Web Programming

Lecture 6 – Introduction to XML

Meta-markup Languages

- A markup language allows the user to identify individual elements of a document, e.g., what is a paragraph, a heading, or an unordered list.
- Used in combination with a style sheets, data can be properly presented on a web page, a slide show, or any other method that is appropriate for the data.

Meta-markup Languages

- A meta-markup language is a little different; it doesn't specify a document – it specifies a ***language***.
- SGML (***S***tandard ***G***eneralized ***M***arkup ***L***anguage) and XML (e***X***tensible ***M***arkup ***L***anguage) are examples of meta-markup languages.

SGML

- SGML was based on GML (***G***eneralized ***M***arkup ***L***anguage), which was developed at IBM in the 1960s. SGML was developed in 1974.
- SGML was intended to allow for the sharing of machine-readable documents.
- While it was used in the printing and publishing industry, its complexity kept it from wider use.
- SGML was used as the basis for HTML.

XML

- HTML describes the layout of information but conveys no information about its meaning. This limits the ability to retrieve information from an HTML document automatically.
- One solution to get around HTML's limitation is for groups of users to define and use their own set of tags and attribute and use a meta-markup language to implement them.
- XML is a simpler language than SGML and therefore more useful.

Using XML

- XML is not a replacement for XHTML. It is intended to provide a way to label data in a way that can be analyzed and manipulated automatically.
- XML is normally used together with a style sheet and an appropriate processor to produce a suitable XHTML document based on the XML file and the style sheet.

Syntax of XML

- XML has two levels of syntax:
  - The general low-level syntax within the XML doucment.
  - The higher-level syntax specified by DTD (Document Type Definitions) or XML schemas.
- XML documents can contain:
  - data elements of the document
  - markup declarations (instructions to the XML parser)
  - processing instructions (instructions for an application process that will process the data).

Elements of XML

- All XML documents begin with an XML declaration, which identifies the document as XML, and provides the version number of the XML standard being used and the encoding standard:

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

- Comments in XML are the same as in XHTML:

```
<!- This is a comment -->
```

Names in XML

- XML names are used to identify elements and attributes.
  - XML names must begin with a letter or an underscore and can contain letter, underscores, digits, hyphens and periods.
  - XML names are case sensitive; e.g., **`Body`**, **`body`** and **`BODY`** are three different names in XML.
  - There are no limits to the length of XML names.

Basic Syntax Rules

- Every XML documents defines a root element and that root element's tag must appear on the first line of XML code.
- All other elements must be nested within that element.
- For a XHTML document, the root tag is **`html`**.
- Every XML element must have a closing tag:
  - **`<myTag>`** … **`</myTag>`**
  - **`<myTag />`**

Sample XML Document

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

Another Sample XML Document

```
<?xml version = "1.0" encoding = "utf-8"?>
<bookstore>
 <book category="CHILDREN">
   <title>Harry Potter</title>
   <author>J K. Rowling</author>
   <year>2005</year>
   <price>29.99</price>
  </book>
  <book category="WEB">
   <title>Learning XML</title>
   <author>Erik T. Ray</author>
   <year>2003</year>
   <price>39.95</price>
  </book>
</bookstore>
```

XML Attributes

- In XML, attributes can be used to provide additional information about elements in an XML document.
- Example:

```
<file type = "gif"> computer.gif </file>
```

- Attributes must be enclosed in quotation marks (single or double)

```
<file type = 'gif'> computer.gif </file>
```

is also valid.

Attributes Or Nested Tags

- Is it better to add an additional attribute to an element or to define a nested element?
- Sometimes there is no choice – an image can only be an attribute (XML only handles text data).
- Nested tags can be added to any existing tag to describe its growing size and complexity – attributes give no information about this.

