► The closing tag is the same as the opening tag, but starts with a forward slash.
► For example, to define text as a first-level heading in HTML, use the `<H1>` tag, as shown.

![Opening Tag](image1.png) ![Closing Tag](image2.png)

```xml
<H1>This is a main heading</H1>
```

Browsers

► A browser interprets tags and displays the text within the tags appropriately.
► The tags themselves are not displayed within a browser unless there is a problem with a tag such as if one of the angle brackets was mistakenly left out.
► Most browsers will ignore any codes within angle brackets that they do not recognize.

Attributes

► Some tags have optional or required attributes. An attribute is usually a keyword that takes one of several possible values.
► A value is defined by enclosing it in quotes.
► For example, the heading tag can take an optional alignment attribute.

HEAD tag

► The `<HEAD>` tag defines the header area of a page, which is not displayed within the page itself in the browser.
► The closing `</HEAD>` tag ends the header area.
Title Tag
► The text between <TITLE> and the closing </TITLE> tag is the title of the Web page and is displayed in the title bar of a browser.
► The title should be descriptive; as it is frequently used by Web indexing and searching programs to name the Web page.
► In Internet Explorer, a page title is the default name used when you save the page as a favorite location.

Comment tag
► The Comment tag is not revealed in the browser, but will be advantageous when editing or viewing the HTML code for a page.
► In Internet Explorer, use the <!---> tag to create descriptive comments within the code, which will be ignored by the browser.
► With other browsers, you can use the combination of symbols to create a comment.

<body>

Paragraph tag
► Use the paragraph tag (<P>) to mark the beginning of a new paragraph; the ending tag, </P>, is optional but should be included for clarity during revisions.
► You can include the ALIGN attribute to specify whether the paragraph should be centered or right-aligned in the page.

Hard Return
► HTML code ignores all hard returns within the code: for example, an Enter at the end of a text line you are editing in Notepad will not be displayed in the user's browser.
Preformatted tag

► In the preformatted tag <PRE>, spaces and hard returns in the HTML code do display.
► It instructs a browser to display the text in a mono-spaced font that allows you to align text precisely, such as you would when showing a program listing.

Save as View

► To view a page's code, click View > Source in Internet Explorer or View > Page Source in Navigator to display the current page's HTML code within Notepad.

Information

► For additional documentation on tags used in HTML http://werbach.com/barebones/barebones.html

Evolving Language

► HTML is constantly evolving.
► Web authors may include new and improved tags within Web pages to produce new effects.
► But browser software may not recognize those HTML features.
► W3C at the MIT defines and establishes new versions of HTML to help with this problem.
► Unofficially, leaders such as Microsoft and Netscape, regularly invent their own extensions to official HTML which eventually may be included in the official HTML specification.

Underlying Code

► Most Web pages are built from the same text-based HTML language, so when viewing an interesting page in a browser, take a look at the underlying code.

Structuring Web Pages

► When developing a Web page, you must determine a structure for your text and images using HTML that best suites your organization.
► Upon completion of this lesson, you will be able to:
  – Add structure
  – Divide sections
  – Create hierarchies
  – Format text and pages
  – Use styles and style sheets
Adding Structure
► Adding structure will benefit any Web page you create.
► To add basic structure to a page, add paragraphs and spaces to text.
► To create a paragraph, enclose text within the paragraph tags (<P> and </P>).
► Your browser will insert some extra space between paragraphs, so in some instances, you will not want to use the <P> tag.

Structure Separations
► If your page has a banner across the top with a company name, you can insert a horizontal line beneath it.
► This separates it from the table of contents showing links to pages, beneath which you can insert another line, followed by the main body of the page.
► At the bottom of the page, you can have another line that shows important page identifiers.

Line Break tag
► You may not want extra space between each line of the address.
► To avoid this, use the line-break tag, <BR>.
► This break tells the browser to wrap the text that follows onto a new line without inserting any extra space between the lines.

HR tag
► The <HR> tag takes several optional attributes. For example, you can specify the line's thickness and how much of the browser's window it should span (as a percentage or in pixels).
► The line thickness default is one or two pixels in most browsers.

Section Divisions
► A simple and effective way to separate sections within a Web page is to insert a horizontal line, which is also called a horizontal rule.
► By default, the line stretches from one side of the page to the other.

Creating Headings
► A common way to add structure to a Web page is through the use of headings.
► A Web page can have a maximum of six levels of headings, the HTML codes for which are conveniently named <H1>, <H2>, <H3>, etc.
► No style is inherent in the headings — different Web browsers might interpret the look of a heading in slightly different ways.
► Structurally, however, all browsers will display headings so low-level headings look subordinate to a higher-level heading.
### Heading Formats

- In a browser, a first-level heading is displayed in a larger, bolder font than a lower-level heading.
- You can use HTML headings in any order, but it is recommended to use them in an outline format.
- The first-level heading, `<H1>`, is the highest level, and the sixth level, `<H6>`, is the lowest or most subordinate.

### Comments in HTML

- The comment tag is used to insert a comment in the HTML source code. A comment will be ignored by the browser. You can use comments to explain your code, which can help you when you edit the source code at a later date.
- Note that you need an exclamation point after the opening bracket, but not before the closing bracket.

### Paragraphs

- Paragraphs are defined with the `<p>` tag.
  ```html
  <p>This is a paragraph</p>
  <p>This is another paragraph</p>
  ```
- HTML automatically adds an extra blank line before and after a paragraph.

### Text Formatting Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;b&gt;</code></td>
<td>Defines bold text</td>
</tr>
<tr>
<td><code>&lt;big&gt;</code></td>
<td>Defines big text</td>
</tr>
<tr>
<td><code>&lt;em&gt;</code></td>
<td>Defines emphasized text</td>
</tr>
<tr>
<td><code>&lt;i&gt;</code></td>
<td>Defines italic text</td>
</tr>
<tr>
<td><code>&lt;small&gt;</code></td>
<td>Defines small text</td>
</tr>
<tr>
<td><code>&lt;strong&gt;</code></td>
<td>Defines strong text</td>
</tr>
<tr>
<td><code>&lt;sub&gt;</code></td>
<td>Defines subscripted text</td>
</tr>
<tr>
<td><code>&lt;sup&gt;</code></td>
<td>Defines superscripted text</td>
</tr>
<tr>
<td><code>&lt;ins&gt;</code></td>
<td>Defines inserted text</td>
</tr>
<tr>
<td><code>&lt;del&gt;</code></td>
<td>Defines deleted text</td>
</tr>
<tr>
<td><code>&lt;s&gt;</code></td>
<td>Deprecated. Use <code>&lt;del&gt;</code> instead</td>
</tr>
<tr>
<td><code>&lt;strike&gt;</code></td>
<td>Deprecated. Use <code>&lt;del&gt;</code> instead</td>
</tr>
<tr>
<td><code>&lt;u&gt;</code></td>
<td>Deprecated. Use styles instead</td>
</tr>
</tbody>
</table>

### Line Breaks

- The `<br>` tag is used when you want to end a line, but don't want to start a new paragraph.
- The `<br>` tag forces a line break wherever you place it.
  ```html
  <p>This <br> is a paragraph</p>
  ```
- The `<br>` tag is an empty tag.
- It has no closing tag.

### "Computer Output" Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;code&gt;</code></td>
<td>Defines computer code text</td>
</tr>
<tr>
<td><code>&lt;kbd&gt;</code></td>
<td>Defines keyboard text</td>
</tr>
<tr>
<td><code>&lt;samp&gt;</code></td>
<td>Defines sample computer code</td>
</tr>
<tr>
<td><code>&lt;tt&gt;</code></td>
<td>Defines teletype text</td>
</tr>
<tr>
<td><code>&lt;var&gt;</code></td>
<td>Defines a variable</td>
</tr>
<tr>
<td><code>&lt;pre&gt;</code></td>
<td>Defines preformatted text</td>
</tr>
<tr>
<td><code>&lt;listing&gt;</code></td>
<td>Deprecated. Use <code>&lt;pre&gt;</code> instead</td>
</tr>
<tr>
<td><code>&lt;plaintext&gt;</code></td>
<td>Deprecated. Use <code>&lt;pre&gt;</code> instead</td>
</tr>
<tr>
<td><code>&lt;xmp&gt;</code></td>
<td>Deprecated. Use <code>&lt;pre&gt;</code> instead</td>
</tr>
</tbody>
</table>
Character Entities

- Some characters have a special meaning in HTML, like the less than sign («) that defines the start of an HTML tag.
- If we want the browser to actually display these characters we must insert character entities in the HTML source.
- A character entity has three parts: an ampersand (&), an entity name or a # and an entity number, and finally a semicolon (;).
- To display a less than sign in an HTML document we must write: &lt; or &lt;.
- The advantage of using a name instead of a number is that a name is easier to remember.
- The disadvantage is that not all browsers support the newest entity names, while the support for entity numbers is very good in almost all browsers.

Non-breaking Space

- The most common character entity in HTML is the non-breaking space.
- Normally HTML will truncate spaces in your text.
- If you write 10 spaces in your text, HTML will remove 9 of them.
- To add spaces to your text, use the &nbsp; character entity.

The Anchor Tag and the href Attribute

- HTML uses the <a> (anchor) tag to create a link to another document.
- An anchor can point to any resource on the Web: an HTML page, an image, a sound file, a movie, etc.
- The syntax of creating an anchor:
  
```html
  <a href="url">Text to be displayed</a>
```
- The <a> tag is used to create an anchor to link from, the href attribute is used to address the document to link to, and the words between the open and close of the anchor tag will be displayed as a hyperlink.
The Target Attribute

► With the target attribute, you can define where the linked document will be opened.
► The line below will open the document in a new browser window:

```html
<a href="http://www.itu.edu.tr" target="_blank">
Visit ITU!
</a>
```

The Anchor Tag and the Name Attribute

► The name attribute is used to create a named anchor.
► When using named anchors we can create links that can jump directly into a specific section on a page, instead of letting the user scroll around to find what he/she is looking for.
► Below is the syntax of a named anchor:
  ```html
  <a name="label">Text to be displayed</a>
  ```
► The line below defines a named anchor:
  ```html
  <a name="tips">Useful Tips Section</a>
  ```

Frames

► With frames, you can display more than one HTML document in the same browser window.
► Each HTML document is called a frame, and each frame is independent of the others.
► The disadvantages of using frames are:
  - The web developer must keep track of more HTML documents
  - It is difficult to print the entire page

Frameset Tag

► The `<frameset>` tag defines how to divide the window into frames
► Each frameset defines a set of rows or columns
► The values of the rows/columns indicate the amount of screen area each row/column will occupy

```
<frameset cols="25%,75%">
  <frame src="frame_a.htm">
  <frame src="frame_b.htm">
</frameset>
```
Tables

- Tables are defined with the `<table>` tag.
- A table is divided into rows (with the `<tr>` tag), and each row is divided into data cells (with the `<td>` tag).
- The letters `td` stands for "table data," which is the content of a data cell.
- A data cell can contain text, images, lists, paragraphs, forms, horizontal rules, tables, etc.

Example

```html
<table border="1">
  <tr>
    <td>row 1, cell 1</td>
    <td>row 1, cell 2</td>
  </tr>
  <tr>
    <td>row 2, cell 1</td>
    <td>row 2, cell 2</td>
  </tr>
</table>
```

Tables and the Border Attribute

- If you do not specify a border attribute the table will be displayed without any borders. Sometimes this can be useful, but most of the time, you want the borders to show.
- To display a table with borders, you will have to use the `border` attribute:

```html
<table border="5">
  <tr>
    <td>Row 1, cell 1</td>
    <td>Row 1, cell 2</td>
  </tr>
</table>
```
Empty Cells in a Table

Table cells with no content are not displayed very well in most browsers.

```html
<table border="1">
  <tr>
    <td>row 1, cell 1</td>
    <td>row 1, cell 2</td>
  </tr>
  <tr>
    <td>row 2, cell 1</td>
  </tr>
</table>
```

Table Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;table&gt;</code></td>
<td>Defines a table</td>
</tr>
<tr>
<td><code>&lt;th&gt;</code></td>
<td>Defines a table header</td>
</tr>
<tr>
<td><code>&lt;tr&gt;</code></td>
<td>Defines a table row</td>
</tr>
<tr>
<td><code>&lt;td&gt;</code></td>
<td>Defines a table cell</td>
</tr>
<tr>
<td><code>&lt;caption&gt;</code></td>
<td>Defines a table caption</td>
</tr>
<tr>
<td><code>&lt;colgroup&gt;</code></td>
<td>Defines groups of table columns</td>
</tr>
<tr>
<td><code>&lt;col&gt;</code></td>
<td>Defines the attribute values for one or more columns in a table</td>
</tr>
<tr>
<td><code>&lt;thead&gt;</code></td>
<td>Defines a table head</td>
</tr>
<tr>
<td><code>&lt;tbody&gt;</code></td>
<td>Defines a table body</td>
</tr>
<tr>
<td><code>&lt;tfoot&gt;</code></td>
<td>Defines a table footer</td>
</tr>
</tbody>
</table>

Ordered Lists

An ordered list is also a list of items. The list items are marked with numbers.

An ordered list starts with the `<ol>` tag. Each list item starts with the `<li>` tag.

```html
<ol>
  <li>Coffee</li>
  <li>Milk</li>
</ol>
```

Definition Lists

A definition list is not a list of items. This is a list of terms and explanation of the terms.

A definition list starts with the `<dl>` tag. Each definition-list term starts with the `<dt>` tag.

Each definition-list definition starts with the `<dd>` tag.

```html
<dl>
  <dt>Coffee</dt>
  <dd>Black hot drink</dd>
  <dt>Milk</dt>
  <dd>White cold drink</dd>
</dl>
```

Unordered Lists

An unordered list is a list of items. The list items are marked with bullets (typically small black circles).

An unordered list starts with the `<ul>` tag. Each list item starts with the `<li>` tag.

```html
<ul>
  <li>Coffee</li>
  <li>Milk</li>
</ul>
```

Inside a list item you can put paragraphs, line breaks, images, links, other lists, etc.

List Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;ol&gt;</code></td>
<td>Defines an ordered list</td>
</tr>
<tr>
<td><code>&lt;ul&gt;</code></td>
<td>Defines an unordered list</td>
</tr>
<tr>
<td><code>&lt;li&gt;</code></td>
<td>Defines a list item</td>
</tr>
<tr>
<td><code>&lt;dl&gt;</code></td>
<td>Defines a definition list</td>
</tr>
<tr>
<td><code>&lt;dt&gt;</code></td>
<td>Defines a definition term</td>
</tr>
<tr>
<td><code>&lt;dd&gt;</code></td>
<td>Defines a definition description</td>
</tr>
<tr>
<td><code>&lt;dir&gt;</code></td>
<td>Deprecated. Use <code>&lt;ul&gt;</code> instead</td>
</tr>
<tr>
<td><code>&lt;menu&gt;</code></td>
<td>Deprecated. Use <code>&lt;ul&gt;</code> instead</td>
</tr>
</tbody>
</table>
HTML Forms and Input

▶ HTML Forms are used to select different kinds of user input.

```html
<form name="input" action="html_form_action.asp" method="get">
  <input type="text" name="FirstName" value="Binnur" size="20">
  <br>Type your last name:
  <input type="text" name="LastName" value="Kurt" size="20">
  <br>
  <input type="submit" value="Submit">
</form>
```

Introducing Web Forms

▶ Web forms collect information from customers.
▶ Web forms include different control elements including:
  - Input boxes
  - Selection lists
  - Drop-down list boxes
  - Option buttons or radio buttons
  - Check boxes
  - Group boxes
  - Text areas
  - Form buttons

Creating the Form Element

▶ Forms are created using the form element, structured as follows:

```html
<form attributes>
  elements
</form>
```
▶ where attributes are the attributes that control how the form is processed and elements are elements places within the form.

Forms and Server-Based Programs

▶ Server-based programs are written in many languages
▶ The earliest and most commonly used are Common Gateway Interface (CGI) scripts that are written in perl.
▶ Other popular languages include:
  - AppleScript - PHP
  - ASP - TCL
  - ColdFusion - the Unix shell
  - C/C++ - Visual Basic

Creating the Form Element

▶ Form attributes usually tell the browser the location of the server-based program to be applied to the form’s data.
▶ Always specify an id or name for the form.
▶ Two attributes are available to identify the form: id and name.
Creating the Form Element

The syntax of the id and name attributes are as follows:

```
<form name="name" id="id"> ... </form>
```

Example:

```
<form name="reg" id="reg"> ... </form>
```

Creating Input Boxes

The general syntax of input elements is as follows:

```
<input type="type" name="name" id="id" />
```

where `type` specifies the type of input field, and the `name` and `id` attributes provide the field's name and id.

```
<input type="text" name="city" id="city" />
```

Form Input Types: `<input type=...>`

<table>
<thead>
<tr>
<th>Type</th>
<th>Description of what is displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>type=&quot;button&quot;</code></td>
<td>Button can be clicked to perform an action from a script</td>
</tr>
<tr>
<td><code>type=&quot;checkbox&quot;</code></td>
<td>Check box</td>
</tr>
<tr>
<td><code>type=&quot;file&quot;</code></td>
<td>Browse button to locate and select a file</td>
</tr>
<tr>
<td><code>type=&quot;hidden&quot;</code></td>
<td>Hidden field, not viewable on the form</td>
</tr>
<tr>
<td><code>type=&quot;image&quot;</code></td>
<td>clickable inline image - performs an action from a script</td>
</tr>
<tr>
<td><code>type=&quot;password&quot;</code></td>
<td>Text box in which hides text entered by the user</td>
</tr>
<tr>
<td><code>type=&quot;radio&quot;</code></td>
<td>radio (option) button</td>
</tr>
<tr>
<td><code>type=&quot;reset&quot;</code></td>
<td>Button which resets the form when clicked</td>
</tr>
<tr>
<td><code>type=&quot;submit&quot;</code></td>
<td>Button which submits the form when clicked</td>
</tr>
<tr>
<td><code>type=&quot;text&quot;</code></td>
<td>Text box in which displays text entered by the user</td>
</tr>
</tbody>
</table>

Working with Form Labels

Link a label with an associated text element for scripting purposes.

