XSLT

XSLT Overview

- **What is XSLT**?
  - XSL is the *Extensible Style Language*.
  - It has two parts: the *transformation language* XSLT and the *formatting language XSL - FO*.
  - XSLT provides a syntax for defining rules that transform an XML document to another document.
    - For example, to an HTML document.
  - An XSLT "style sheet" consists primarily of a set of template ***rules*** that are used to transform nodes matching some ***patterns***.

Why transform XML?

XML is a success because it is designed

- for separation between content and presentation
- as a format for EDI
- as human readable/writable format

Transforming XML is not only desirable, but necessary.

XML attempts to fulfill this need by supporting

- publishing data (not necessarily XML)
- conversion between two proprietary formats

What is XSL?

- XSL is a language that allows one to describe a browser how to process an XML file.
- XSL can convert an XML file into another XML with different format.
- XSL can convert an XML file into a non-XML file.

XSL

- The most common type of XSL processing is to convert XML file into HTML file which can be displayed by browsers.
- XSL is the bridge between XML and HTML.
- We can use XSL to have different HTML formats for the same data represented in XML.
- Separating data (contents) from style tags (display commands).

XSLT Overview

- Example of XML document

```
<?xml version=”1.0”?>
<?xml-stylesheet type=”text/xml” href=”planets.xsl”?>
<PLANETS>

  <PLANET>
    <NAME>Mercury</NAME>
    <MASS UNITS=”(Earth = 1)”>.0553</MASS>
    <DAY UNITS=”days”>58.65</DAY>
    <RADIUS UNITS=”miles”>1516</RADIUS>
    <DENSITY UNITS=”(Earth = 1)”>.983</DENSITY>
    <DISTANCE UNITS=”million miles”>43.4</DISTANCE><!--At perihelion-->
  </PLANET>
```

XSLT Overview

- XML document example

```
<PLANET>
    <NAME>Venus</NAME>
    <MASS UNITS="(Earth = 1)">.815</MASS>
    <DAY UNITS="days">116.75</DAY>
    <RADIUS UNITS="miles">3716</RADIUS>
    <DENSITY UNITS="(Earth = 1)">.943</DENSITY>
    <DISTANCE UNITS="million miles">66.8</DISTANCE><!--At
  perihelion-->
</PLANET>

<PLANET>
    <NAME>Earth</NAME>
    <MASS UNITS="(Earth = 1)">1</MASS>
    <DAY UNITS="days">1</DAY>
    <RADIUS UNITS="miles">2107</RADIUS>
    <DENSITY UNITS="(Earth = 1)">1</DENSITY>
    <DISTANCE UNITS="million miles">128.4</DISTANCE><!--At
  perihelion-->
</PLANET>
</PLANETS>
```

XSLT Overview

- Example of a style sheet planet.xsl

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

   <xsl:template match=”PLANETS”>
      <HTML>
         <xsl:apply-templates/>
      </HTML>
   </xsl:template>

   <xsl:template match=”PLANET”>
      <P>
         <xsl:value-of select=”NAME”/>
      </P>
   </xsl:template>

</xsl:stylesheet>
```

XSLT Overview

- Result

```
<HTML>

    <P>Mercury</P>

    <P>Venus</P>

    <P>Earth</P>

</HTML>
```

XSLT Overview

- The xml-stylesheet element in the XML instance references an XSL style sheet.
- In general, children of the stylesheet element in a stylesheet are ***templates***.
- A template specifies a ***pattern***; the template is applied to nodes in the XML source document that ***match*** this pattern.
  - Note: the pattern "/" matches the root node of the document
- In the transformed document, the *body* of the template element replaces the matched node in the source document.
- In addition to text, the body may contain further XSL terms, e.g.:
  - **xsl:value-of** extracts data from selected sub-nodes.

XSLT Overview

- We have an XML document and the style sheet (or rules) to transform it. So, how do you transform the document?.
- You can transform documents in three ways:
  - **In the server**. A server program, such as a Java servlet, can use a style sheet to transform a document automatically and serve it to the client. Example, XML Enabler, which is a servlet that you'll find at XML for Java Web site, www.alphaworks.ibm.com/tech/xml4j
  - **In the client.** An XSL-enabled browser may convert XML downloaded from the server to HTML, prior to display. Currently Internet Explorer supports a subset of XSLT.
  - **In a standalone program**. XML stored in or generated from a database, say, may be "manually" converted to HTML before placing it in the server's document directory.
- In any case, a suitable program takes an XML document as input, together with an XSLT "style-sheet".

Format of Style Sheet

- XSLT style sheet is itself an XML document.
- XSLT elements are referred from the namespace http://www.w3.org/1999/XSL/Transform
  - As a matter of convention we use the prefix **xsl:** for this namespace.
- The document root in an XSLT style sheet is an xsl:stylesheet element, e.g.:

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

  - A synonym for **xsl:stylesheet** is **xsl:transform.**
- Several kinds of element can be nested inside xsl:stylesheet, but by far the most important is the xsl:template element.

The **xsl:template** element

- When you match or select nodes, a template tells the XSLT processor how to transform the node for output
- So all our templates will have the form:

```
<xsl:template  match="pattern">
    template body
</xsl:template>
```

- The *pattern* is an Xpath expression describing the nodes to which the template can be applied.
- The processor scans the input document for nodes matching this pattern, and replaces them with the text included in the *template body*.

An input document

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xml" href="eg.xsl"?>
<planets>
   <planet>
      <name>Mercury</name>
      <mass>0.0553</mass>
      <day units="days">58.65</day>
      <radius units="miles">1516</radius>
      <density>0.983</density>
   </planet>
   <planet>
      <name>Venus</name>
      <mass>0.815</mass>
      <day units="days">116.75</day>
      <radius units="miles">3716</radius>
      <density>0.943</density>
   </planet>
   <planet>
      <name>Earth</name>
      <mass>1</mass>
      <day units="days">1</day>
      <radius units="miles">2107</radius>
      <density>1</density>
   </planet>
</planets>
```

Using an empty style sheet

- Consider the example where there are no templates explicitly specified, eg.xsl has the form:

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

- The transformation of the input document is:

Mercury0.055358.6515160.983Venus0.815116.7537160.943Earth1121071

i.e. just the concatenated string values in all text nodes.

- This happens because there is a default template rule:

```
<xsl:template match="text()">
    <xsl:value-of  select="."/>
</xsl:template>
```

Templates without embedded XSLT

- Now consider a single template, with no embedded XSLT commands:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0”
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 >
    <xsl:template match="planet">
        <p>planet discovered</p>
    </xsl:template>
</xsl:stylesheet>
```

- The transformation of the input document is:

```
<?xml version="1.0" encoding="UTF-16"?><p>planet
discovered</p><p>planet discovered</p><p>planet discovered</p>
```

This is valid HTML, but not very readable (as text).

- We can add the command:

```
<xsl:output indent="yes"/>
```

to the xsl:stylesheet element to get prettier output formatting.

The **xsl:apply-templates** element

- Suppose a second template matching the planets element is added:

```
<xsl:template match="planet">
    <p>planet discovered</p>
</xsl:template>

<xsl:template match="planets">
    <h1>All  Known Planets</h1>
</xsl:template>
```

- The output now only contains the header:

```
<h1>All  Known Planets</h1>
```

  not the “planet discovered” messages from processing the nested planet elements.

- Once a match is found, nested elements are not processed unless there is an explicit <xsl:apply-templates> instruction:

```
<xsl:template match="planets">
    <h1>All  Known Planets</h1>
    <xsl:apply-templates/>
</xsl:template>
```

The **xsl:value-of** element

- We can now match arbitrary nodes in the source document, but we don't yet have a way to extract data from those nodes.
- To do this we need the xsl:value-of element, e.g.:

```
<xsl:template match="planet">
    <p>planet <xsl:value-of  select="name"/>
discovered</p>
</xsl:template>

<xsl:template match="planets">
    <h1>All  Known Planets</h1>
    <xsl:apply-templates/>
</xsl:template>
```

- We now get the more interesting output:

```
<h1>All  Known Planets</h1>
<p>planet Mercury discovered</p>
<p>planet Venus discovered</p>
<p>planet Earth discovered</p>
```

Selections

- The **select** attribute of the **xsl:value-of** element is a general Xpath expression.
- Its result—which may be a node set or other allowed value—is converted to a string and included in the output.
- For example, the selection can be an attribute node, a set of elements, or it could be the result of a numeric computation.
- If the selection is a set of elements, the text contents of all the element bodies, including nested elements, are concatenated and returned as the value.

Xpath expressions in attributes

- Suppose we want to generate an XML element in the output with an attribute whose value is computed from source data.
- One might be tempted to try a template like:

```
<planet name = "<xsl:value-of  select='name'/>" >
    Status: discovered
</planet>
```

- This is ill-formed XML: we cannot have an XML element as an attribute value.
- Instead **{}**s can be used in an attribute value to surround an Xpath expression:

```
<planet name = "{name}" >
    Status: discovered
</planet>
```

- The Xpath expression **name** is evaluated exactly as for a select attribute, and interpolated into the attribute value string.

The **xsl:element** element

- For similar reasons we cannot use **<xsl:value-of>** to compute an expression that is used as the ***name of an element*** in the generated file.
- Instead one can use instead the **xsl:element** element.
  - These can optionally include nested **xsl:attribute** elements (as their first children):

```
<xsl:template match="planet">
  <xsl:element name="{name}">
      <xsl:attribute name="distance">
         <xsl:value-of select="distance"/>
      </xsl:attribute>
      Status: discovered
  </xsl:element>
</xsl:template>
```

  - When this template matches a **planet**, it generates an XML element whose name is the planet, with a **distance** attribute.

A Table of Planets