A Tag With One Attribute

```
<!-- A tag with one attribute -->
<patient name = "Maggie Dee Magpie ">
.... ….
</patient>
```

A Tag With One Nested Tag

```
<!-- A tag with one nested tag -->
<patient>
  <name> Maggie Dee Magpie
  </name>
.... ….
</patient>
```

An Extra Level Of Nested Tags

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

XML and Auxiliary Files

- An XML document often uses two auxiliary files:
  - One file specifies its tag set and structural syntactic rules. This can be either a DTD or an XML schema.
  - One file contains a style sheet to describe how the document’s content is to be printed and/or displayed. This can be either a Cascading Style Sheet or an XSLT Style Sheet.

XML Document Structure

- An XML document consists of one or more entities that are logically related sets of data.
- The document entity describes the document as a whole and is usually subdivided into other entities.
- These other entities may (or may not) be physically located in the same file.
- Entity names can be any length

Entity Names

- Entity names can be any length.
- They must begin with a letter, a dash or a colon.
- The remaining characters can be letters, digits, periods, dashes, underscores or colons.
- Adding an amersand before and asemi-colon after a reference name turns it into a reference.
  - **`&apple_image`** is a reference to the entity **`apple_image`**.

Character Data Sections

- When a document requires several predefined entities near each other, it becomes hard to read; therefore, we can use a character data section.
- Character data sections are not parsed and appear in an XML document as they are written.
- Character data sections cannot contain tags because they are considered literal text they do not mark up the document.

CDATA

- Their basic syntax is:
  - **`<![CDATA [`***content***`]]>`**
- An example:

```
<![CDATA [The last word of the line is >>> here <<<]]>
```

- This is clearly superior to writing:

```
The last word of the line is &gt; &gt;
&gt; here &lt; &lt; &lt;
```

- If I wrote

```
<![CDATA [The form of the tag is &lt; tag name &lt;]]>
```

I would get:

```
The form of the tag is &lt; tag name &lt;
```

Document Type Definitions

- A document type definition (DTD) is a set of rules specifying how a set of elements can appear in an XML document as well as entity declarations.
- While XML documents do not require DTDs, it allows the programmer to check an XML document for validity.
- A DTD can be internal (placed inside the XML document) or external (placed in a separate file that the XML document references).

DTD Syntax

- A DTD is a sequence of declarations:

  `<!keyword … >`
- There are 4 valid keywords:
  - **`ELEMENT`** – used to define tags
  - **`ATTLIST`** – used to define tag attributes
  - **`ENTITY`** – used to define entities
  - **`NOTATION`** – used to define data type notations

Declaring Elements

- Element declarations are a form that is similar to BNF.
- Each element declaration specifies the structure of ***<u>one</u>*** element, containing its names, its constituents (if it has child elements) or the data type of its parent (if it is a leaf).

Declaring Non-leaf Elements

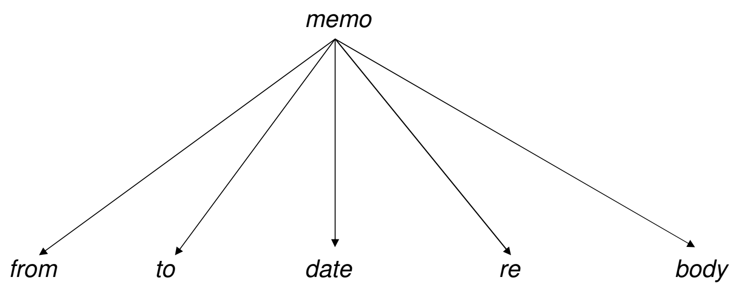
- The general form of an element declaration if there are child elements is:

  `<!ELEMENT ElementName (`*ChildElementList***`)>`**

- An example:

```
<!ELEMENT memo (from, to, date, re, body)>
```

Document Tree Structure



Child Element Specification Modifiers

- Normally, an element specification indicate one occurrence of an element.
- Using a modifier allows the programmer to have multiple occurrences of an element.