The syntax for creating a form label is as follows:

```
<label for="zip">Zip Code</label>
```

Where `id` is the value of the id attribute for a field on the form, and `label text` is the text of the label.
Creating a Selection List

► A selection list
  – list box from which a user selects a particular value or set of values.
  – Selection lists are useful when there are a fixed set of possible responses from the user.
► create selection list using <select> tag.
► specify each individual selection item using the <option> tag.

Modify the Appearance of a Selection List

► change the number of options displayed in the selection list
  – by modifying the size attribute.
  – Syntax:
    `<select size="value">…</select>`
  – Where value is the number of items that the selection list displays in the form.

Select list

```html
<form id="reg">
  <select name="item" id="item">
    <option>dogs</option>
    <option>cats</option>
    <option>mice</option>
  </select>
</form>
```

Making Multiple Selections

► Add the multiple attribute to the select element to create multiple selections

```html
<select multiple="multiple">…</select>
```

Working with Option Groups

Use option groups to organize selection lists into distinct groups.

```html
<select attributes>
  <optgroup label="label1">
    <option>item1</option>
    <option>item2</option>
  </optgroup>
  …
</select>
```

```html
<select attributes>
  <optgroup label="label2">
    <option>item1</option>
    <option>item2</option>
  </optgroup>
  …
</select>
```
Creating a Field Set

► HTML and XHTML allow you to organize option buttons into a group of fields called field sets.
  – Most browsers place a group box around a field set
  – indicates fields belong to a common group.
    
    ```html
    <fieldset> fields </fieldset>
    ```
  – where fields are the individual fields within a set.

Creating Check Boxes

► To create a check box, use:
  
  ```html
  <input type="checkbox" name="name" id="id" value="value"/>
  ```
  – where the name and id attributes identify the check box field and
    – the value attribute specifies the value sent to the server if the check box is selected.
  
  ► To specify that a check box be selected by default, use the checked attribute as follows:
    
    ```html
    <input type="checkbox" checked="checked"/>
    ```
    
    or
    
    ```html
    <input type="checkbox" checked/>
    ```

Creating a Text Area Box

► Text area boxes allow users to enter comments about the products they've purchased.
  ► An input box would be too small to accommodate the length of text for this use.

Creating a Command button

► Command buttons are created using the <input> tag:
  
  ```html
  <input type="button" value="text" />
  ```
  
  ► Submit buttons submit forms to the server for processing when clicked. Syntax is as follows:
    
    ```html
    <input type="submit" value="text" />
    ```
  
  ► Reset buttons reset forms to their original (default) values. Syntax is as follows:
    
    ```html
    <input type="reset" value="text" />
    ```

Creating a Text Area Box

► To create a text area box, use the textarea element:
  
  ```html
  <textarea name="name" id="id" rows="value" cols="value">
  default text
  </textarea>
  ```
  Where the rows and cols attributes define
  – dimensions of the input box
  – rows attribute indicates the number of lines in the input box.
Creating a File button

► File buttons are used to select files so that their contents can be submitted for processing to a program.
► The Web page then only displays the file’s location, not the file’s contents.

Working with Form Attributes

► The method attribute can have one of two values:
  – Post
  – Get
► The get method is the default
  – get appends the form data to the end of the URL specified in the action attribute.
► The post method sends form data in a separate data stream
  – allows the Web server to receive the data through “standard input”.

Working with Hidden Fields

► Hidden fields are added to a form, but not displayed in the Web page.
► The syntax is as follows:
  
  ```html
  <input type="hidden" name="name" id="id" value="value" />
  ```

Using the mailto Action

► The mailto action accesses the user’s own e-mail program and uses it to mail form information to a specified e-mail address.
► By-passes the need for server-based programs.
► Syntax:
  
  ```html
  <form action=mailto:e-mail_address method="post" enctype="text/plain">
  …
  </form>
  ```
  – where `e-mail_address` is the e-mail address of the recipient in the form.
► Newer browsers do not allow this action for security reasons

Specifying the Tab Order

► Users typically navigate through a form with the tab key
► Specify an alternate tab order by adding the tabindex attribute to any control element in your form.
► Example:
  
  ```html
  <input name="fname" tabindex="1" />
  ```
  – assigns the tab index number “1” to the fname field from the registration form.
Specifying an Access Key

► An access key
  – single key typed with the Alt key
  – in order to jump to one of the control elements in the form.
► Create an access key by adding the accesskey attribute to any control element.
► Example of creating an access key for the lname field:
  `<input name="lname" accesskey="1" />`

The Alt Attribute

► The alt attribute is used to define an "alternate text" for an image.
► The value of the alt attribute is an author-defined text
  `<img src="boat.gif" alt="Big Boat" />`
► The "alt" attribute tells the reader what he or she is missing on a page if the browser can't load images. The browser will then display the alternate text instead of the image.
► It is a good practice to include the "alt" attribute for each image on a page, to improve the display and usefulness of your document for people who have text-only browsers.

Form Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;form&gt;</td>
<td>Defines a form for user input</td>
</tr>
<tr>
<td>&lt;input&gt;</td>
<td>Defines an input field</td>
</tr>
<tr>
<td>&lt;textarea&gt;</td>
<td>Defines a text-area (a multi-line text input control)</td>
</tr>
<tr>
<td>&lt;label&gt;</td>
<td>Defines a label to a control</td>
</tr>
<tr>
<td>&lt;fieldset&gt;</td>
<td>Defines a fieldset</td>
</tr>
<tr>
<td>&lt;legend&gt;</td>
<td>Defines a caption for a fieldset</td>
</tr>
<tr>
<td>&lt;select&gt;</td>
<td>Defines a selectable list (a drop-down box)</td>
</tr>
<tr>
<td>&lt;optgroup&gt;</td>
<td>Defines an option group</td>
</tr>
<tr>
<td>&lt;option&gt;</td>
<td>Defines an option in the drop-down box</td>
</tr>
<tr>
<td>&lt;button&gt;</td>
<td>Defines a push button</td>
</tr>
</tbody>
</table>

Backgrounds

► The <body> tag has two attributes where you can specify backgrounds. The background can be a color or an image.
► The bgcolor attribute specifies a background-color for an HTML page.
► The value of this attribute can be a hexadecimal number, an RGB value, or a color name
  `<body bgcolor="#000000"/>
  <body bgcolor="rgb(0,0,0)"/>
  <body bgcolor="black"/>`
► The lines above all set the background-color to black

The Image Tag and the src Attribute

► In HTML, images are defined with the <img> tag.
► The <img> tag is empty, which means that it contains attributes only and it has no closing tag
► To display an image on a page, you need to use the src attribute
► src stands for "source"
► The value of the src attribute is the URL of the image you want to display on your page
► The syntax of defining an image:
  `<img src="url"/>`

Background

► The background attribute specifies a background-image for an HTML page.
► The value of this attribute is the URL of the image you want to use.
► If the image is smaller than the browser window, the image will repeat itself until it fills the entire browser window
  `<body background="clouds.gif"/>
  <body background="http://www.cs.itu.edu.tr/bg.gif"/>`
Useful Tips

► The bgcolor, background, and the text attributes in the <body> tag are deprecated in the latest versions of HTML (HTML 4 and XHTML).
► The World Wide Web Consortium (W3C) has removed these attributes from its recommendations.
► Style sheets (CSS) should be used instead (to define the layout and display properties of HTML elements).

Color Values

► Colors are defined using a hexadecimal notation for the combination of Red, Green, and Blue color values (RGB).
► The lowest value that can be given to one light source is 0 (#00). The highest value is 255 (#FF).

<table>
<thead>
<tr>
<th>Color</th>
<th>Color HEX</th>
<th>Color RGB</th>
</tr>
</thead>
<tbody>
<tr>
<td>black</td>
<td>#000000</td>
<td>rgb(0,0,0)</td>
</tr>
<tr>
<td>darkred</td>
<td>#800000</td>
<td>rgb(128,0,0)</td>
</tr>
<tr>
<td>darkgreen</td>
<td>#008000</td>
<td>rgb(0,128,0)</td>
</tr>
<tr>
<td>darkblue</td>
<td>#000080</td>
<td>rgb(0,0,128)</td>
</tr>
<tr>
<td>darkmagenta</td>
<td>#800080</td>
<td>rgb(128,0,128)</td>
</tr>
<tr>
<td>darkgreen</td>
<td>#008080</td>
<td>rgb(0,128,128)</td>
</tr>
<tr>
<td>darkslateblue</td>
<td>#483D8B</td>
<td>rgb(72,61,139)</td>
</tr>
<tr>
<td>darkgoldenrod</td>
<td>#ba5500</td>
<td>rgb(182,85,0)</td>
</tr>
<tr>
<td>darkolive</td>
<td>#556b2f</td>
<td>rgb(85,107,47)</td>
</tr>
<tr>
<td>darkorchid</td>
<td>#9932cc</td>
<td>rgb(153,50,204)</td>
</tr>
<tr>
<td>darkmagenta</td>
<td>#800080</td>
<td>rgb(128,0,128)</td>
</tr>
<tr>
<td>darkgreen</td>
<td>#008080</td>
<td>rgb(0,128,128)</td>
</tr>
<tr>
<td>darkgrey</td>
<td>#a9a9a9</td>
<td>rgb(169,169,169)</td>
</tr>
<tr>
<td>deepskyblue</td>
<td>#00bfff</td>
<td>rgb(0,191,255)</td>
</tr>
<tr>
<td>deeppink</td>
<td>#ff1493</td>
<td>rgb(255,20,147)</td>
</tr>
<tr>
<td>deepskyblue</td>
<td>#00bfff</td>
<td>rgb(0,191,255)</td>
</tr>
<tr>
<td>deepskyblue</td>
<td>#00bfff</td>
<td>rgb(0,191,255)</td>
</tr>
</tbody>
</table>

16 Million Different Colors

► The combination of Red, Green and Blue values from 0 to 255 gives a total of more than 16 million different colors to play with (256 x 256 x 256).
► Most modern monitors are capable of displaying at least 16384 different colors.
► If you look at the color table (next slide), you will see the result of varying the red light from 0 to 255, while keeping the green and blue light at zero.

Color Names

► A collection of color names is supported by most browsers.
► Only 16 color names are supported by the W3C HTML 4.0 standard (aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow).
► For all other colors you should use the Color HEX value.
What is XHTML?
► XHTML stands for Extensible HyperText Markup Language
► XHTML is aimed to replace HTML
► XHTML is almost identical to HTML 4.01
► XHTML is a stricter and cleaner version of HTML
► XHTML is HTML defined as an XML application
► XHTML is a W3C Recommendation

Why XHTML?
► We have reached a point where many pages on the WWW contain "bad" HTML.
► The following HTML code will work fine if you view it in a browser, even if it does not follow the HTML rules:
  \[<html><head><title>This is bad HTML</title><body><h1>Bad HTML</h1></body></html>\]
► XML is a markup language where everything has to be marked up correctly, which results in "well-formed" documents.
► XML was designed to describe data and HTML was designed to display data.

XHTML is a W3C Recommendation
► XHTML 1.0 became a W3C Recommendation January 26, 2000.
► A W3C Recommendation means that the specification is stable, that it has been reviewed by the W3C membership, and that the specification is now a Web standard.
► W3C defines XHTML as the latest version of HTML. XHTML will gradually replace HTML.

Why XHTML?
► Today's market consists of different browser technologies, some browsers run Internet on computers, and some browsers run Internet on mobile phones and handhelds.
► Therefore - by combining HTML and XML, and their strengths, we got a markup language that is useful now and in the future - XHTML.
► XHTML pages can be read by all XML enabled devices AND while waiting for the rest of the world to upgrade to XML supported browsers, XHTML gives you the opportunity to write "well-formed" documents now, that work in ALL browsers and that are backward browser compatible.
How To Get Ready For XHTML

► XHTML is the next generation of HTML, but it will of course take some time before browsers and other software products are ready for it.
► In the meantime there are some important things you can do to prepare yourself for it.
► XHTML is not very different from HTML 4.01, so bringing your code up to 4.01 standards is a very good start.
► In addition, you should start NOW to write your HTML code in lowercase letters, and NEVER make the bad habit of skipping end tags like the </p>

The Most Important Differences

► XHTML elements must be **properly nested**
► XHTML documents must be **well-formed**
► Tag names must be in **lowercase**
► All XHTML elements must be **closed**

Elements Must Be Properly Nested

► In HTML some elements can be improperly nested within each other like this:
  <b><i>This text is bold and italic</i></b>
► In XHTML all elements must be properly nested within each other like this:
  <b><i>This text is bold and italic</i></b>

Documents Must Be Well-formed

► All XHTML elements must be nested within the <html> root element.
► All other elements can have sub (children) elements.
► Sub elements must be in pairs and correctly nested within their parent element.
► The basic document structure is:
  <html>
  <head> ... </head>
  <body> ... </body>
  <html>

Quiz

<ul>
  <li>Coffee</li>
  <li>Tea
    <ul>
      <li>Black tea</li>
      <li>Green tea</li>
    </ul>
  </li>
  <li>Milk</li>
</ul>

Answer

<ul>
  <li>Coffee</li>
  <li>Tea
    <ul>
      <li>Black tea</li>
      <li>Green tea</li>
    </ul>
  </li>
  <li>Milk</li>
</ul>
Tag Names Must Be In Lower Case

- XHTML documents are XML applications.
- XML is case-sensitive.
- Tags like `<br>` and `<BR>` are interpreted as different tags.

This is wrong:

```xml
<BODY>
  <P>This is a paragraph</P>
</BODY>
```

This is correct:

```xml
<body>
  <p>This is a paragraph</p>
</body>
```

Important Compatibility Note

- To make your XHTML compatible with today’s browsers, you should add an extra space before the “/” symbol like this: `<br />`, and this: `<hr />`

All XHTML Elements Must Be Closed

- Non-empty elements must have an end tag.
  - This is wrong:
    ```xml
    <p>This is a paragraph
    <p>This is another paragraph
    ```
  - This is correct:
    ```xml
    <p>This is a paragraph</p>
    <p>This is another paragraph</p>
    ```

Some More XHTML Syntax Rules

- Attribute names must be in lower case
- Attribute values must be quoted
- Attribute minimization is forbidden
- The id attribute replaces the name attribute
- The XHTML DTD defines mandatory elements

Empty Elements Must Also Be Closed

- Empty elements must either have an end tag or the start tag must end with `/>`
  - This is wrong:
    ```xml
    This is a break<br>
    Here comes a horizontal rule:<hr>
    Here's an image `<img src="happy.gif" alt="Happy face">
    ```
  - This is correct:
    ```xml
    This is a break<br/>
    Here comes a horizontal rule:<hr/>
    Here's an image `<img src="happy.gif" alt="Happy face">
    ```

Attribute Names Must Be In Lower Case

- This is wrong:
  ```xml
  <table WIDTH="100%">
  ```
- This is correct:
  ```xml
  <table width="100%">
  ```
Attribute Values Must Be Quoted

► This is wrong:
[table width="100%"

► This is correct:
[table width="100%"/>

Attribute Minimization Is Forbidden

► This is wrong:
<input checked=
<input readonly=
<input disabled=
<option selected=
<frame noresize=

► This is correct:
<input checked="checked"/>
<input readonly="readonly"/>
<input disabled="disabled"/>
<option selected="selected"/>
<frame noresize="noresize"/>

The id Attribute Replaces The name Attribute

► HTML 4.01 defines a name attribute for the elements a, applet, frame, iframe, img, and map. In XHTML the name attribute is deprecated. Use id instead.

► This is wrong:
<img src="picture.gif" name="picture1"/>

► This is correct:
<img src="picture.gif" id="picture1"/>

To interoperate with older browsers for a while, you should use both name and id, with identical attribute values, like this:
<img src="picture.gif" id="picture1" name="picture1"/>

The Lang Attribute

► The lang attribute applies to almost every XHTML element. It specifies the language of the content within an element.

► If you use the lang attribute in an element, you must add the xml:lang attribute, like this:
<div lang="no" xml:lang="no">Heia Norge!</div>

Mandatory XHTML Elements

► All XHTML documents must have a DOCTYPE declaration.

► The html, head and body elements must be present, and the title must be present inside the head element.

► This is a minimum XHTML document template:
<!DOCTYPE Doctype goes here>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Title goes here</title>
</head>
<body>
Body text goes here
</body>
</html>
Mandatory XHTML Elements

► The DOCTYPE declaration is not a part of the XHTML document itself. It is not an XHTML element, and it should not have a closing tag.

► The xmlns attribute inside the <html> tag is required in XHTML. However, the validator on w3.org does not complain when this attribute is missing in an XHTML document.

► This is because "xmlns=http://www.w3.org/1999/xhtml" is a fixed value and will be added to the <html> tag even if you do not include it.

The <!DOCTYPE> Is Mandatory

► An XHTML document consists of three main parts:
  – the DOCTYPE
  – the Head
  – the Body

► The basic document structure is:
  <!DOCTYPE ...
  <html>
  <head>
  <title>... </title>
  </head>
  <body> ... </body>
  </html>

► The DOCTYPE declaration should always be the first line in an XHTML document.

An XHTML Example

► This is a simple (minimal) XHTML document:
  <!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
  <html>
  <head>
  <title>simple document</title>
  </head>
  <body>
  <p>a simple paragraph</p>
  </body>
  </html>

The 3 Document Type Definitions

► DTD specifies the syntax of a web page in SGML.

► DTD is used by SGML applications, such as HTML, to specify rules that apply to the markup of documents of a particular type, including a set of element and entity declarations.

► XHTML is specified in an SGML document type definition or ‘DTD’.

► An XHTML DTD describes in precise, computer-readable language, the allowed syntax and grammar of XHTML markup.

An XHTML Example

► The DOCTYPE declaration defines the document type:
  <!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

► The rest of the document looks like HTML:
  <html>
  <head>
  <title>simple document</title>
  <head>
  <body>
  <p>a simple paragraph</p>
  </body>
  </html>

The 3 Document Type Definitions

► XHTML 1.0 specifies three XML document types that correspond to three DTDs:
  – Strict
  – Transitional
  – Frameset
XHTML 1.0 Strict

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

► Use this when you want really clean markup, free of 
presentational clutter.
► Use this together with Cascading Style Sheets.
```

XHTML 1.0 Transitional

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

► Use this
  – when you need to take advantage of HTML’s 
presentational features
  – when you want to support browsers that don’t 
understand Cascading Style Sheets.
```

XHTML 1.0 Frameset

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

► Use this when you want to use HTML Frames to partition 
the browser window into two or more frames.
```

Why XHTML Modularization?

► XHTML is a simple but large language, containing most 
of the functionality a web developer will need.
► For some purposes XHTML is too large and complex, and 
for other purposes it is much too simple.
► By splitting XHTML into modules, the W3C (World 
Wide web Consortium) has created small and well-defined 
sets of XHTML elements that can be used separately for 
simple devices as well as combined with other XML 
standards into larger and more complex applications.

Why XHTML Modularization?

► With modular XHTML, product and application designers 
can:
  – Choose the elements to be supported by a device using 
standard XHTML building blocks.
  – Add extensions to XHTML, using XML, without 
breaking the XHTML standard.
  – Simplify XHTML for devices like hand held computers, 
mobile phones, TV, and home appliances.
  – Extend XHTML for complex applications by adding 
new XML functionality (like MathML, SVG, Voice and 
Multimedia).
  – Define XHTML profiles like XHTML Basic (a subset of 
XHTML for mobile devices).

XHTML Modules

<table>
<thead>
<tr>
<th>Module name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applet Module</td>
<td>Defines the deprecated applet element.</td>
</tr>
<tr>
<td>Basic Module</td>
<td>Defines the basic element.</td>
</tr>
<tr>
<td>Basic Forms Module</td>
<td>Defines the basic forms elements.</td>
</tr>
<tr>
<td>Basic Tables Module</td>
<td>Defines the basic table elements.</td>
</tr>
<tr>
<td>Bi-directional Text Module</td>
<td>Defines the bdi element.</td>
</tr>
<tr>
<td>Client Image Map Module</td>
<td>Defines browser side image map elements.</td>
</tr>
<tr>
<td>Edit Module</td>
<td>Defines the editing elements del and ins.</td>
</tr>
<tr>
<td>Forms Module</td>
<td>Defines all elements used in forms.</td>
</tr>
<tr>
<td>Frames Module</td>
<td>Defines the frames elements.</td>
</tr>
<tr>
<td>Hypertext Module</td>
<td>Defines the a element.</td>
</tr>
<tr>
<td>Iframe Module</td>
<td>Defines the iframe element.</td>
</tr>
<tr>
<td>Image Module</td>
<td>Defines the img element.</td>
</tr>
</tbody>
</table>
XHTML Standard Attributes
► XHTML tags can have attributes.
► The special attributes for each tag are listed under each tag description.
► The attributes listed here are the core and language attributes that are standard for all tags (with a few exceptions).

Language Attributes
► Not valid in base, br, frame, frameset, hr, iframe, param, and script elements.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir</td>
<td>ltr</td>
<td>rtl</td>
</tr>
<tr>
<td>lang</td>
<td>language code</td>
<td>Sets the language code</td>
</tr>
</tbody>
</table>

Keyboard Attributes
► Not valid in base, head, html, meta, param, script, style, and title elements.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accesskey</td>
<td>character</td>
<td>Sets a keyboard shortcut to access an element</td>
</tr>
<tr>
<td>tabindex</td>
<td>number</td>
<td>Sets the tab order of an element</td>
</tr>
</tbody>
</table>

Core Attributes
► Not valid in base, head, html, meta, param, script, style, and title elements.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>class_rule or style_rule</td>
<td>The class of the element</td>
</tr>
<tr>
<td>id</td>
<td>id_name</td>
<td>A unique id for the element</td>
</tr>
<tr>
<td>style</td>
<td>style_definition</td>
<td>An inline style definition</td>
</tr>
<tr>
<td>title</td>
<td>tooltip_text</td>
<td>A text to display in a tool tip</td>
</tr>
</tbody>
</table>

XHTML Event Attributes
► New to HTML 4.0 was the ability to let HTML events trigger actions in the browser, like starting a JavaScript when a user clicks on an HTML element. Below is a list of attributes that can be inserted into HTML tags to define event actions.
► We will learn more about programming with these events in studying JavaScript and DHTML.
Window Events

- Only valid in body and frameset elements

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onload</td>
<td>script</td>
<td>Script to be run when a document loads</td>
</tr>
<tr>
<td>onunload</td>
<td>script</td>
<td>Script to be run when a document unloads</td>
</tr>
</tbody>
</table>

Form Element Events

- Only valid in form elements.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onchange</td>
<td>script</td>
<td>Script to be run when the element changes</td>
</tr>
<tr>
<td>onsubmit</td>
<td>script</td>
<td>Script to be run when the form is submitted</td>
</tr>
<tr>
<td>onreset</td>
<td>script</td>
<td>Script to be run when the form is reset</td>
</tr>
<tr>
<td>onselect</td>
<td>script</td>
<td>Script to be run when the element is selected</td>
</tr>
<tr>
<td>onblur</td>
<td>script</td>
<td>Script to be run when the element loses focus</td>
</tr>
<tr>
<td>onfocus</td>
<td>script</td>
<td>Script to be run when the element gets focus</td>
</tr>
</tbody>
</table>

Mouse Events

- Not valid in base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, and title elements.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onclick</td>
<td>script</td>
<td>What to do on a mouse click</td>
</tr>
<tr>
<td>ondblclick</td>
<td>script</td>
<td>What to do on a mouse doubleclick</td>
</tr>
<tr>
<td>onmousedown</td>
<td>script</td>
<td>What to do when mouse button is pressed</td>
</tr>
<tr>
<td>onmouseup</td>
<td>script</td>
<td>What to do when mouse button is released</td>
</tr>
<tr>
<td>onmousemove</td>
<td>script</td>
<td>What to do when mouse pointer moves</td>
</tr>
<tr>
<td>onmouseover</td>
<td>script</td>
<td>What to do when mouse pointer moves over an element</td>
</tr>
<tr>
<td>onmouseout</td>
<td>script</td>
<td>What to do when mouse pointer moves out of an element</td>
</tr>
</tbody>
</table>

Keyboard Events

- Not valid in base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, and title elements.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onkeydown</td>
<td>script</td>
<td>What to do when key is pressed</td>
</tr>
<tr>
<td>onkeypress</td>
<td>script</td>
<td>What to do when key is pressed and released</td>
</tr>
<tr>
<td>onkeyup</td>
<td>script</td>
<td>What to do when key is released</td>
</tr>
</tbody>
</table>

Quiz

Question # 1

- What does XHTML stand for?
  a. EXtensible HyperText Markup Language
  b. EXtreme HyperText Markup Language
  c. EXtensible HyperText Markup Language
  d. EXtra Hyperlinks and Text Markup Language
Question # 2
► XHTML is a Web standard
  a. True
  b. False

Question # 3
► XML and HTML will be replaced by XHTML
  a. False
  b. True

Question # 4
► HTML will be replaced by XHTML
  a. False
  b. True

Question # 5
► What is the correct XHTML for a paragraph?
  a. </p><p>
  b. <P></P>
  c. <P></p>
  d. </p></p>

Question # 6
► What is a correct XHTML tag for a line break?
  a. <br>
  b. <br />
  c. <break/>

Question # 7
► What is the correct XHTML for an attribute and its value?
  a. WIDTH="80"
  b. width=80
  c. WIDTH=80
  d. width="80"
Question # 8
► All elements in XHTML must be closed
a. True
b. False

Question # 9
► Is this correct XHTML?
  <ul>
    <li>Coffee</li>
    <li>Tea</li>
    <li>Black tea</li>
    <li>Green tea</li>
    <li>Milk</li>
  </ul>
a. No
b. Yes

Question # 10
► The DOCTYPE declaration has no closing tag
a. True
b. False

Question # 11
► Which elements are mandatory in an XHTML document?
  a. doctype, html, head, body, and title
  b. doctype, html and body
  c. doctype, html, head, and body

Question # 12
► XHTML documents must be "well-formed"
  a. True
  b. False

Question # 13
► What XHTML code is "well-formed"?
  a. <p>A <b><i>short</i></b> paragraph</p>
  b. <p>A <b><i>short</i></b> paragraph</p>
  c. <p>A <b><i>short</i></b> paragraph</p>
Question # 14
Which of the following is the right use of the lang attribute?
- a. `<div language="en">Hello World!</div>`
- b. `<div xml:language="en">Hello World!</div>`
- c. `<div lang="en" xml:lang="en">Hello World!</div>`

Question # 17
What are the different DTDs in XHTML?
- a. Strict, Transitional, Frameset
- b. Strict, Transitional, Loose
- c. Strict, Transitional, Loose, Frameset

Question # 15
Which attribute replaces the name attribute?
- a. None
- b. The src attribute
- c. The id attribute

Question # 18
Do all XHTML documents require a doctype?
- a. Yes
- b. No

Question # 16
Is attribute minimization allowed in XHTML?
- a. Yes
- b. No

Question # 19
What is the most common XHTML DTD?
- a. Loose
- b. Normal
- c. Frameset
- d. Transitional
Question # 20

- All XHTML tags and attributes must be in lower case
  a. False
  b. True

XSL

- XSL = eXtensible Stylesheet Language
  - the Stylesheet Language for XML
    - implemented in XML
    - includes a transformation language XSLT
    - includes a formatting language XSLFO – XSL Formatting Objects
    - defines how a source document should be transformed to provide an output typically in XML or HTML
    - XPath is a language to define XML parts or patterns

XSLT

- XSLT - XSL Transformations
- XSLT Version 1.0 is a W3C Recommendation 16 November 1999
- XSLT is designed for use as part of
  - XSL - A transformation expressed in
    - XSLT is called a stylesheet.
  - XSLT processors must use the XML namespaces mechanism to recognize elements
- XSLT is incredibly useful
  - transforming data into a viewable format in a browser/phone/PDA (Presentation Oriented Publishing (POP)) - data to presentation
  - transforming business data between content models (Message Oriented Middleware (MOM)) – Transmitting data between applications

Content

- Define XSL and XSLT
- Describe the main components of an XSL document
- Write a Java program that uses the TrAX and Xalan
- Transform an XML document using the Xalan XSLT processor and a stylesheet
XSLT in POP

► XML document separates content from presentation
► Transformations can be used to style (render, present) XML documents
► A common styling technique presents XML in HTML format

XSLT in POP (Con’t)

XSLT in MOM

► Important for eCommerce, B2B/EDI, and dynamic content generation
  – Different content model
  – Different structural relationship
  – Different vocabularies

XSLT in MOM (Con’t)

Transformation process

► XSLT uses XPath to define parts of the source document that match one or more predefined templates.
► When a match is found, XSLT will transform the matching part of the source document into the result document.
► Transforming XML document into
  – Another XML document
    • XML
    • XHTML
    • WML
  – HTML document

XSLT Data Transformation
XSLT Operational Model

XSLT Processor

- Piece of software
  - Reads an XSLT stylesheet and input XML document
  - Converts the input document into an output document
  - According to the instruction given in the stylesheet
- Called stylesheet processor sometimes

Examples of XSLT Processor

- Built-in within a browser
  - IE 6
- Built-in within web or application server
  - Apache Cocoon
- Standalone
  - Michael Kay’s SAXON
  - Apache.org’s Xalan

XPath

- Used by XSLT (and by other XML technologies such as XPointer) for referencing elements and attributes internal to an XML document
- Defines expression language (pattern) for referencing
- Supports a tree structure expression
  - 7th child element of the third person element

XPath (Con’t)

- XPath expression results in a node set
  - A node set of “person” elements under “people” element
- Various functions can be used on node sets, including:
  - `not()` – eliminate a specific node
  - `position()` – return the position within a node set
  - `count()` – returns the number of nodes in a node set

XML Example Document

```
<?xml version="1.0"?>
<people>
  <person born="1912" died="1954">
    <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
    </name>
    <profession>computer scientist</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
  </person>
  <person born="1918" died="1988">
    <name>
      <first_name>Richard</first_name>
      <middle_initial>M</middle_initial>
      <last_name>Feynman</last_name>
    </name>
    <profession>physicist</profession>
    <hobby>Playing the bongoes</hobby>
  </person>
</people>
```
XSLT Stylesheet

- Genuine XML document
- Root element typically is
  - stylesheet or transform
  - Both are defined in standard XSLT namespace
    - http://www.w3.org/XSL/Transform
    - xsl as customary prefix
  - XSLT processor should understand both

Minimal but Complete XSLT Stylesheet

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="*">
    <xsl:copy>
      <xsl:apply-templates select="*"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Result of XSLT Processing

```
<?xml version="1.0" encoding="utf-8"?>
Alan
Turing
computer scientist
mathematician
cryptographer
Richard
M
Feynman
physicist
Playing the bongoes
```

Apache Xalan

- Implements XSLT 1.0 and XPath 1.0
- Can be run from both the command line and within application code
- Support scripting extension
- Command line syntax:
  - `java org.apache.xalan.xslt.Process`
    - `--IN <input document>`
    - `--XSL <stylesheet>`
    - `--OUT <output document>`

xml-stylesheet Processing Instruction

- Included as part of XML document
- Tells XML-ware browser where to find associated stylesheet

```xml
<people>
```

Result of XSLT Processing

```
Alan
Turing
computer scientist
mathematician
cryptographer
Richard
M
Feynman
physicist
Playing the bongoes
```
Templates

- Controls which output is created from which input
- `xsl:template` element form
- `match` attribute contains an XPath expression
  - XPath expression identifies input node set it matches
- For each node in the node set, the template contents (things between `xsl:template` tags) are instantiated and inserted into the output tree

Very Simple XSLT Stylesheet # 1

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="people">
    Hello World!
  </xsl:template>
</xsl:stylesheet>
```

Result

```xml
<?xml version="1.0" encoding="UTF-8"?>
Hello World!
```

Very Simple XSLT Stylesheet # 2

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="people">
    Hello World!
  </xsl:template>
</xsl:stylesheet>
```

Result

```xml
<?xml version="1.0" encoding="UTF-8"?>
Hello World!
```
Very Simple XSLT Stylesheet # 3

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="person">
        A Person
    </xsl:template>
</xsl:stylesheet>
```

XML Example Document

```xml
<people>
    <person born="1912" died="1954">
        <name>
            <first_name>Alan</first_name>
            <last_name>Turing</last_name>
        </name>
        <profession>computer scientist</profession>
        <profession>mathematician</profession>
        <profession>cryptographer</profession>
    </person>
    <person born="1918" died="1988">
        <name>
            <first_name>Richard</first_name>
            <middle_initial>M</middle_initial>
            <last_name>Feynman</last_name>
        </name>
        <profession>physicist</profession>
        <hobby>Playing the bongos</hobby>
    </person>
</people>
```

Result

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

Very Simple XSLT Stylesheet # 4

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="person">
        A Person
    </xsl:template>
</xsl:stylesheet>
```

XML Example Document

```xml
<people>
    <person born="1912" died="1954">
        <name>
            <first_name>Alan</first_name>
            <last_name>Turing</last_name>
        </name>
        <profession>computer scientist</profession>
        <profession>mathematician</profession>
        <profession>cryptographer</profession>
    </person>
    <person born="1918" died="1988">
        <name>
            <first_name>Richard</first_name>
            <middle_initial>M</middle_initial>
            <last_name>Feynman</last_name>
        </name>
        <profession>physicist</profession>
        <hobby>Playing the bongos</hobby>
    </person>
</people>
```

Result

```xml
<?xml version="1.0" encoding="UTF-8"?>
A Person
```

Some text here under people element!