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

Declaration With Element Modifiers

```
<!ELEMENT person (parent+, age, spouse?, sibling*>
```

indicates

- One or more parents
- Age
- Spouse (optional)
- Zero or more siblings

Declaring a Leaf

- The syntax for an element that does not have child elements is:

  `<!ELEMENT` *element_name* **`(#PCDATA)>`**

- An example

```
<!ELEMENT year (#PCDATA)>
```

Declaring Attributes

- Attributes of an element are specified separately from the element declaration.
- An attribute declaration must include:
  - the attribute's name
  - the element to which it belongs
  - its type

Syntax for Attribute Declarations

- If the element has only one attribute it can be declared:
  `<! ATTLIST` *ElmntName AttribName AttribType [DefltVal]***>**

- Multiple attributes can be declared separately or in one declaration:

```
<! ATTLIST ElmntNm1 AttribNm1 AttribType1 DefltVal1
       ElmntNm2 AttribNm2 AttribType2 DefltVal2
              … …
       ElmntNmN AttribNmN AttribTypeN1 DefltValN    >
```

Possible Default Values

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

Declaring Attributes : An Example

- The declarations:

```
<! ATTLIST airplane places CDATA "4">
<! ATTLIST airplane engineType CDATA #REQUIRED>
<! ATTLIST airplane price CDATA #IMPLIED>
<! ATTLIST airplane manufactr CDATA #FIXED "Cessna">
```

- This is a valid XML element for this DTD:

```
<airplane places = "10" engineType = "jet">
  </airplane>
```

Declaring Entities

- General entities can be referenced anywhere in the content of an XML document.
- Parameter entities can only be referenced in DTDs.
- Syntax:

  `<!ENTITY` *`[%]`* *entityName* `"`*entityValue*`"`**`>`**

- If the percent sign is included, it is a parameter entity.

Referencing Entities

- An entity can be referenced in a declaration by placing an ampersand before and a semi-colon after the name:

```
<!ATTLIST airport airportName CDATA &jfk; >
```

External Text Entities

- An entity can be too long to be placed within the DTD; they can be pages long.
- This can be handled by placing it in a different file. These are called external text entities:

`<!ENTITY` *entityName* **`SYSTEM`** "*fileLocation*"**`>`**

A Sample DTD

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

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

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

```
<!ELEMENT location (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>

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

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

Internal DTDs

- DTDs can appear in the same file as the XML document or in a file of their own.
- If the DTD is included in the XML file, it's include in a **`DOCTYPE`** tag:

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE planes [
   All the declarations go here
   ]>
   The XML document goes here
```

External DTDs

- If the DTD is in its own file, the XML document contains a DOCTYPE tag identifying the DTD file
  <!DOCTYPE *XMLDocumentRootName* **SYSTEM** "*DTDFileName*"**>**
- In our case, it is
  <!DOCTYPE planes_for_sale SYSTEM "planes.dtd">

planes.xml

```
<?xml version = "1.0" encoding = "utf-8"?>
<!-- planes.xml - A document that lists ads for
                  used airplanes  -->

<!DOCTYPE planes_for_sales SYSTEM "planes.dtd">

<planes_for_sale>
  <ad>
    <year> 1977 </year>
    <make> &c; </make>
    <model> Skyhawk </model>
    <color> Light blue and white </color>
    <description>  New paint, nearly new interior,
              685 hours SMOH, full IFR King avionics
    </description>
    <price> 23,495 </price>
    <seller phone = "555-222-3333"> Skyway Aircraft
    </seller>
    <location>
      <city> Rapid City </city>
      <state> South Dakota </state>
    </location>
  </ad>

  <ad>
    <year> 1965 </year>
    <make> &p; </make>
    <model> Cherokee </model>
    <color> Gold </color>
    <description>  240 hours SMOH, dual NAVCOMs,
             DME,  new Cleveland brakes, great shape
    </description>
    <price> 23, 495 </price>
```

```
    <seller phone = "555-333-2222"
            email = "jseller@www.axl.com">
              John SellerSkyway Aircraft </seller>
    <location>
      <city> St. Joseph </city>
      <state> Missouri </state>
    </location>
  </ad>

</planes_for_sale>
```

Disadvantages of DTDs

- Since DTDs use a syntax different from XML, XML processors cannot analyze them.
- DTDs do not allow restrictions in the form of data that can appear in a tag.
  – DTDs have only 10 data types, none of which are numeric.