```xml
<some-tag>
some information inside some-tag
</some-tag>
```

```xml
A Person
```

```xml
Some text here under people element!
```

```xml
<some-tag>
some information inside some-tag
</some-tag>
```
Very Simple XSLT Stylesheet # 5

<?xml version="1.0"?><xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="person">
<p>A Person</p>
</xsl:template>
</xsl:stylesheet>

Literal data characters - text copied from the stylesheet into the output document

Example Stylesheet

Extract names of all the people

<?xml version="1.0"?><xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="person">
<p><xsl:value-of select="name"/></p>
</xsl:template>
</xsl:stylesheet>

XML Example Document

<person born="1912" died="1954">
  name:
  <first_name>Alan</first_name>
  <last_name>Turing</last_name>
  profession: computer scientist
</person>
<person born="1918" died="1988">
  name:
  <first_name>Richard</first_name>
  <middle_initial>M</middle_initial>
  <last_name>Feynman</last_name>
  profession: physicist
</person>

Result

<?xml version="1.0" encoding="UTF-8"?>
<p>Alan Turing</p>
<p>Richard M Feynman</p>

Template content contains tags and character data

They have to be well-formed XML

<xsl:value-of> Element

Extracts the string value of an element or an attribute and writes it to output
- text content of the element after all the tags have been removed and entity references are resolved

select attribute containing XPath expression identifies an element or an attribute
- It could be a node set, in which case, the string value of first node is taken

Result

<?xml version="1.0" encoding="UTF-8"?>
<p>Alan Turing</p>
<p>Richard M Feynman</p>
xsl:apply-templates

- XSLT processor reads (traverses) the input XML document sequentially from top to bottom
- Templates are activated in the order they match elements encountered
  - Template for a parent will be activated before the children

xsl:apply-templates (Con’t)

- The order of the traversal can be changed by apply-templates
  - It can specify which element or elements should be processed next
  - It can specify an element or elements should be processed in the middle of processing another element
  - It can prevent particular elements from being processed

xsl:apply-templates Example

- I would like the output to look like as following
  - Last name then first name
  - Only name not profession nor hobby

```xml
<?xml version="1.0" encoding="utf-8"?>
<Turing>
  <Alan>
  <Feyman>
  <Richard>
</xsl:apply-templates Example

Result

```xml
<?xml version="1.0" encoding="utf-8"?>
<Turing>
  <Alan>
  <Feyman>
  <Richard>
  <computer scientist
  <mathematician
  <cryptographer
  <Playing the bongoes
<?xml version="1.0"?>
xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="name">
    <xsl:value-of select="last_name"/>
    ,
    <xsl:value-of select="first_name"/>
  </xsl:template>
</xsl:stylesheet>

<!-- Apply templates only to name children -->
<xsl:template match="person">
  <xsl:apply-templates select="name"/>
</xsl:template>

<xsl:template match="people">
  <html>
    <head><title>Famous Scientists</title></head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="person">
  <xsl:apply-templates select="name"/>
</xsl:template>

<xsl:template match="name">
  <p><xsl:value-of select="last_name"/>
  ,
  <xsl:value-of select="first_name"/></p>
</xsl:template>

<?xml version="1.0" encoding="UTF-8"?>
Alan
Turing

Richard
M
Feynman
**Result**

```html
<html>
<head>
  <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Famous Scientists</title>
</head>
<body>
  <p>Alan Turing</p>
  <p>Richard Feynman</p>
</body>
</html>
```

**Attributes**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="people">
    <html>
      <head><title>Famous Scientists</title></head>
      <body>
        <dl>
          <xsl:apply-templates/>
        </dl>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

**Modes**

- Same input content needs to appear multiple times in the output document formatted according to different template:
  - Titles of chapters
  - In the chapters themselves
- `mode` attribute
  - `xsl:template`
  - `xsl:apply-templates`

**Example with mode attribute**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="people">
    <html>
      <head><title>Famous Scientists</title></head>
      <body>
        <ul><xsl:apply-templates select="person" mode="toc"/></ul>
        <xsl:apply-templates select="person"/>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

**Attributes**

- Default rule does not apply
  - `apply-templates` has to be present in order to output values of attributes
Result

<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Famous Scientists</title>
</head>
<body>
<ul>
<li>Turing, Alan</li>
<li>Feynman, Richard</li>
</ul>
<p>
Richard M Feynman
physicist
Playing the bongoes
</p>
</body>
</html>

Filtering

► So far we either process all the elements relative to a node or one element
► We need a way to filter out elements as well
► This is done with an XPath control structure

Filtering Example

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="person">
<xsl:apply-templates select="*[not(self::hobby)]"/>
</xsl:template>
</xsl:stylesheet>
► The self keyword is needed to inform the XSLT processor that the node following is a child of the current one

Example of xsl:for-each

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="people">
<xsl:for-each select="person">
  <xsl:value-of select="name"/>
  <xsl:value-of select="@born"/>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
We can test content for certain values with XSL:

\[<\text{xsl:if test} \text{criteria}\rangle\langle/\text{xsl:if}\rangle\]

The test attribute is required and will either be true or false.

Example of xsl:if

```xml
<?xml version="1.0" standalone="yes" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
  <xsl:template match="people">
    <xsl:for-each select="person">
      <xsl:value-of select="name"/>
      <xsl:choose>
        <xsl:when test="@born='1912'">
          Died in <xsl:value-of select="@died"/>
        </xsl:when>
        <xsl:otherwise>
          Did not die in 1912
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

We can also select content using:

\[<\text{xsl:choose}\rangle<\text{xsl:when test} \text{criteria}\rangle<\text{xsl:otherwise}\rangle<\text{xsl:choose}\rangle\]

The test attribute works in the same fashion as xsl:if.

```xml
<?xml version="1.0" standalone="yes" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
  <xsl:template match="people">
    <xsl:for-each select="person">
      <xsl:value-of select="name"/>
      <xsl:choose>
        <xsl:when test="@born='1912'">
          Died in <xsl:value-of select="@died"/>
        </xsl:when>
        <xsl:otherwise>
          Did not die in 1912
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```
Alan Turing
Died in 1954

Richard M
Feynman
Did not die in 1912

Example of xsl:sort (“Ascending”)

Example of xsl:sort (“Descending”)

XSLT provides a nice way to sort documents by element contents
The construct to use is:

Sorting can only be done in the following constructs:
- <xsl:apply-templates…/>
- <xsl:for-each …/>
**xsl:copy**

- Used for creating an XML Document
- The copying is done using this construct:
  ```xml
  <xsl:copy></xsl:copy>
  ```
- We will also specify to the processor that our output should be XML instead of HTML.
  ```xml
  <xsl:output method="xml"/>
  ```

**Example of xsl:copy**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml"/>
    <xsl:template match="people">
        <l>
            <xsl:copy>
                <xsl:apply-templates>
                    <xsl:sort select="name"/>
                </xsl:apply-templates>
            </xsl:copy>
        </l>
    </xsl:template>
</xsl:stylesheet>
```

**Result**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<people><person>
    <name>
        <first_name>Richard</first_name>
        <middle_initial>M</middle_initial>
        <last_name>Feynman</last_name>
    </name>
    <profession>physicist</profession>
    <hobby>Playing the bongos</hobby>
</person><person>
    <name>
        <first_name>Alan</first_name>
        <last_name>Turing</last_name>
    </name>
    <profession>computer scientist</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
</person>
</people>
```

**Programming API**

- **Input (Source tree)**
  - File, Character stream, Byte stream
  - DOM
  - SAX input stream
- **Output (Result tree)**
  - File, Character stream, Byte stream
  - DOM
  - SAX events
Programming API using Xalan

//Have XSLTProcessorFactory obtain a interface to a
// new XSLTProcessor object.
XSLTProcessor processor =
XSLTProcessorFactory.getProcessor();

// Have the XSLTProcessor processor object transform
// "foo.xml" to System.out, using the XSLT instructions
// found in "foo.xsl".
    processor.process(new XSLTInputSource("foo.xml"),
            new XSLTInputSource("foo.xsl"),
            new XSLTResultTarget(System.out));

Programming API using JAXP 1.1

TransformerFactory tf = TransformerFactory.newInstance();
Transformer transformer =
    tf.newTransformer(new StreamSource("foo.xsl");
    transformer.transform(
        new StreamSource("foo.xml"),
        new StreamSource("bar.xml"));

XSLT vs. Programming

- Programming is useful when you do more than
  transformation
- Examples
  - Interpreting certain elements as database queries
  - Inserting the query results into output document
  - asking users questions in the middle of transformation

What is CSS?

- CSS stands for Cascading Style Sheets
- Styles define how to display HTML elements
- Styles are normally stored in Style Sheets
- Styles were added to HTML 4.0 to solve a problem
- External Style Sheets can save you a lot of work
- External Style Sheets are stored in CSS files
- Multiple style definitions will cascade into one
**Styles Solve a Common Problem**
- HTML tags were originally designed to define the content of a document.
- They were supposed to say "This is a header", "This is a paragraph", "This is a table", by using tags like <h1>, <p>, <table>, and so on.
- The layout of the document was supposed to be taken care of by the browser, without using any formatting tags.
- As new HTML tags and attributes (like the <font> tag and the color attribute) are added to the original HTML specification, it became more difficult to create Web sites where the content of HTML documents was clearly separated from the document's presentation layout.

**Style Sheets Can Save a Lot of Work**
- As a Web developer you can define a style for each HTML element and apply it to as many Web pages as you want.
- To make a global change, simply change the style, and all elements in the Web are updated automatically.

**Styles Solve a Common Problem**
- To solve this problem, the World Wide Web Consortium (W3C) - the non profit, standard setting consortium, responsible for standardizing HTML - created STYLES in addition to HTML 4.0.

**Multiple Styles Will Cascade Into One**
- Style sheets allow style information to be specified in many ways.
- Styles can be specified inside a single HTML element, inside the <head> element of an HTML page, or in an external CSS file.
- Even multiple external style sheets can be referenced inside a single HTML document.

**Style Sheets Can Save a Lot of Work**
- Styles sheets define HOW HTML elements are to be displayed, just like the font tag and the color attribute in HTML 3.2.
- Styles are normally saved in external .css files.
- External style sheets enable you to change the appearance and layout of all the pages in your Web, just by editing one single CSS document!
- CSS is a breakthrough in Web design because it allows developers to control the style and layout of multiple Web pages all at once.

**Cascading Order**
What style will be used when there is more than one style specified for an HTML element?
- All the styles will "cascade" into a new "virtual" style sheet by the following rules, where number four has the highest priority:
  1. Browser default
  2. External style sheet
  3. Internal style sheet (inside the <head> tag)
  4. Inline style (inside an HTML element)
- Inline style (inside an HTML element) has the highest priority, which means that it will override a style declared inside the <head> tag, in an external style sheet, or in a browser.
Top 10 Reasons for Using CSS

1. Build from the ground up to replace traditional Web design methods
2. Faster download times
3. Shorter development time
4. Greater control over the typography in a Web page
5. It's easy to write
6. Improvements in accessibility
7. Print designs as well as Web page designs
8. Better control over the placement of elements in Web page
9. The design of Web pages is separated from the content
10. Better search engine rankings

A Brief History of HTML and CSS

- Separating Style from Structure
  - Mixing display instructions and structural information:
    - Adds to complexity of code
    - Inefficient mechanism for handling display characteristics of multi-page Web sites
    - Limits cross-platform compatibility of content
    - Limits diversity of web devices

Status of CSS

- **CSS, Level 1 (1996)**
  - Concerned with applying simple styles to HTML elements
  - http://www.w3.org/TR/REC-CSS1
- **CSS, Level 2 (1998)**
  - Supports media-specific style sheets (visual browsers, aural devices, printers, Braille devices)
  - http://www.w3.org/TR/REC-CSS2
- **CSS, Level 3 (draft 2001)**
  - Focused on modularization of the CSS specification
  - http://www.w3.org/TR/css3-roadmap/
What are they?

Types of CSS

- External
- Embedded
- Imported
- Inline

External

- Connection made via the LINK tag
- Use the optional TYPE attribute to specify a media type
  - type/css

Internal/Embedded

- Style characteristics are embedded in the HEAD section of the webpage
- Perhaps best used when a single page requires a unique style sheet
Imported

► Allows for using style sheets from other sources
► Must be included at the beginning of the style sheet using the @import statement
► Other CSS rules can be included

Inserting a CSS

► Internal/embedded sheet

```html
<head>
<style type="text/css">
hr { color: navy}
body {margin-left: 20px}
</style>
</head>
```

Inline

► Least flexible
► Requires each element to be tagged if you want them to appear differently
► Loses the advantage of using CSS

Inserting a CSS

► Internal/embedded sheet for older browsers

```html
<head>
<!-- hr { color: navy}
body {margin-left: 20px} -->
</style>
</head>
```

Inserting a CSS

► External sheet

```html
<head>
<link rel="stylesheet" type="text/css"
href="mystyle.css" />
</head>
```

Inserting a CSS

► Inline

```html
<p style="color: yellow; font-family: verdana">This is a paragraph</p>
```
Using multiple sheets

- You can use multiple sheets to define the style of your document
- Internal styles will override external styles, if they are duplicated

Using multiple sheets

h3 {color: red; text-align: right; font-size: 8pt} *(external CSS)*

h3 {text-align: center; font-size: 20pt} *(internal CSS)*

*will yield*

Using multiple sheets

h3 {color: red; text-align: right; font-size: 8pt} *(external CSS)*

h3 {text-align: center; font-size: 20pt} *(internal CSS)*

*will yield*

Understanding the Cascade

- Cascading
  - Determining rule weight by specificity
    - Rules with more specific selectors take precedence over rules with less specific selectors
  - Determining rule weight by order
    - Based on order of rule within style sheet
      - Those listed later take precedence over those listed earlier in the style sheet

Understanding the Cascade

- Inheritance
  - Based on hierarchical structure of documents
    - CSS rules inherit from parent elements to child elements:
      - thus <LI> elements will inherit style rules from <UL> elements unless a style rule is specifically set for the <LI> element
Basic CSS Syntax

Selector {property: value}

- **property**: the attribute of the selector that you wish to change

body {color: black}

Three parts:
- selector
- property
- value

}; declaration

If the value has multiple words, put the value in quotes

p {font-family: "sans serif"}
Basic CSS Syntax

You can specify multiple properties to a single selector. Properties must be separated by a semicolon.

```
P { text-align: left; color: red }  
```

Selectors can be descendants

```
P B { color: yellow }  
```

In this example, only those `<B>` elements within a `<P>` element would be yellow

```
<p><b>This would be yellow</b></p>
<p>This would not be yellow</p>  
```

To make properties more readable, put each on a separate line.

```
p { text-align: center;  
       color: navy;  
       font-family: arial }  
```

CSS Syntax - class

The class selector allows you to create different styles for the same HTML element.

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

Selectors can be grouped so that a common property can be specified

```
h1,h2,h3,h4,h5,h6 { color: yellow }  
```

```
<h1> This is a level 1 heading </h1>  
<h2> This is a level 2 heading </h2>  
```

CSS Syntax - class

```
p.right { text-align: right }  
```

```
<p class="right">This paragraph will be right aligned.</p>  
```

Note: the class name must be in quotes inside the opening tag
CSS Syntax - class

This is improper use of the class selector:

```html
<p class="right" class="center">
  This paragraph will be right aligned.
</p>
```

Only one class selector can be included inside the tag

CSS Syntax - class

You can also create a class selector free of a tag name if you want all tags that have that class to be formatted the same.

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

Any tag with a “center” class will be aligned center

CSS Syntax - id

While the class selector can apply to several different elements, the id selector can only apply to one, unique element.