Displaying Raw XML Documents

- A browser will not know how to format an XML.
- It will display both the data and the tags, allowing subfields to be collapsed.
- It is important to be able to format XML data using Cascading Stylesheets or XSLT.

Formatting XML Documents

- There are two ways to provide format information to the browser for an XML document:
  - a Cascading Style Sheet (CSS) file
  - Extensible Stylesheet Language Transformations (XSLT)
- While not every browser supports XSLT, it has more power than CSS over the document's appearance.

Displaying XML Documents with CSS

- A CSS style sheet for XML has a list of element names, each followed by the element's attributes (and their values) delimited by braces.
- The only common style property that has not been discussed before is **`display`**, which can be **`inline`** (the default) or **`block`**. These determine if the element is to be displayed inline or in a separate block.
- To establish a connection between the style sheet and the XML file, add the following tag into the XML file:

```
<?xml-stylesheet type = "text/css"
                      href = "FileName.css">
```

planes.xml

```
<!-- planes.css - a style sheet for the
                       planes.xml document -->
ad { display: block; margin-top: 15px; color: blue;}
year, make, model { color: red; font-size: 16 pt;}
color  {display: block; margin-left: 20px;
                        font-size: 12pt; }
description {display:block; margin-left: 20px;
                             font-size: 12pt;}
price { dislay: block; color: green;
           margin-left: 10px; font-size: 12pt;}
seller { display: block; margin-left: 15px;
                            font-size: 14pt;}
location { display: block; margin-left: 40px;}
city  { font-size: 12pt;}
state { font-size: 12pt;}
```

planes.xml

```
<?xml version = "1.0" encoding = "utf-8"?>
<!-- planes.xml - A document that lists ads for
                  used airplanes  -->
<?xml-stylesheet type = "text/css" href =
  "planes.css" ?>
<!DOCTYPE planes_for_sales SYSTEM "planes.dtd">

<planes_for_sale>
  <ad>
    <year> 1977 </year>
    <make> &c; </make>
    <model> Skyhawk </model>
    <color> Light blue and white </color>
    <description>  New paint, nearly new interior,
             685 hours SMOH, full IFR King
  avionics
    </description>
    <seller phone = "555-333-2222"
            email = "jseller@www.axl.com">
              John Seller </seller>
    <location>
      <city> St. Joseph </city>
      <state> Missouri </state>
    </location>
  </ad>

</planes_for_sale>
```

Namespaces

- An XML namespace is a collection of element and attributes names used in XML documents.
- A namespace's name usually looks like a URI (***U***niform ***R***esource ***I***dentifier).
- Using namespaces allows you to give tags meaningful names that may conflict with tags that appear in other documents (e.g., HTML tags).

Why Use Namespaces?

- Markup for an XHTML table

```
<table>
  <tr> <td>Apples</td>
       <td>Bananas</td>
  </tr>
</table>
```

- Markup for data regarding furniture:

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

Declaring a Namespace

- When we write:

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

we are making the namespace defined at **`www.w3.org/1999/xhtml`** the default for the document.

- This tells the browser that the html document that follows (until the **`</html>`** tag) uses the tags located at that URI, where they are defined.

Declaring a Namespace With A Prefix

- To use multiple namespaces in the same document, we need to use an optional prefix:

```
<birds
  xmlns:bd="http://www.audobon.org/names/species">
```

- An element can have more than one namespace declared, but then all but one MUST have prefixes if more than one defines the same identifier:

```
<birds
  xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns:bd="http://www.audobon.org/names/species">
```

Using Namespaces

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

XSLT Style Sheets

- XSLT is one of three different XSLs (e***X***tensible ***S***tylesheet ***L***anguages) that can be used to transform XML documents into other forms, including XHTML documents.
- XSLT can be used to move, modify, sort XML elements and even to convert them into attrtibutes.
- Since XLST files are XML documents themselves, they can be validate using DTDs or transformed using XSLT stylesheets.

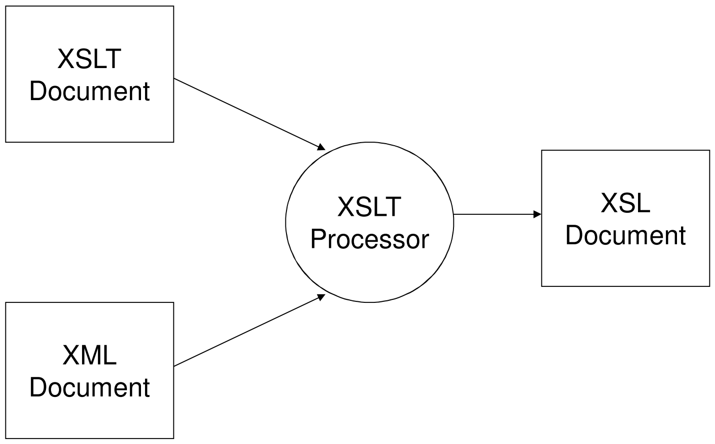
Overview of XSLT

- XSLT is a functional programming language, similar in some ways to LISP or Scheme.
- XSLT has functions, parameters, names to which values can be bound, selection constructs and conditional expressions capable of multiple selection.
- The XSLT processors takes an XSLT as its program, an XML document as input and produces an XSL files as output that can be saved or displayed in the browser.

An XSLT Document

- An XSLT document consists of one or more templates, using XPath (a related language) to describe element/attribute patterns in the XML input document.
- Each template has an associated section of XSLt code, which is executed when there is a match.
- The XSLT processor searches the XML document for template matches and is usually a "template-driven model."

XSLT Processing



XSL Transformations for Presentation

- This discussion of XSLt will be limited to basic formatting.
- We will assume that we are processing an XML document with an associated XSLT style sheet to produce an XHTML document.
- For an XML document to serve as data for an XSLT document, it must contain a processing instruction:

```
<?x-stylesheet type = "text/xsl"
                     href="XSLName" ?>
```

Original **people.xml**

```
<?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> P </middle_initial>
      <last_name> Feynman </last_name>
    </name>
    <profession> physicist </profession>
    <hobby> Playing the bongoes </hobby>
  </person>

</people>
```

The XSLT Style Sheet

- An XSLT style sheet is an XML document whose root tag is **`stylesheet`**. This tag must define the namespaces:

```
<xsl:stylesheet
 xmlns:xsl = http://w3.org/1999/XSL/Format />
```

person.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="person">
  A person
</xsl:template>
</xsl:stylesheet>
```

Rewriting **person.xml** to Read the Stylesheet

```
<?xml version="1.0"?>
<?xml-stylesheet type="application/xml"
                 href="people.xsl"?>
<people>
  <person born = "1912" died = "1954">
   … …
  </person>

  <person born = "1918" died = "1988">
    … …
  </person>
</people>
```

`person.xml` As Displayed

A person A person

`value-of`

- Generally, what you want as output is closely related to the text that is in the XML document that provides your input.
- **`xsl:value-of`** calculates the string value of an Xpath expression and inserts it into the output.
  – The value of an element is the text that it contains after removing the tags and resolving the entity and cahracter references.

Using **`value-of`**

```
<?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:value-of select = "name" />
</xsl:template>
</xsl:stylesheet>
```

Using **`value-of`**: An Example