```css
p#para1 { text-align: center; color: green; }
```

```html
<p id="para1">
  This text would be centered and green
</p>
```

CSS Syntax - class

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

```html
<h1 class="center">
  This heading will be centered
</h1>
```

```html
<p class="center">
  So will this text
</p>
```

CSS Syntax - comment

You can insert comments to help you describe the particular style

```css
/* This is a comment */
P { color: red; /* This is another comment */
  font-family: verdana; }
```
CSS syntax - <div>

► <DIV> can be used with the CLASS attribute to create customized block-level elements
– Declare it in the style rule:
  • DIV.introduction {font-size: 14pt; margin: 24 pt;}
– Apply the style rule in the document:
  • <DIV CLASS="introduction">This is the introduction to the document</DIV>

CSS syntax - <span>

► <SPAN> can be used with the CLASS attribute to create customized inline elements
– Declare it in the style rule:
  • SPAN.logo {color: white; background-color: black;}
– Apply the style rule in the document:
  • <P>Welcome to the <SPAN CLASS="logo">Wonder Software</SPAN> Web site</P>

Background properties

► Define the background effects of an element
► Effects include color, using an image for a background, repeating an image and positioning an image

Background properties

► Basic syntax
  – background
  – background-color
  – background-image
  – background-repeat
  – background-attachment
  – background-position

Background Properties

► All attributes can be set in a single declaration:

background: #000000 url('psumark.gif') no-repeat fixed center
Background properties

Setting the body background (internal CSS)

body { background: #000000 url('psumark.gif') no-repeat fixed center }

Setting the body background (external CSS)

body { background: #000000 url('psumark.gif') no-repeat fixed center }

Elements can also be set separately

body

{ background-image: url(psumark.gif),
  background-color: navy }

Text Properties

Controls the appearance of text in the web page

Commonly used attributes
- color
- direction
- text-align
- text-decoration
- text-indent
Text properties

► **color**
  - sets the color of the text
  - color can be represented by the color name (red), an rgb value (rgb(255,0,0)), or by a hexadecimal number (#ff0000)

► Syntax
  - body { color: #ff0000}

► **direction**
  - sets the direction of the text
  - can be set as left to right (ltr) or right to left (rtl)

► Syntax
  - body { direction: rtl}

► **text-align**
  - aligns the text in an element
  - possible values are left, right, center and justify

► Syntax
  - p { text-align: center}

Font Properties

► **text-decoration**
  - adds certain decoration elements to the text
  - possible values are none, underline, overline, line-through and blink

► Syntax
  - p { text-decoration: underline}

► **text-indent**
  - indents the first line of text inside an element
  - possible values are length (defines a fixed value) and % (defines a % of the parent element)

► Syntax
  - p { text-indent: 20px}
Font properties
► Define the look of the font in text areas
► One of the broader sets of properties in CSS

Font properties
► font
► font-style
► font-variant
► font-weight
► font-size/line-height
► font-family

Font properties
► font-variant
► normal
► font displays as is
► small-caps
► font displays in all capitals, with lower case letters in smaller size

Syntax: body {font-variant: small-caps}

Font properties
► font-weight
► normal
► bold
► bolder
► lighter
► weighted values

Syntax: body {font-weight: bold}

Font properties
► font-style
► normal
► italic
► oblique

Syntax: body {font-style: italic}

Weighted values
► range from 100 – 900
► 400 is the same as normal weight
► 700 is the same as bold weight
Font properties

- **font-size**
  - XX-small to xx-large
  - Smaller
    - Smaller than parent
  - Larger
    - Larger than parent
  - %
    - % of the parent

Syntax: body {font-size: 20px}
{font-size: x-large}
{font-size: 125%}

Border properties

- Allows you to specify the style, color and width of an element’s border
- Many different properties can be applied

Font properties

- **font-family**
  - Family-name
    - "times", "arial", "courier", "verdana"
  - Generic-family
    - "serif", "sans-serif", "monospace"

Syntax: body {font-family: verdana, sans-serif}

Border Properties

Margin Properties
Margin properties
► Define the space around elements
► You can use negative values to overlap content
► Margins can be set independently or collectively
► Can be set to auto, a fixed length or a % of the total height of the document

Margin properties
► Can be set in one declaration
► Think clock face
  – top, right, bottom, left

h1 {margin: 10px 20px 30px 40px}

Margin properties
► Properties
  – margin
  – margin-top
  – margin-right
  – margin-bottom
  – margin-left

h1 {margin: 40px}

Margin properties
► All margins can be set the same

h1 {margin: 40px}

Margin properties
► margin-bottom
► auto
  – set by the browser
► length
  – fixed
► %

Syntax: h1 {margin-bottom: 20px}

Margin properties
► Margin settings can be paired (left and right, top and bottom)

h1 {margin: 40px 5%}

In this example, the top and bottom margins would be 40 pixels, while the left and right margins would be 5% of the total height of the document.
Margin properties

- 0 size margins do not need to be specified. 0px, 0pt and 0 are all equivalent.

```
h1 {margin: 40px 0 5% 0}
```

In this example, the top margin would be 40 pixels, the left and right margins would be 0, and the bottom margin would be 5% of the total height of the document.

Benefits of Compressing Data

- **Less data**
  - Optimizes
    - Cost-efficiencies
    - Bandwidth
    - Storage
  - Enables
    - Higher quality
    - More channels

Multimedia Standards

- **Image Compression Standards**: JPEG, JPEG2000
- **Video Compression Standards**: MPEG-1,2,4
- **Image and Video Indexing Standards**: MPEG-7

Benefits of Compressing Data

- Facilitates the exchange of compressed data between various devices and applications.
- Economy of scale: permits common hardware/software to be used for a wide range of products, thus lowering the cost and shortening the development time.
- Provides reference points for the expected quality of compressed images.
International Standard Organizations

- **ISO/IEC**
  - International Organization for Standardization
  - Deals with information processing, e.g., image storage and retrieval

- **ITU-T**
  - International Telecommunication Union
  - Formerly known as CCITT
  - Deals with information transmission

Image Compression Standards

- **Binary (bi-level) images**:
  - Group 3 & 4 (1980); JBIG (1994); JBIG2

- **Continuous-tone still images**:

- **Image sequences (moving pictures)**:
  - H.261 (1990); H.263 (1995);
  - H.263+ (1997), H.263L
  - MPEG1 (1994); MPEG2 (1995)
  - MPEG4 (1997)
  - MPEG7 (2001)
  - MPEG21

JPEG Summary

- The JPEG committee has published the following standards:
  - ISO/IEC 10918-2 | ITU-T Rec. T.83: Compliance testing
  - ISO/IEC 10918-3 | ITU-T Rec. T.84: Extensions
  - ISO/IEC 14495-1 | ITU-T Rec. T.87: Lossless and Near-Lossless Compression of Continuous-Tone Still Images – Baseline

Examples of JPEG Applications

- Consumer imaging (digital cameras, picture disk, etc.)
- Professional imaging (desktop publishing, graphic arts, digital cameras, etc.)
- Medical imaging
- Internet imaging
- Scanning and printing
- Image databases
- Mobile

What Is JPEG?

- The JPEG (Joint Photographic Experts Group) committee, formed in 1986, has been chartered with the
  - “Digital compression and coding of continuous-tone still images”
- Joint between ISO and ITU-T
- Has developed standards for the compression of lossy, lossless, and nearly lossless of still images in the past decade
- Web site: www.jpeg.org

Baseline JPEG Encoder Block Diagram
Baseline JPEG Decoder Block Diagram

Baseline JPEG Pros and Cons

Advantages
► Memory Efficient
► Low complexity
► Compression efficiency
► Visual model utilization
► Robustness

Disadvantages
► Single resolution
► Single quality
► No target bit rate
► No lossless capability
► No tiling
► No ROI
► Blocking artifacts
► Poor error resilience

JPEG2000 Compression Paradigm

- Coding efficiency
- Multi-resolution
- Target bit rate
- Quality scalability
- Lossless to lossy progression
- Tiling
- Improved error resilience
- Flexible bit stream syntax
- Idempotent recompression
- PKE format
0.125 bpp, CR=64:1

0.25 bpp, CR=32:1

0.5 bpp, CR=16:1
The JPEG 2000 Standard

► Part 1: Core Image Coding System (royalty and fee free)
► Part 2: Extensions
► Part 3: Motion JPEG 2000
► Part 4: Conformance Testing
► Part 5: Reference Software
► Part 6: Compound Image File Format

JPEG 2000 Fundamental Building Blocks

Preprocessing

► The input image is partitioned into rectangular and non-overlapping tiles of equal size (except possibly for those tiles at the image borders) that are compressed independently using their own set of specified compression parameters.
► The unsigned sample values in each component are level shifted (DC offset) by subtracting a fixed value from each sample to make its value symmetric around zero.
► The level-shifted values can be subjected to a forward point-wise inter-component transformation to decorrelate the color data. A restriction is that components must have identical bit-depths and dimension.
Reversible Color Transform

- Two color transforms have been defined in JPEG2000.
  - The reversible color transform (RCT) that is integer-to-integer and is intended for lossless coding.
  - The irreversible color transform (ICT) that is the same as the conventional RGB to YCbCr transform.

- Forward RCT:
  \[ Y = \frac{1}{4}(R + 2G + B) \]
  \[ C_b = B - G \]
  \[ C_r = R - G \]

- Inverse RCT:
  \[ Y = \frac{1}{4}(4Y - C_b - C_r) \]
  \[ R = C_r + G \]
  \[ B = C_b + G \]

Irreversible Color Transform (ICT)

The ICT is the same as the conventional YCbCr transform for the representation of image and video signals:

\[
\begin{bmatrix}
Y \\
C_b \\
C_r
\end{bmatrix} =
\begin{bmatrix}
0.299 & 0.587 & 0.114 \\
-0.169 & -0.331 & 0.500 \\
0.500 & -0.419 & -0.081
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} =
\begin{bmatrix}
1.0 & 0.0 & 1.4021 \\
1.0 & -0.3441 & -0.7142 \\
1.0 & 1.7718 & 0.0
\end{bmatrix}
\begin{bmatrix}
Y \\
C_b \\
C_r
\end{bmatrix}
\]
Discrete Wavelet Transform (DWT)

Transformations in JPEG 2000

- Multi-resolution image representation is inherent to DWT.
- The full-frame nature of the transform decorrelates the image across a larger scale and eliminates blocking artifacts at high compression ratios.
- Use of integer DWT filters allows for both lossless and lossy compression within a single compressed bit-stream.
- DWT provides a frequency band decomposition of the image where each subband can be quantized according to its visual importance.

1D Two-Band DWT

Example of Analysis Filter Bank

- 1-D signal:
  \[
  \ldots 100 \quad 100 \quad 100 \quad 100 \quad 200 \quad 200 \quad 200 \ldots
  \]
- Low-pass filter \( h_0 \): \((-1 \quad 2 \quad 6 \quad 2 \quad -1)/8\)
- High-pass filter \( h_1 \): \((-1 \quad 2 \quad -1)/2\)
- Before downsampling:
  \[
  \ldots \quad 100 \quad 100 \quad 87.5 \quad 112.5 \quad 187.5 \quad 212.5 \quad 200 \quad 200 \ldots
  \]
  \[
  \ldots \quad 0 \quad 0 \quad 0 \quad -50 \quad 20 \quad 0 \quad 0 \quad 0 \ldots
  \]
- After downsampling:
  \[
  \ldots \quad 100 \quad 112.5 \quad 50 \quad 212.5 \quad 200 \quad 0 \quad 0 \ldots
  \]
The 1D Two Band DWT

Ideally, it is desired to choose the analysis filter-bank \( h_0 \) and \( h_1 \), and the synthesis filter-bank \( g_0 \) and \( g_1 \), in such a way so as to make the overall distortion zero, i.e., \( y(t) = x(t) \). This is called the perfect reconstruction property.

Inverse DWT

- During the inverse DWT, each subband is interpolated by a factor of two by inserting zeros between samples and then filtering each resulting sequence with the corresponding low-pass, \( g_0 \), or high-pass, \( g_1 \), synthesis filter-bank.
- The filtered sequences are added together to form an approximation to the original signal:

\[
\begin{align*}
... & 0 \quad 100 \quad 0 \quad 112.5 \quad 0 \quad 212.5 \quad 0 \quad 200... \\
... & 0 \quad 0 \quad 0 \quad 0 \quad 50 \quad 0 \quad 0 \quad 0... \\
... & 100 \quad 100 \quad 100 \quad 200 \quad 200 \quad 200 \quad 200... 
\end{align*}
\]
Bi-Orthogonal Filter Banks

- Most wavelet based image compression systems use a class of analysis/synthesis filters known as bi-orthogonal filters:
  - The basis functions corresponding to $h_0(n)$ and $g_0(n)$ are orthogonal; and the basis functions for $h_1(n)$ and $g_1(n)$ are orthogonal.
  - Linear-phase (symmetrical) and perfect reconstruction
  - Unequal length; odd-length filters differ by an odd multiple of two (e.g., 7/9), while even-length filters differ by an even multiple of two (e.g., 6/10)
  - Symmetric boundary extension.
**Error Resilience**

- **Error-prone Channels**
  - When delivering compressed images across error-prone channels any transmission error can severely affect the decoded image quality. This is specially true since variable length coding is used in the code-block entropy coding and packet heads.
  - Error types can be random errors, burst error and missing bytes (i.e. network packet loss).
  - Since each code-block is independently coded an error in a code-block's bitstream will be contained within that code-block. Nevertheless severe distortion can occur in the case of an error.
  - Packet heads are interdependent and thus fragile.

- **Error-Resilience Visual Results**
  - BER = $10^{-5}$
  - JPEG CR=16:1
  - JPEG 2000 CR=16:1

**Error Effects**

- In a **packet body**: corrupted arithmetically coded data for some code-block; wrong symbols are decoded and wrong contexts are formed for subsequent bit-planes ⇒ severe distortion.
- In a **packet head**: wrong body length can be decoded, code-block data can be assigned to wrong code-blocks and subsequent packets cannot be correctly located ⇒ total synchronization loss.
- **Bytes missing** (i.e. network packet loss): combined effects of error in packet head and body.

**Protecting Code-block Data**

- **Segmentation symbols**: a special symbol sequence is coded with a fixed context at the end of each bitplane. If the wrong sequence is decoded an error has occurred and the last bitplane is corrupted (at least).
- **Regular predictable termination**: the arithmetic coder is terminated at the end of each coding pass using a special algorithm (predictable termination). The decoder reproduces the termination and if it does not find the same unused bits at the end an error has occurred in the last coding pass (at least).
- Both mechanisms can be freely mixed but slightly decrease the compression efficiency.
Error-Resilience Visual Results

BER = $10^{-4}$

JPEG CR=16:1  JPEG 2000 CR=16:1

Test Images

BER = $10^{-4}$

Multimedia Standards

JPEG-LS

- Low Complexity Lossless Compression (LOCO-I)
  - Prediction w/ context modeling
  - Golomb coding (of power-of-two order)
  - Flat region detector
  - The prediction is based on a median edge detector for the non-adaptive part.
  - The context modeling is based on the local gradients (D-B, B-C, C-A). It determines the adaptive part of the prediction and the Golomb coder order.
  - "Near-lossless" coding is obtained using a maximum allowable sample error.

Multimedia Standards

Test Images (Cont’d)

Multimedia Standards

PNG

- Lossless only
- Predictive scheme
  - Current value X predicted from A, B, C
  - Five possible predictors
  - Predictors chosen on a line-by-line basis
- Entropy coding
  - "Deflate" method: LZ77 coupled with Huffman
  - Same as used in ZIP file compression
Lossless Results

- JPEG-LS is overall best performer with best compression ratios and fastest execution. JPEG 2000 compression ratios are, in general, close to JPEG-LS ones. PNG and SPIHT come close behind.
- Notable exception to the general trend is target and, to a lesser extent, cmpnt1. Because of L777 the regular structure of target is very well exploited by PNG.
- JPEG 2000 is considerably slower than JPEG-LS or L-JPEG but faster than others at compression.
- JPEG 2000 performs well with various image types and reasonably better than the other DWT algorithm with non-natural images.

Non-progressive Results

- JPEG 2000 with the non-reversible (9,7) filter outperforms all other algorithms at all bitrates.
- The reversible (5,3) filter incurs a small penalty for the capability of lossless decoding, but still outperforms all other algorithms (except SPIHT @ 2 bpp).
- JPEG exhibits a considerable quality difference at all bitrates, from 2.7 to 1.8 dB inferior PSNR.
- The difference in compression efficiency of JPEG 2000 over the other algorithms gets larger as the compression ratio increases.
Video Compression Standards

- MPEG-1 and MPEG-2 provide interoperable ways of representing audiovisual content, commonly used on digital media and on the air
- MPEG-4 defines how to represent content
- MPEG-7 specifies how to describe content
- MPEG-21 provides a truly interoperable multimedia framework
MPEG-1 — ISO/IEC 11172

- Coding of moving pictures and associated audio for digital storage media
- Video and audio at 1.5M bit/s for CD-ROM
- Five parts
  - Part 1 (systems): multiplexing & synchronization
  - Part 2 (video): ~VHS quality at 1.15M bit/s
  - Part 3 (audio): stereo at 384K, 256K, 192K bit/s
  - Part 4 (conformance testing): references for decoder
  - Part 5 (reference software): C implementation
- Applications: Video CD, MP3

MPEG-2 — ISO/IEC 13818

- Generic coding of moving pictures and associated audio
- Digital Storage Media Command and Control (DSM-CC) for session set up and remote control of a server, used in set top boxes for satellite and cable TV
- Advanced Audio Coding (AAC) for multi-channel audio
- 4:2:2 profile for TV production studios
- Provisions for Intellectual Property Management and Protection (IPMP)
- Applications: digital TV set top boxes, DVD
- Transport Stream version
- Patent issues

MPEG-4 — ISO/IEC 14496

- Coding of audiovisual objects
- MPEG-4 defines how to represent content
  - ancestry: VRML
  - interoperability of content structure
    - AFX — Animation Framework eXtension
    - XMT — textual XML format for SMIL, Web3D, etc.
  - adapt transparently to device capabilities
- FSG — Fine Granularity Scalability
- Extensions of AAC and IPMP, Studio Profile
- MP4 and AVC file formats, multi-user environment
- Patent issues

MPEG-7 — ISO/IEC 15938

- Multimedia content description interface
- MPEG-7 specifies how to describe content
  - describe content way beyond metadata
  - facilitate content management, in particular searching

MPEG-21 — ISO/IEC (18034) 21000

- Define the technology needed to support Users to exchange, access, trade and otherwise manipulate Digital Items in an efficient, transparent and interoperable way

MPEG-2 applications

- BSS — Broadcasting Satellite Service (to the home)
- CATV — Cable TV Distribution on optical networks, copper, etc.
- GBAD — Global Digital Audio Distribution
- DBR — Digital Audio Broadcasting (terrestrial and satellite broadcasting)
- DTSB — Digital Terrestrial Television Broadcast
- EC — Electronic Cinema
- ENG — Electronic News Gathering (including SNG, Satellite News Gathering)
- FSS — Fixed Satellite Services (e.g. head ends)
- HTT — Home Televisio Theatre
- IPC — Interpersonal Communications (videoconferencing, videophone, etc.)
- ISM — Interactive Storage Media (optical disks, etc.)
- MM — Multimedia Mailing
- NCA — News and Current Affairs
- NSD — Networked Database Services (via ATM, etc.)
- RVS — Remote Video Surveillance
- SSM — Serial Storage Media (digital VTR, etc.)
Typical video data rates

- HDTV: ~1500 Mbps
- 10-bit CCIR 601: 275 Mbps
- 8-bit CCIR 601: 216 Mbps
- 8-bit CCIR 601 (active only): 167 Mbps
- Digital Betacam (R): ~90 Mbps
- MPEG-1 4:2:2 PAL: 10.50 Mbps
- MPEG-2 MP@ML: 2-15 Mbps
- MPEG-1 constrain. param.: 0.5-1.3 Mbps
- H.261 videoconferencing: 64 Kbps - 1.5 Mbps
- H.263 videoconferencing: 4 Kbps - 0.5 Mbps

MPEG picture coding tools

- Entropy coding
  - Run length coding
  - Huffman coding
- Predictive coding
  - Motion estimation
  - DPCM (Discrete Pulse Code Modulation)

Digital video compression

- Goal: Minimize video storage capacity or bandwidth (measured in bits/second of video)
- What determines the bit rate?
  - Picture format
  - Scene complexity
  - Constraints
    - Quality
    - Delay
    - Encoder complexity and algorithm
    - Noise

MPEG-2

- Start with CCIR-601 video
- Serial component digital video
- 270 Mbit rate

MPEG picture coding tools

- Reduced resolution
  - Spatial: send small pictures
  - Temporal: reduce number of picture per second
- Transform coding (discrete cosine transform)
- Subjective importance
  - 4:2:2 and 4:2:0 chrominance formats
  - Quantization matrices

IntRA-frame coding

I
Reduce size
Picture sizes

- CCIR 601 525/30/2:1: 720 x 576
- CCIR 601 525/25/2:1: 720 x 576
- MPEG-2 422P@ML 30fps: 720 x 512
- MPEG-2 422P@ML 25fps: 720 x 500
- MPEG-2 30fps (quasi-std): 704 x 480
- MPEG-2 25fps (quasi-std): 704 x 576
- SIF (30fps, 25fps): 352 x 240
- CIF (always 30fps): 352 x 240
- QCIF, 2/3-HR, 3/4-HR: 352, 480, 528 x 240
- QSIF (20fps, 25fps): 176 x 120

Macroblocks & Chroma formats

<table>
<thead>
<tr>
<th>Y</th>
<th>Cb</th>
<th>Cr</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:2:0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4:2:2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4:4:4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

IntRA-frame coding

- Reduce size
- Sub-sample

MPEG-2 encoder

Discrete levels to frequency coefficient conversion

Video in ➔ DCT ➔ VLC ➔ RLC ➔ MUX ➔ Buffer underflow/overflow ➔ Buffer
**Discrete cosine transform**

- Picture
- Sample values
- DCT coefficients

![Flowchart](image)

**DCT/Quantizer**

- Eye is less sensitive to high frequencies
- DCT matrix coefficients divided by quantization matrix
- To minimize distortion, quantization step must be small
- Quantizer is non-linear giving further bit reduction

**MPEG-2 encoder, continued**

- Video in → DCT → Quantizer → VLC → RLC → MUX → Buffer

**Entropy coding**

- Use short code words for the most common symbols
- Huffman codes

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Code word</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0.25</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>0.125</td>
<td>100</td>
</tr>
<tr>
<td>D</td>
<td>0.0625</td>
<td>1010</td>
</tr>
<tr>
<td>E</td>
<td>0.03125</td>
<td>11110</td>
</tr>
<tr>
<td>F</td>
<td>0.03125</td>
<td>11111</td>
</tr>
</tbody>
</table>
Run Length Coding (RLC)
- The effect of the various coding techniques up to this point is to reduce most of the values to zero or near zero.
- The output of the VLC has strings of zeros.
- This can be optimized by the pattern used to read the data.
- The string of zeros is replaced with a code representing the \( n \) of zeros in the string.

Inter-frame coding
Motion compensation

Buffer
- Buffer maintains constant output data rate
- If buffer begins to get empty, it sends underflow to one quantizer to increase the bit rate output
- If buffer begins to get full, it sends overflow back to the quantizer to reduce its bit rate output

Inverse Quantizer, DCT and Sum
- Area function = return data to input format
- Data summed with prediction data
- Data now representation of actual and predicted data
- Data can now be checked for errors in prediction and a new prediction made

Compression
- Encoder now basically JPEG or MPEG-1
- MPEG-1 uses motion compensation in progressive frame

Macroblock search
- Macroblock checked against previous frames for a match
- If match found, only motion vector need be encoded
- Encoder search within 1/2 pixel accuracy
- Results: lossless compression with large bit rate reduction
- Checks forward and backward from anchor frames
Motion Compensation
Inter-frame coding
- I frames stored uncompressed
- P frames predicted from I frames
- B frames generated forward and backward
- I and P frames transmitted anchors
- In practice, every 12 to 16 frames to prevent errors in prediction from feeding too long

Picture types and groups
Bidirectional interpolation
- I pictures: Composed of intra macroblocks only
- P pictures: Contain forward motion compensation and intra macroblocks
- B pictures: Contain forward, backward & bi-directional MC plus intra macroblocks

MPEG-2
ISO/IEC 13818/Recommendation H.220.0 (1994)
- 13818-1 system level coding (audio and video multiplex)
- 13818-2 coding and decoding of video data
- 13818-3 coding of audio data
- 13818-4 compliance testing of the other three
- Standard covers the decoder methodology and transport stream syntax

MPEG-2 encoder/compressor
- MPEG-2 requires component digital video (Rec. 601)
- MPEG-2 encoder/compression product can include composite decoders and/or A to U converters.
- Audio input must be AES/EBU digital or encoder must include a converter/formatter
- Elementary streams can be of any length

MPEG-2 profiles and levels

MPEG-2 Packetizer
- Packetizer forms video and/or audio into Packetized Elementary Stream (PES) packets
- PES packets contain:
  - Header data
  - Elementary stream data
Program stream mux

- Combines video and audio PES packets into a stream
- Designed for transmission in error and noise free environments

Transport layer

- The Transport Stream (TS) is a continuous data stream in 188 byte packets containing format (syntax) information and payload data

MPEG-2 compression data

Basic MPEG-2 data layers and terminology

- Video Data
- Audio Data
- Video encoder
- Audio encoder
- Packetized Video PES
- Packetized Audio PES
- Program Stream MUX
- Program Stream
- Elementary stream
- Transport Stream MUX
- Transport Stream

Transport layer

- Transport stream PSI
  - Program Association Table (PAT)
  - Program Map Tables (PMT)
  - Conditional Access Table (CAT)
  - Packet Identification (PID)

Transport stream MUX

- Combines video and audio PES packets into a stream
- Designed for transmission to ITU-T Rec. H.262
- QCIF, CIF, and SVC streams
- TS contains no error protection itself
- Use of small packets provides some resistance to results of errors
- Combines asynchronous signals to synchronous data stream

MPEG-2 decoder

- Channel decoder
- Video decoder
- Audio decoder
- Decoded Video
- Decoded Audio
MPEG-7

- MPEG-7 is not about compression; it is about metadata “bits about the bits”
- Metadata is digital information that describes the content of other digital data

MPEG-21

- An open framework for multimedia delivery and consumption
- Focal points:
  - Content creators
  - Content consumers

MPEG-21 Scope

- User application
- Metadata processing
- Resource processing
  - Multiple formats
- Application level protocols
  - Multiple formats
- Transport & network level protocols
  - Multiple formats
MPEG-21 Digital Item
► Structured digital objects, including a standard representation and identification, and metadata
► Fundamental unit of distribution and transaction within the MPEG-21 framework
► No further technical meaning

MPEG-21 User
► A User is any entity that interacts in the MPEG-21 environment or makes use of a Digital Item
  – Users include individuals, consumers, communities, organizations, corporations, consortia, governments and other standards bodies and initiatives around the world.
► Users are identified specifically by their relationship to another User for a certain interaction
► MPEG-21 makes no distinction between a “content provider” and a “consumer” — both are Users
  – A single entity may use content in many ways
  – however, a User may assume specific or even unique rights and responsibilities according to their interaction with other Users within MPEG-21

The parts of MPEG-21
► 1. Vision, technologies and strategies
► 2. Digital Item Declaration
► 3. Digital Item Identification
► 5. Rights Expression Language
► 6. Rights Data Dictionary
► 7. Digital Item Adaptation
► 8. Reference Software
► 9. File Format

MPEG-21 CE Testbed

Users and content
Digital assets

- World Wide Web’s phases
  - 1990 (info.cern.ch) — scientific exchange
  - 1995 (tidal wave) — free content
  - 2000 (dot bomb) — ubiquitous fast network

- Users are starting to recognize the value of their digital asset resources
- Markets must be efficient

The need for harmonization

- s/w platform owners
- h/w platform owners
- creators
- distributors
- service providers
- gadget platform owners
- fiduciaries

MPEG-21 Part 2 — Digital Item Declaration

- Purpose: describe a set of abstract terms and concepts to form a useful model for defining Digital Items
- Three normative sections:
  - Model
    - set of abstract terms and concepts
  - Representation
    - normative description of syntax & semantics of DID elements
  - Schema
    - normative XML schema comprising the entire grammar of DID

MPEG-21 Part 3 — Digital Item Identification

- The scope of the Digital Item Identification (DII) specification includes:
  - How to uniquely identify Digital Items and parts thereof (including resources)
  - How to use identifiers to link Digital Items with related information such as descriptive metadata

Digital Item Declaration in detail

Digital Item Declaration example
DII example: MPEG-21 music album

MPEG-21 Part 4 — IPMP
► Improvements over MPEG-4 IPMP:
  – Internetworking
  – IPMP tool retrieval & authentication
  – Integration of Rights Expressions (RDD & REL)
► Intellectual Property Management and Protection involves the enforcement of REL permissions
  – IPMP shall consult REL before any actions are taken in the User’s system
► REL: What is protected? What right applies?
  – IPMP: How is it protected?

MPEG-21 Part 5 — REL
► The Rights Expression Language consists of licenses and grants that give specific permissions to Users to perform certain actions on certain resources, given that certain conditions are met
  – Grants can also allow Users to delegate authority to others
► User’s system shall parse and validate the RE
► User’s system shall check permissions before any further action is done
► DID parser is responsible for discovering and identifying where to gather licenses
► REL licenses are wrapped in Digital Items

REL data model
► REL grant consist of
  – principal to whom grant is issued
  – rights the grant specifies
  – resource to which right in grant applies
  – condition to be met before grant can be exercised

MPEG-21 Part 5 — REL
► Rights Expression Language
  – A machine-readable language
  – Can declare rights and permissions
  – Uses terms defined in the Rights Data Dictionary

MPEG-21 Part 6 — Rights Data Dictionary
► Set of clear, consistent, structured, integrated and uniquely identified Terms to support REL
► Specification of dictionary structure and methodology to create dictionary
► Dictionary is prescriptive, inclusive, and has audit provisions
► Legal definitions are mapped from other Authorities
► Supports mapping & transformation of metadata from terminology of one namespace (or Authority) into that of another namespace in automated or partially automated way
► Dictionary is based on a logical model, the Context Model, which is the basis of the dictionary ontology
MPEG-21 Part 7 — Digital Item Adaptation

- Goal: achieve transparent interoperable access to distributed multimedia content
- Enable ad hoc formation of User communities in which contents is shared with agreed or contracted
  - Quality
  - Reliability
  - Flexibility
  - Diversity
- Guaranteed user experience

Scope of standardization

- User Characteristics
  - Terminal Capabilities
  - Network Characteristics
  - Natural Environment Characteristics
  - Resource Adaptability
  - Session Mobility

Concept of Digital Item Adaptation

Overview of DIA Tools

Relation between DIA and other MPEG-21 parts

Bitstream Syntax Description

- A BSD describes the syntax (high level structure) of a binary media resource
- BSDL: XML schema based language to design specific bitstream syntax schemas for particular media formats
- gBS schema: generic schema enabling the construction of resource format independent bitstream syntax descriptions
Adaptation architecture

Terminal and Network Quality of Service
- AdaptationQoS specifies the relationship between constraints and feasible adaptation operations
- Constraints: BandwidthInkbps, ComputationTimeInMilliSEcs
- Utilities (qualities): PSNRIndB
- Adaptation Methods:
  - frameDroppingAndOrCoefficientDropping, requantization, fineGranularScalability, waveletReduction, spatialSizeReduction
- UtilityFunction:
  - describes possible adaptation operators and associated qualities using a set of constraint points as indexes
  - Linear interpolation is assumed between constraint points
- LookUpTable:
  - additional multi-dimensional sets of data to support more elaborate adaptation scenarios
- StackFunction
  - tool for describing the data in numerical function format

(g)BSD, AdaptationQoS, and Link
- Success of a standard depends on the availability of reference software
- Plan to use the software developed in Core Experiments (CE) as a basis
- Future repository (requires membership)
  - http://mpeg.nist.gov/cvsweb/MPEG-21/
  - Temporary repository
    - http://www.titr.uow.edu.au/cgi-bin/mpeg-ref-sw.pl
- Current main issue: parsing DID

Metadata Adaptability
- Metadata: AdaptationHint describes adaptation hint information pertaining to metadata within a digital item
- Hint: a set of syntactical elements with prior knowledge about the metadata that is useful for reducing the complexity of the metadata adaptation process
## Digital Item Processing

- Digital Items act as a structure for organizing resources and its descriptions.
- Need a mechanism for defining a set of operations by which a terminal can process a DI or DID.
- Currently considering to specify a set of operations that can be used to process DIs: Digital Item Method.
- A DIM defines an intended method for configuring, manipulating and/or validating a DI.

### Multimedia Standards

- **Digital Item Processing**

## Methods vs. processing

- Interoperability of Digital Items means that terminals must handle the DIs in a consistent manner.
- Digital Item Methods provide a way to specify a selection of preferred procedures by which the DI should be handled at the DI level.
- A menu of user interaction possibilities.
- Digital Item Processing encompasses all aspects of processing a DI from an application perspective.
- Applications build DIP environments around a fundamental DIME.

### DIP flow control

## Digital Item processing terminology

- **CDI** — Content Digital Item.
  - A DID containing the actual content.
- **DIBO** — Digital Item Base Operation.
- **DIM** — Digital Item Method.
  - Method that can be applied to a DID.
- **DIME** — DIM Engine.
  - Part of the terminal responsible for executing the DIM.
- **DIML** — DIM Language.
- **DIP** — Digital Item Processing.
- **MI** — Method Item.
- **PI** — Processing Item.
- **XDI** — Context Digital Item.

### MPEG-21 Part 9 — File Format

- An MPEG-21 file format shall be capable of storing MPEG-21 Digital Items.
  - All components of the DI within a single file.
- The MPEG-21 file format will inherit several concepts from MP4, in order to make ‘multi-purpose’ files possible.
**Dynamic HTML: JavaScript**

**What is JavaScript?**
- Language developed by Netscape
- Primary purpose is for "client-end" processing of HTML documents
  - JavaScript code is embedded within the html of a document
  - An interpreter in the browser interprets the JavaScript code when appropriate
  - Code typically allows for "preprocessing" of forms and can add "dynamic content" to a Web page

**What can a JavaScript Do?**
- JavaScript gives HTML designers a programming tool
  - HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- JavaScript can put dynamic text into an HTML page
  - A JavaScript statement like this: `document.write("<h1>" + name + "</h1>")` can write a variable text into an HTML page
- JavaScript can react to events: A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element

**Are Java and JavaScript the Same?**
- NO!!
- Java and JavaScript are two completely different languages in both concept and design!
- Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C# and C++.
What can a JavaScript Do?

► JavaScript can read and write HTML elements: A JavaScript can read and change the content of an HTML element

► JavaScript can be used to validate data: A JavaScript can be used to validate form data before it is submitted to a server, this will save the server from extra processing

► JavaScript can be used to detect the visitor's browser: A JavaScript can be used to detect the visitor's browser, and depending on the browser - load another page specifically designed for that browser

► JavaScript can be used to create cookies: A JavaScript can be used to store and retrieve information on the visitor's computer

How to include JavaScript in html?

► JavaScript programs require the <SCRIPT> tag in .html files
<script type = "text/javascript">
ACTUAL JavaScript code here
</script>

► These can appear in either the <HEAD> or <BODY> section of an html document
  – Functions and code that may execute multiple times is typically placed in the <HEAD>
  • These are only interpreted when the relevant function or event-handler are called

Simple Example

<HTML>
<HEAD>
<TITLE>First JavaScript Example</TITLE>
</HEAD>
<BODY>
<H2>This line is straight HTML</H2>
<script type = "text/javascript">
document.write("These lines are produced by<br/>");
document.write("the JavaScript program<br/>");
alert("Hey, JavaScript is fun!");
</script>
</BODY>
</HTML>

JavaScript Variables

► JavaScript variables have no types
  – Type is determined dynamically, based on the value stored
    • This is becoming familiar!
  – The typeof operator can be used to check type of a variable

► Declarations are made using the var keyword
  – Can be implicitly declared, but not advisable
  – Declarations outside of any function are global
  – Declarations within a function are local to that function
  – Variables declared but not initialized have the value undefined
JavaScript Variables

- A big difference between JavaScript and other languages like JAVA and C is that JavaScript is untyped.
- This means that a JavaScript variable can hold a value of any data type, and its data type does not have to be set when declaring the variable.
- This allows you to change the data type of a variable during the execution of your program, for example:
  ```javascript
  var x = 10;
  x = "ten";
  ```
- In this example the variable x is first assigned the integer value of 10, and then the string value of the word ten.

JavaScript Expressions

- Similar to PHP, with mixed number/string type expressions, JavaScript will coerce if it can
  - If operator is + and an operand is string, it will always coerce other to string
  - If operator is arithmetic, and string value can be coerced to a number it will do so
    - If string is non-numeric, result is NaN
    - We can also explicitly convert the string to a number using parseInt and parseFloat
      - Again looks like Java

JavaScript Expressions

- Numeric operators in JavaScript are similar to those in most languages
  - +, -, *, /, %, ++, --
  - Precedence and associativity are also fairly standard
- Strings
  - Have the + operator for concatenation
  - Have a number of methods to do typical string operations
    - charAt, indexOf, toLowerCase, substring
  - Looks kind of like Java – intentionally

Control Statements

- Relational operators:
  - ==, !=, <, >, <=, >=
  - The above allow for type coercion. To prevent coercion there is also
    - ===, !==
      - Similar to PHP
- Boolean operators
  - &&, ||, !
    - &&, || are short-circuited (as in Java and PHP)

Control Statements

- Control statements similar to Java
  - if, while, do, for, switch
    - Variables declared in for loop header are global to the rest of the script
- Functions
  - Similar to C++ functions, but
    - Header is somewhat different
      - function name(param_list)
        - Return type not specified (since JS has dynamic typing)
        - Param types also not specified
Functions

- Functions execute when they are called, just as in any language.
- To allow this, function code should be in the `<HEAD>` section of the .html file.
- Variables declared in a function are local to the function.
- Parameters are all value.
  - No parameter type-checking
  - Numbers of formal and actual parameters do not have to correspond
    - Extra actual parameters are ignored
    - Extra formal parameters are undefined
  - All actual parameters can be accessed regardless of formal parameters by using the `arguments` array.

Array Objects

- More relaxed version of Java arrays:
  - Size can be changed and data can be mixed.
  - Cannot use arbitrary keys as with PHP arrays.
- Creating arrays:
  - Using the new operator and a constructor with multiple arguments.
    - `var A = new Array("hello", 2, "you");`
  - Using the new operator and a constructor with a single numeric argument.
    - `var B = new Array(50);`
  - Using square brackets to make a literal.
    - `var C = ["we", "can", 50, "mix", 3.5, "types"];`

Array Objects

- Array Length:
  - Like in Java, length is an attribute of all array objects.
  - However, like in Javascript it does not necessarily represent the number of items or even mem. locations in the array.
  - Unlike Java, length can be changed by the programmer.
  - Actual memory allocation is dynamic and occurs when necessary.
    - An array with length 1234 may in fact have memory allocated for only a few elements.
    - When accessed, empty elements are undefined.

Array Methods

- There are a number of predefined operations that you can do with arrays.
  - `concat` two arrays into one.
  - `join` array items into a single string (commas between).
  - `push`, `pop`, `shift`, `unshift`.
    - Push and pop are a "right stack".
    - Shift and unshift are a "left stack".

Array Methods

- `sort`:
  - Sort by default compares using alphabetical order.
  - To sort using numbers we pass in a comparison function defining how the numbers will be compared.
- `reverse`:
  - Reverse the items in an array.

Array Methods

- These operations are invoked via method calls, in an object-based way:
  - Also many, such as `sort` and `reverse` are mutators, affecting the array itself.
- JavaScript also has 2-dimensional arrays:
  - Created as arrays of arrays, but references are not needed.
JavaScript Objects

JavaScript is an object-based language

- It is NOT object-oriented
- It has and uses objects, but does not support some features necessary for object-oriented languages
  - Class inheritance and polymorphism not supported
  - They can be “faked” but are not really there

JavaScript Objects

- JavaScript objects are actually represented as property-value pairs
  - Very much like Perl hashes
  - The object is analogous to the hash name, and the properties are analogous to the data stored in the hash
  - Properties can be data or functions (methods)
  - Ex:

```javascript
var my_tv = new Object();
my_tv.brand = "Sony";
my_tv.size = 27;
my_tv.jacks = new Object();
my_tv.jacks.input = 3;
my_tv.jacks.output = 2;
```

- Note that the objects can be created and their properties can be changed dynamically
  - Also, objects all have the same data type - object
  - We can write constructor functions for objects if we’d like, but these do not create new data types – just easy ways of uniformly initializing objects

```javascript
function TV(brand, size, injacks, outjacks)
{
  this.brand = brand;
  this.size = size;
  this.jacks = new Object();
  this.jacks.input = injacks;
  this.jacks.output = outjacks;
}

var my_tv = new TV("Sony", 27, 3, 2);
```

DOM

- The Document Object Model
  - Developed by W3C (World-Wide Web Consortium)
  - http://www.w3c.org/DOM/
  - Specifies the contents of Web documents in an object-oriented way
  - Allows programming languages to access and manipulate the components of documents
  - Defined at a high level so that a variety of languages can be used with it
  - It is still being updated / revised
  - We are not even scratching the surface here
Events

- With documents DOM specifies events and event handlers
  - Event model is similar to the one used in Java
  - Different parts of a document have different events associated with them
  - We can define handlers to react to these events
    - These allow us to "interact" with and add "dynamic content" to web documents
    - Ex: Can preprocess form elements
    - Ex: Can load / update / change what is displayed in response to an event

DOM and Events

- document refers to the top-level document
  - Each document has access to its properties and to the components that are declared within it
    - Ex: title, URL, forms[], applets[], images[]
  - Attributes with IDs can also be specified by ID
  - Once we know the components, events and event-handlers, we can write JavaScript programs to process Web pages on the client-side
    - Client computers are typically less busy than servers, so whatever we can do at the client will be helpful overall
    - Ex: Checking form correctness before it is submitted

Example: Pre-processing a Form

- A very common client-side operation is pre-processing a form
  - Ensure that fields are filled and formatted correctly, so server does not have to
    - Saves load on the server, saves time and saves bandwidth
    - We can check a form overall by using the attribute onsubmit
      - We can put it right into the form as an attribute
      - Or we can assign the attribute through the document object

Example: Pre-processing a form

- We can check individual components as they are entered as well
  - Ex: <input type = "text"> has the onchange attribute
    » Triggered when contents are changed and focus changes
  - Ex: <input type = "radio"> has the onclick attribute
    » Triggered when the radio button is clicked with the mouse

Javascript and XSLT

function doTransform(source){
  var objXSL= new ActiveXObject("Msxml2.DOMDocument");
  objXSL.load("query1.xslt");
  str=source.transformNode(objXSL); return (str);
}
### Setting Cookies

- Create a string and then set the `document.cookie` property to that.
- Cookie values must never have spaces, commas, or semicolons.
- A pair of functions will code and decode your properties
  - `escape()`
  - `unescape()`

```javascript
function setCookie() {
  var the_name = prompt("What's your name?", "");
  var the_cookie = "username: " + escape(the_name);
  document.cookie = the_cookie;
  alert("Thanks, now go to the next page.");
}
```

### Complicated Cookie Reading

- If you want your cookie to contain more than just one piece of information, you can make the value of the cookie as long as you want (up to the 4000 character limit).
- Say you're looking to store a person's name, age, and phone number.
- You do something like this:
  ```javascript
  var the_cookie = "username:binnur/age:32/phone:2853608";
  document.cookie = "my_cookie=" + escape(the_cookie);
  ```
- The delimiters you choose are up to you. Just remember what you used so you can decode the cookie later.

### Complicated Cookie Reading

- I use a slash to separate property names and a colon to distinguish the property name from the property value.
- The slash and colon are arbitrary choices — they could be anything, like this:
  ```javascript
  var the_cookie = "username:binnur&age:32&phone:2853608";
  document.cookie = "my_cookie=" + escape(the_cookie);
  ```
- The delimiters you choose are up to you. Just remember what you used so you can decode the cookie later.
function readTheCookie(the_info) {
    // load the cookie into a variable and unescape it
    var the_cookie = document.cookie;
    var the_cookie = unescape(the_cookie);
    // separate the values from the cookie name
    var broken_cookie = the_cookie.split("=");
    // break each name: value pair into an array
    var separated_values = broken_cookie[1];
    // loop through the list of name: values and load up the associate array
    var property_value = ";
    for (var loop = 0; loop < separated_values.length; loop++) {
        property_value = separated_values[loop];
        var broken_info = property_value.split("=");
        var the_property = broken_info[0];
        var the_value = broken_info[1];
        // break each name: value pair into an array
        var separated_values = the_value.split("/");
        // loop through the list of name:values and load up the associate array
        var property_value = ";
        for (var loop = 0; loop < separated_values.length; loop++) {
            property_value = separated_values[loop];
            var broken_info = property_value.split("=");
            var the_property = broken_info[0];
            var the_value = broken_info[1];
            the_info[the_property] = the_value;
        }
    }
}

function readCookie(name) {
    if (document.cookie == '') {
        // there's no cookie, so go no further
        return false;
    } else {
        var firstChar, lastChar;
        var theBigCookie = document.cookie;
        firstChar = theBigCookie.indexOf(name);
        // find the start of 'name'
        if(firstChar != -1) {
            // if you found the cookie
            firstChar += name.length + 1;
            // skip 'name' and the next =
            lastChar = theBigCookie.indexOf(";", firstChar);
            // Find the end of the value string (i.e. the next =)
            if(lastChar == -1) lastChar = theBigCookie.length;
            return unescape(theBigCookie.substring(firstChar, lastChar));
        } else {
            // if there was no cookie of that name, return false.
            return false;
        }
    }
}

function setCookie() {
    // get the information
    var the_name = prompt("What's your name?" , "");
    var the_date = new Date("December 31, 2023");
    var the_cookie_date = the_date.toGMTString();
    // build and save the cookie
    var the_cookie = "my_cookie=" + escape(the_name);
    the_cookie = the_cookie + ";expires=" + the_cookie_date;
    document.cookie = the_cookie;
}

More About Cookies
▼ If you do want to save cookies on users' hard drives, you have to set an expiration date, which has to be in a special format called GMT. For example:
Mon, 27-Apr-1998 00:00:00 GMT
▼ To set an expiration date to some distant time in the future
var the_date = new Date("December 31, 2023");
var the_cookie_date = the_date.toGMTString();

More About Cookies
▼ Here's how to build a cookie that will last until the end of the Mayan calendar:
function setCookie() {
    // get the information
    var the_name = prompt("What's your name?" , "");
    var the_date = new Date("December 31, 2023");
    var the_cookie_date = the_date.toGMTString();
    // build and save the cookie
    var the_cookie = "my_cookie=" + escape(the_name);
    the_cookie = the_cookie + ";expires=" + the_cookie_date;
    document.cookie = the_cookie;
}
Creating an HTML Form

The FORM Tag

The following is a partial structure of an HTML form:

```html
<form action='URL TO CONTROLLER' method='GET or POST'>
<!-- PUT FORM COMPONENT TAGS HERE -->
</form>
```

For example:

```html
<form action='add_league.do' method='POST'>
Year: <input type='text' name='year' /> <br/><br/>
Season: <select name='season'>
<option value='UNKNOWN'>select...</option>
<option value='Spring'>Spring</option>
<option value='Summer'>Summer</option>
<option value='Fall'>Fall</option>
<option value='Winter'>Winter</option>
</select> <br/><br/>
Title: <input type='text' name='title' /> <br/><br/>
<input type='submit' value='Add League' />
</form>
```

Textfield Component

In Netscape, a textfield component looks like this:

```html
<p>This form allows you to create a new soccer league.</p>
<form action='add_league.do’ method='POST'>
Year: <input type='text' name='year' /> <br/><br/>
Season: <select name='season'>
<option value='UNKNOWN'>select...</option>
<option value='Spring'>Spring</option>
<option value='Summer'>Summer</option>
<option value='Fall'>Fall</option>
<option value='Winter'>Winter</option>
</select> <br/><br/>
Title: <input type='text' name='title' /> <br/><br/>
<input type='submit' value='Add League' />
</form>
```

Drop-Down List Component

In Netscape, a drop-down list component looks like this:

```html
<form action='add_league.do’ method='POST'>
Year: <input type='text' name='year' /> <br/><br/>
Season: <select name='season'>
<option value='UNKNOWN'>select...</option>
<option value='Spring'>Spring</option>
<option value='Summer'>Summer</option>
<option value='Fall'>Fall</option>
<option value='Winter'>Winter</option>
</select> <br/><br/>
Title: <input type='text' name='title' /> <br/><br/>
<input type='submit' value='Add League' />
</form>
```

Submit Button

In Netscape, a submit button component might look like this:

```html
<form action='add_league.do’ method='POST'>
Year: <input type='text' name='year' /> <br/><br/>
Season: <select name='season'>
<option value='UNKNOWN'>select...</option>
<option value='Spring'>Spring</option>
<option value='Summer'>Summer</option>
<option value='Fall'>Fall</option>
<option value='Winter'>Winter</option>
</select> <br/><br/>
Title: <input type='text' name='title' /> <br/><br/>
<input type='submit' value='Add League' />
</form>
```
Objectives

► Describe JSP technology
► Write JSP code using scripting elements
► Write JSP code using the page directive
► Write JSP code using standard tags

JavaServer Pages Technology

► JavaServer Pages enable you to write standard HTML pages containing tags that run powerful programs based on Java technology.
► The goal of JSP technology is to support separation of presentation and business logic:
► Web designers can design and update pages without learning the Java programming language.
► Programmers for Java platform can write code without dealing with web page design.

Steps of JSP Page Processing

1. Translate the JSP into a servlet.
2. Compile the servlet bytecode.
3. Load the servlet class.
4. Create the servlet instance.
5. Call the init method.
6. Call the _jspService method.
7. Call the _jspService method.

The hello.jsp Page

```jsp
<%! private static final String DEFAULT_NAME = "World"; %>
<html>
<head>
<title>Hello JavaServer Page</title>
</head>
<bodybgcolor='white'>
<b>Hello, <%= name %></b>
</body>
</html>
```
JSP Processing: Compilation Step

- Web container compiles the servlet source code into a Java class file.
- This servlet class bytecode is then loaded into the Web container’s JVM.

JSP Processing: Initialization Step

- Web container creates an instance of the servlet class and performs the initialization life cycle step by calling the special jspInit method.

JSP Processing: Service Step

- Finally, the Web container can call the _jspService method for the converted JSP page so that it can respond to client HTTP requests.

Developing and Deploying JSP Pages

- Deploying JSP pages is as easy as deploying static pages.
- JSP pages are placed in the same directory hierarchy as HTML pages.
- In the development environment, JSP pages are in the web directory.
- In the deployment environment, JSP pages are placed at the top-level directory of the Web application.

JSP Scripting Elements

- JSP scripting elements are embedded with the `<% %>` tags and are processed by the JSP engine during translation of the JSP page.
- There are five types of scripting elements:
  - Comments `<%-- comment --%>`
  - Directive tag `<%@ directive %>`
  - Declaration tag `<%! decl %>`
  - Scriptlet tag `<% code %>`
  - Expression tag `<%= expr %>`

Comments

- HTML comments
  `<-- This is an HTML comment. It will show up in the response. -->`
- JSP page comments
  `<%-- This is a JSP comment. It will only be seen in the JSP code. It will not show up in either the servlet code or the response. -->%>`
- Java comments
  - Java comments can be embedded with scriptlet and declaration tags.
  - These comments are included in the servlet source code during the translation phase, but do not appear in the HTTP response.
  `<%`  
  `* This is a Java comment. It will show up in the servlet code. It will not show up in the response. */`
### Directive Tag

- A directive tag provides information that will affect the overall translation of the JSP page.
- The syntax for a directive tag is:
  `<%@ DirectiveName [attr="value"]* %>`
- Three types of directives are currently specified in the JSP specification: page, include, and taglib.
- Here are a couple of examples:
  `<%@ page session="false" %>`
  `<%@ include file="incl/copyright.html" %>`

### Declaration Tag

- You can use a declaration tag to override the `jspInit` and `jspDestroy` life cycle methods.
- The signature of these methods has no arguments and returns void.
- For example:
  ```java
  <%!
  public static final String DEFAULT_NAME = "World";
  %>
  public void jspInit() {
    /* initialization code here */
  }
  public void jspDestroy() {
    /* clean up code here */
  }
  ```

### Scriptlet Tag

- A scriptlet tag allows the JSP page developer to include arbitrary Java technology code in the `jspService` method.
- The syntax for a declaration tag is:
  `<%= JavaCode %>`
- Here are a couple of examples:
  ```java
  <% int i = 0; %>
  <% if (i > 10) { %>
    I is a big number.
  <% } else { %>
    I is a small number
  <% } %>
  ```

### Example Table

```html
<table>
<thead>
<tr>
<th>number</th>
<th>squared</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
</tr>
</tbody>
</table>
```

### Declaration Tag

- You can think of one JSP page as being equivalent to one servlet class.
- The declaration tags become declarations in the servlet class, either attributes or methods.
- Here are a couple of examples:
  ```java
  <%!
  public static final String DEFAULT_NAME = "World";
  %>
  public int counter = 0;
  ```

### Scriptlet Tag

- You can use a declaration tag to override the `jspInit` and `jspDestroy` life cycle methods.
- The signature of these methods has no arguments and returns void.
- For example:
  ```java
  <%!
  public void jspInit() {
    /* initialization code here */
  }
  public void jspDestroy() {
    /* clean up code here */
  }
  ```
Expression Tag

- An expression tag holds a Java language expression that is evaluated during an HTTP request.
- The result of the expression is included in the HTTP response stream.
- The syntax for a declaration tag is:

```
<%= JavaExpression %>
```

```
<Ten is <%= (2 * 5) %></Ten>
```

This example shows an arithmetic expression. When this is evaluated, the number 10 is the result. The string "Ten is 10" is sent back in the HTTP response stream.