```
Alan  Turing  Richard  Feynman
```

Using **`apply-templates`**

- By default, an XSLT processor reads the input XML document from top to bottom, doing a preorder traversal of the document from the root.
- **`xsl:apply-templates`** allows the user to make the processing explicit, changing it if one wishes.

Why Use **`apply-templates`**

- Using in our earlier example

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

  would still result middle initials, hobbies and professional being outputted.
- We can limit the output to first and last name by writing:

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

people.xsl With **apply-templates**

```
<?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">
<html lang = "en">
  <head>
    <title> Famous Scientists</title>
    <meta charset = "utf-8" />
  </head>
  <body>
<h1> Famous People </h1>
<table border = "1">
```

```
<xsl:apply-templates />
    </table>
  </body>
</html>
</xsl:template>

<xsl:template match="person">
   <tr> <td>
     <xsl:apply-templates select="name" />
  </td> <td>
  <xsl:value-of select = "profession" />
  </td> </tr>
</xsl:template>
```

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

</xsl:stylesheet>
```

The Output

Alan Turing	computer scientist
Richard Feynman	**physicist**

Famous Scientists

Alan Turing	computer scientist
Richard Feynman	physicist

`xsl:for-each`

- The **`<xsl:for-each>`** element allows you to do looping in XSLT.
- The XSL **`<xsl:for-each>`** element can be used to select every XML element of a specified node-set.

mycds.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="application/xml" href="mycds.xsl"?>
<catalog>
      <cd>
            <title>Empire Burlesque</title>
            <artist>Bob Dylan</artist>
            <country>USA</country>
            <company>Columbia</company>
            <price>10.90</price>
            <year>1985</year>
      </cd>
      <cd>
            <title>Hide your heart</title>
            <artist>Bonnie Tyler</artist>
            <country>UK</country>
            <company>CBS Records</company>
            <price>9.90</price>
            <year>1988</year>
      </cd>
```

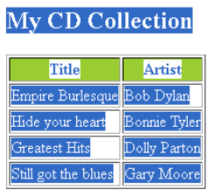
```
      <cd>
            <title>Greatest Hits</title>
            <artist>Dolly Parton</artist>
            <country>USA</country>
            <company>RCA</company>
            <price>9.90</price>
            <year>1982</year>
      </cd>
      <cd>
            <title>Still got the blues</title>
            <artist>Gary Moore</artist>
            <country>UK</country>
            <company>Virgin records</company>
            <price>10.20</price>
            <year>1990</year>
      </cd>
</catalog>
```

mycds.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
```

```
      <xsl:for-each select="catalog/cd">
      <tr>
        <td><xsl:value-of select="title" /></td>
        <td><xsl:value-of select="artist" /></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Using `for-each`



My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore

XML Documents and Their Nodes

- There are seven different types of nodes that XML documents can have. These include:
  - Root nodes
  - Element nodes
  - Attribute nodes
- There are built-in template rules for these different types of nodes.
- For attribute nodes, matching an attribute type will lead to outputting the value of the attribute, not its name

`people.xsl` Checking Attributes

```
<?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">
  <html>
    <body>
      <dl>
        <xsl:apply-templates />
      </dl>
    </body>
  </html>
</xsl:template>
```

```
<xsl:template match="person">
  <dt> <xsl:apply-templates select="name" /> </dt>
  <dd> <ul>
    <li>Born: <xsl:apply-templates select="@born" />
</li>
    <li>Died: <xsl:apply-templates select="@died" />
</li>
  </ul> </dd>
</xsl:template>
</xsl:stylesheet>
```

```
    <price> 23,495 </price>
    <seller phone = "555-222-3333"> Skyway Aircraft
  </seller>
    <location>
      <city> Rapid City </city>
      <state> South Dakota </state>
    </location>
  </ad>

  <ad>
    <year> 1965 </year>
    <make> &p; </make>
    <model> Cherokee </model>
    <color> Gold </color>
    <description>  240 hours SMOH, dual NAVCOMs,
              DME, new Cleveland brakes, great shape
    </description>
    <price> 23,495 </price>
```