```
Thank you <%= name %>, for registering for the soccer league.
```

This example shows that you can access local variables declared in the JSP page. If the name variable holds a reference to a String object, then that string is sent back in the HTTP response stream.

```
The current day and time is: <%= new java.util.Date() %>
```

Implicit Variables

- The Web container gives the JSP technology developer access to the following variables in scriptlet and expression tags.
- These variables represent commonly used objects for servlets that JSP developers might need to use.
- For example, you can retrieve HTML form parameter data by using the request variable, which represents the HttpServletRequest object.

The page Directive

- The page directive is used to modify the overall translation of the JSP page.
- For example, you can declare that the servlet code generated from a JSP page requires the use of the Date class:

```
<%@ page import="java.util.*,java.text.*" %>
```

- The import attribute defines the set of classes and packages that must be imported in the servlet class definition. The value of this attribute is a comma-delimited list of fully-qualified class names or packages: `import="java.sql.Date,java.util.*,java.text.*"`
**What is PHP?**

- PHP stands for **H**ypertext **P**reprocessor
- PHP is a server-side scripting language, like ASP
- PHP scripts are executed on the server
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- PHP is an open source software (OSS)
- PHP is free to download and use

**Brief History of PHP**

- PHP (PHP: Hypertext Preprocessor) was created by Rasmus Lerdorf in 1994. It was initially developed for HTTP usage logging and server-side form generation in Unix.
- PHP 2 (1995) transformed the language into a Server-side embedded scripting language. Added database support, file uploads, variables, arrays, recursive functions, conditionals, iteration, regular expressions, etc.
- PHP 3 (1998) added support for ODBC data sources, multiple platform support, email protocols (SNMP, IMAP), and new parser written by Zeev Suraski and Andi Gutmans
- PHP 4 (2000) became an independent component of the web server for added efficiency. The parser was renamed the Zend Engine. Many security features were added.
- PHP 5 (2004) adds Zend Engine II with object oriented programming, robust XML support using the libxml2 library, SOAP extension for interoperability with Web Services, SQLite has been bundled with PHP

**What is MySQL?**

- MySQL is a small database server
- MySQL is ideal for small and medium applications
- MySQL supports standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use

**As of August 2004, PHP is used on 16,946,328 Domains, 1,348,793 IP Addresses**

http://www.php.net/usage.php

This is roughly 32% of all domains on the web.
**PHP + MySQL**

- PHP combined with MySQL are cross-platform (means that you can develop in Windows and serve on a Unix platform)

**PHP**

- PHP is another HUGE language
  - It is a fully functional language
  - It has an incredible amount of built-in features
    - Form processing
    - Output / generate various types of data (not just text)
    - Database access
      - Allows for various DBs and DB formats
    - Object-oriented features
  - We will look at only a TINY part of PHP

**Why PHP?**

- PHP runs on different platforms (Windows, Linux, Unix, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP is FREE to download from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

**Where to Start?**

- Install an Apache server on a Windows or Linux machine
- Install PHP on a Windows or Linux machine
- Install MySQL on a Windows or Linux machine

**PHP Syntax**

- You cannot view the PHP source code by selecting "View source" in the browser - you will only see the output from the PHP file, which is plain HTML.
  - This is because the scripts are executed on the server before the result is sent back to the browser.

**Basic PHP Syntax**

- A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code.
  - A simple PHP script which sends the text “Hello World” to the browser:
    ```html
    <html>
    <body><?php echo "Hello World";</body>
    </html>
    ```
  - A PHP scripting block always starts with `<?php` and ends with `?>`.
  - A PHP scripting block can be placed anywhere in the document.
  - Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another.
  - There are two basic statements to output text with PHP: `echo` and `print`. In the example above we have used the `echo` statement to output the text “Hello World”.

Scalar Types

- **Simple (scalar) types**
  - boolean
    - TRUE or FALSE
  - integer
    - Platform dependent – size of one machine word
      - 32 bits on most machines
  - float
    - Double precision
    - We could call it a double, but since we don’t declare variables float works

- **string**
  - We have single-quoted and double-quoted string literals
  - Double quoted allows for more escape sequences and allows variables to be interpolated into the string
  - What does that mean?
    » Rather than outputting the name of the variable, we output its contents, even within a quote
    » We’ll see an example once we define variables
  - Length can be arbitrary
    - Grows as necessary

Variables in PHP

- **PHP variables**
  - All PHP variables begin with the $ symbol
    - Variable names can begin with an underscore
    - Otherwise rules are similar to most other languages
  - Variables are dynamically typed
    - No type declarations
  - Variables are BOUND or UNBOUND
    - Unbound variables have the value NULL
  - Type information is obtained from the current bound value

Below, the PHP script assigns the string “Hello World” to a variable called $txt:

```php
<html>
<body>
    <?php
        $txt = "Hello World";'s output here is 'e'
    echo $txt;
    ?></body>
</html>
```

PHP programs have access to a large number of predefined variables

- These variables allow the script access to server information, form parameters, environment information, etc.
- Ex:
  - $_SERVER is an array containing much information about the server
  - $_POST is an array containing variables passed to a script via HTTP POST
  - $_ENV is an array containing environment information

Variables in PHP

- No type declarations
- Variables are BOUND or UNBOUND
  - Unbound variables have the value NULL
  - Type information is obtained from the current bound value

Variables in PHP

- String
  - We have single-quoted and double-quoted string literals
  - Double quoted allows for more escape sequences and allows variables to be interpolated into the string
  - What does that mean?
    » Rather than outputting the name of the variable, we output its contents, even within a quote
    » We’ll see an example once we define variables
  - Length can be arbitrary
    - Grows as necessary

Scalar Types

- **Easy conversion back and forth between strings and numbers**
  - In Web applications these are mixed a lot, so PHP will cast between the types when appropriate
  - Can be indexed – the preferred way is using curly braces
    - $mystring = “hello”;
    - Output here is ‘e’
Variables in PHP

To concatenate two or more variables together, use the dot (.) operator:

```
<?php
$txt1 = "Hello World";
$txt2 = "1234";
echo $txt1 . " " . $txt2;
?></body>
```

The output of the script above will be: "Hello World 1234".

Comments in PHP

In PHP, we use // to make a single-line comment or /* and */ to make a large comment block.

```
<?php
//This is a comment
/*
This is a comment block
*/
?></body>
```

PHP Operators – Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>x=2</td>
<td>4</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>x=2</td>
<td>3</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>x=4</td>
<td>x^5</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>15/3</td>
<td>5</td>
</tr>
<tr>
<td>%</td>
<td>Modulus (division remainder)</td>
<td>7%2</td>
<td>1</td>
</tr>
<tr>
<td>++</td>
<td>Increment</td>
<td>x=5</td>
<td>x=6</td>
</tr>
<tr>
<td>--</td>
<td>Decrement</td>
<td>x=5</td>
<td>x=4</td>
</tr>
</tbody>
</table>

PHP Operators – Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Is The Same As</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>x=y</td>
<td>x=y</td>
</tr>
<tr>
<td>+=</td>
<td>x+=y</td>
<td>x=x+y</td>
</tr>
<tr>
<td>-=</td>
<td>x-=y</td>
<td>x=x-y</td>
</tr>
<tr>
<td>*=</td>
<td>x*=y</td>
<td>x=x*y</td>
</tr>
<tr>
<td>/=</td>
<td>x/=y</td>
<td>x=x/y</td>
</tr>
<tr>
<td>%=</td>
<td>x%=-y</td>
<td>x=x%=-y</td>
</tr>
</tbody>
</table>

PHP Operators – Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>is equal to</td>
<td>5==8</td>
</tr>
<tr>
<td>!=</td>
<td>is not equal</td>
<td>5!=8</td>
</tr>
<tr>
<td>&gt;</td>
<td>is greater than</td>
<td>5&gt;8</td>
</tr>
<tr>
<td>&lt;</td>
<td>is less than</td>
<td>5&lt;8</td>
</tr>
<tr>
<td>&gt;=</td>
<td>is greater than or equal to</td>
<td>5&gt;=8</td>
</tr>
<tr>
<td>&lt;=</td>
<td>is less than or equal to</td>
<td>5&lt;=8</td>
</tr>
</tbody>
</table>

PHP Operators – Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>x=6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>not</td>
<td>x=6</td>
</tr>
</tbody>
</table>

Operator Description Example

|x=6
y=3
(x < 10 && y > 1) returns true

|x=6
y=3
(x==5 || y==5) returns false

|x=6
y=3
(x==y) returns true
PHP Operators

- Similar to those in C++ / Java / Perl
- Be careful with a few operators
  - `/` in PHP is always floating point division
  - To get integer division, we must cast to int
    - $x = 15;
    - $y = 6;
    - echo (int) ($x/$y), "<BR />";
    - Output is 2.5 2
  - Inequality operators do not compare strings
  - Will cast strings into numbers before comparing

Conditional Statements

- Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.
- In PHP we have two conditional statements:
  1. if (...else) statement - use this statement if you want to execute a set of code when a condition is true (and another if the condition is not true)
  2. switch statement - use this statement if you want to select one of many sets of lines to execute

if Statement

- If you want to execute some code if a condition is true and another code if a condition is false, use the if...else statement.
- Syntax
  ```php
  if (condition)
  code to be executed if condition is true;
  else
  code to be executed if condition is false;
  ```

Example

- The following example will output "Have a nice weekend!” if the current day is Friday, otherwise it will output "Have a nice day!":
  ```php
  <html>
  <body><?php
  $d=date("D");
  if ($d=="Fri")
    echo "Have a nice weekend!";
  else
    echo "Have a nice day!";
  ?></body>
  </html>
  ```
### if Statement

- If more than one line should be executed when a condition is true, the lines should be enclosed within curly braces:

```php
<?php
$x=10;
if ($x==10) {
    echo "Hello<br />";
    echo "Good morning<br />
}?
</body>
</html>
```

### switch Statement

- If you want to select one of many blocks of code to be executed, use the Switch statement.

```php
<?php
switch ($x) {
    case 1:
        echo "Number 1";
        break;
    case 2:
        echo "Number 2";
        break;
    case 3:
        echo "Number 3";
        break;
    default:
        echo "No number between 1 and 3";
}
</html>
```

### while Statement

- The while statement will execute a block of code if and as long as a condition is true.

```php
<?php
$ii=1;
while($ii<=5) {
    echo "The number is " . $ii . "<br />
    $ii++;
}?
</body>
</html>
```

### Looping

- Very often when you write code, you want the same block of code to run a number of times. You can use looping statements in your code to perform this.

- In PHP we have the following looping statements:
  1. **while** - loops through a block of code if and as long as a specified condition is true
  2. **do...while** - loops through a block of code once, and then repeats the loop as long as a special condition is true
  3. **for** - loops through a block of code a specified number of times
  4. **foreach** - loops through a block of code for each element in an array

```php
<?php
$ii=1;
while($ii<=5) {
    echo "The number is " . $ii . "<br />
    $ii++;
}?
</body>
</html>
```
**do...while Statement**

- The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.
- Syntax
  ```
  do
  {
  code to be executed;
  }
  while (condition);
  ```

**Example**

The following example will increment the value of i at least once, and it will continue incrementing the variable i while it has a value of less than 5:

```
<html>
<body>
<?php
$i=0;
do{
  $i++;
  echo "The number is " . $i . "<br />
;}
while ($i<5);
?></body>
</html>
```

**for Statement**

- The for statement has three parameters.
- The first parameter is for initializing variables,
- The second parameter holds the condition,
- The third parameter contains any increments required to implement the loop.
- If more than one variable is included in either the initialization or the increment section, then they should be separated by commas.
- The condition must evaluate to true or false.

**Example**

The following example prints the text "Hello World!" five times:

```
<html>
<body>
<?php
for ($i=1; $i<=5; $i++)
{
  echo "Hello World!\n";
}
?></body>
</html>
```

**foreach Statement**

Loops over the array given by the parameter. On each loop, the value of the current element is assigned to $value and the array pointer is advanced by one - so on the next loop, you'll be looking at the next element.

**Syntax**

```
foreach (array as value)
{
  code to be executed;
}
```
Arrays in PHP are quite versatile
- We can use them as we use traditional arrays, indexing on integer values
- We can use them as hashes, associating a key with a value in an arbitrary index of the array
- In either case we access the data via subscripts
  - In the first case the subscript is the integer index
  - In the second case the subscript is the key value
- We can even mix the two if we’d like

PHP Arrays

<?
$array[0] = 1;
$array[5] = "This is element 5";
$array[200] = 300;
$array['one'] = "One";
$b = $array;
$b does not overwrite $array

foreach ($array as $elem)
{
    echo "Element is $elem<br>
;
}
?>

PHP Arrays

Accessing Arrays – can be done in many ways
- We can use direct access to obtain a desired item
  - Good if we are using the array as a hash table or if we need direct access for some other reason
- For sequential access, the foreach loop was designed to work with arrays
  - Iterates through the items in two different ways
    - foreach ($arrayvar as $key => $value)
      - Gives both the key and value at each iteration
    - foreach ($arrayvar as $value)
      - Gives just the next value at each iteration

PHP Arrays

Accessing Arrays – can be done in many ways
- We can use direct access to obtain a desired item
  - Good if we are using the array as a hash table or if we need direct access for some other reason
- For sequential access, the foreach loop was designed to work with arrays
  - Iterates through the items in two different ways
    - foreach ($arrayvar as $key => $value)
      - Gives both the key and value at each iteration
    - foreach ($arrayvar as $value)
      - Gives just the next value at each iteration

PHP Arrays

Accessing Arrays – can be done in many ways
- We can use direct access to obtain a desired item
  - Good if we are using the array as a hash table or if we need direct access for some other reason
- For sequential access, the foreach loop was designed to work with arrays
  - Iterates through the items in two different ways
    - foreach ($arrayvar as $key => $value)
      - Gives both the key and value at each iteration
    - foreach ($arrayvar as $value)
      - Gives just the next value at each iteration

PHP Arrays

Accessing Arrays – can be done in many ways
- We can use direct access to obtain a desired item
  - Good if we are using the array as a hash table or if we need direct access for some other reason
- For sequential access, the foreach loop was designed to work with arrays
  - Iterates through the items in two different ways
    - foreach ($arrayvar as $key => $value)
      - Gives both the key and value at each iteration
    - foreach ($arrayvar as $value)
      - Gives just the next value at each iteration

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results
PHP Arrays

- each returns an array of two items:
  - A key field for the current key
  - A value field for the current value
- It returns the next (key,value) pair, then moves, so the first item is no longer a special case
  ```php
  while ($curr = each($a1)):
    $k = $curr["key"];
    $v = $curr["value"];
    echo "key is $k and value is $v <BR />
  endwhile;
  ```
  • This function is preferable to next() if it is possible that FALSE or an empty string or 0 could be in the array
  - The loop on the previous slide will stop for any of those values

PHP Control Structures

• Again, these are similar to those in C++ / Java
  - if, while, do, for, switch are virtually identical to those in C++ and Java
  - PHP allows for an alternative syntax to designate a block in the if, while, for and switch statements
    • Open the block with \{ rather than
    • Close the block with endif, endwhile, endfor, endswitch
    - Advantage to this syntax is readability
    - Now instead of seeing a number of close braces, we see different keywords to close different types of control structures

Example

The following example demonstrates a loop that will print the values of the given array:

```html
<html>
<body>
<?php
$arr=array("one", "two", "three");
foreach ($arr as $value) {
    echo "Value: "$ . $value . "<br />";
}
?></body>
</html>
```
Form Validation

```php
<?
}

else {
    mail( "kurt@ce.itu.edu.tr", "Feedback Form Results", $message, "From: $email" );
    header( "Location: form1.html" );
}
?>
```

Preventing the Browser From Caching

```
elseif (empty($email) || empty($message)) {
    header( "Expires: Mon, 20 Dec 1998 01:00:00 GMT" );
    header( "Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT" );
    header( "Cache-Control: no-cache, must-revalidate" );
    header( "Pragma: no-cache" );
}
```