```
<xsl:template match="planets">
    <html><body>
    <h1>All  Known Planets</h1>
    <table width="100%" align="center" border="1">
      <tr><th>name</th><th>mass</th><th>density</th>
            <th>radius</th></tr>
      <xsl:apply-templates/>  <!-- rows of table -->
      <tr>
        <td>AVERAGES</td>
        <td>
          <xsl:value-of
            select="sum(planet/density) div count(planet)"/>
        </td>
        <td></td>
        <td></td>
      </tr>
    </table>
    </body></html>
  </xsl:template>
```

A row of the table

```
<xsl:template match="planet">
   <tr>
      <td><xsl:value-of select="name"/></td>
      <td><xsl:value-of select="mass"/></td>
      <td><xsl:value-of select="density"/></td>
      <td><xsl:value-of select="radius"/></td>
   </tr>
</xsl:template>
```

The display

All Known Planets

name	mass	density	radius
Mercury	0.0553	0.983	1516
Venus	0.815	0.943	3716
Earth	1	1	2107
AVERAGES	0.9753333333333334		

XSLT Examples

Sample XML

```
<person>
    <name first="Neil"
    last="Armstrong" />
    <quote>...one giant leap for
    mankind.</quote>
</person>
```

Sample XSLT

```
<xsl:stylesheet version="1.0" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text" />
    <xsl:template match="/person">
        <xsl:value-of select="concat(
            name/@first, ' ', name/@last)" />
        said "
        <xsl:value-of select="quote" />"
    </xsl:template>
</xsl:stylesheet>
```

Sample Output

```
Neil Armstrong said "...one giant
    leap for mankind."
```

XSLT Examples

Sample XML

```
<person>
    <name first="Neil"
    last="Armstrong" />
    <quote>...one giant leap for
    mankind. </quote>
</person>
```

Sample XSLT ➔ HTML

```
<xsl:stylesheet version="1.0" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" />
    <xsl:template match="/person">
        <html><head><title>
            <xsl:value-of select="concat(
                name/@first, ' ', name/@last)" />
        </title></head><body>
        <blockquote><xsl:value-of
            select="quote" /></blockquote>
        </body></html>
    </xsl:template>
</xsl:stylesheet>
```

All well-formed

Sample Output

```
<html><head><title>Neil
    Armstrong</title>
    </head>
<body><blockquote>…one giant
    leap for mankind.</blockquote>
</body></html>
```

XSLT Examples

Sample XML

```
<person>
    <name first="Neil"
    last="Armstrong" />
    <quote>...one giant leap for
    mankind. </quote>
</person>
```

Sample Output

```
<quote><speaker firstname="Neil"
    lastname="Armstrong"/>
    <text>…one giant leap for
    mankind.</text>
</quote>
```

Sample XSLT ➔ XML

```
<xsl:stylesheet version="1.0" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" />
    <xsl:template match="/person">
        <quote><speaker
          firstname="{name/@first}"
          lastname="{name/@last}"/>
        <text><xsl:value-of select="quote"
          /></text>
        </quote>
    </xsl:template>
</xsl:stylesheet>
```

XSLT Loops: for-each

Sample XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

Sample XSLT ➔ HTML

```
<xsl:stylesheet version="1.0" xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" />
    <xsl:template match="/zoo">
        <html><head><title>Zoo</title>
          </head><body>
        <xsl:for-each select="*">
            <h1>
                <xsl:value-of select="name(.)" />
            </h1>
        </xsl:for-each>
        </body></html>
    </xsl:template>
</xsl:stylesheet>
```

XSLT Loops: for-each

Sample XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

Result HTML

```
<html><head><title>Zoo</title>
  </head><body>
    <h1>
      birds
    </h1>
    <h1>
      mammals
    </h1>
</body></html>
```

XSLT Loops: for-each

Sample XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

Sample XSLT ➔ HTML

```
<xsl:stylesheet version="1.0" xmlns:xsl=
   "http://www.w3.org/1999/XSL/Transform">
     <xsl:output method="html" />
     <xsl:template match="/zoo">
        <html><head><title>Zoo</title>
          </head><body>
        <xsl:for-each select="*">
             <h1><xsl:value-of select="name(.)" /></h1>
             <ul><xsl:for-each select="*">
                  <li><xsl:value-of select="name(.)" />
                   (<xsl:value-of select="@pop">)</li>
             </xsl:for-each></ul>
        </xsl:for-each>
        </body></html>
     </xsl:template>
</xsl:stylesheet>
```

XSLT Loops: for-each

Sample XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

Result HTML

```
<html><head><title>Zoo</title>
  </head><body>
    <h1>birds</h1>
    <ul>
      <li>albatross (4)</li>
      …
    </ul>
    <h1>mammals</h1>
    <ul>
      <li>aardvark (5)</li>
      …
    </ul>
</body></html>
```

XSLT Loops: apply-templates

Sample XSLT

```
<xsl:stylesheet version="1.0" xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" />
   <xsl:template
    match="/zoo">
      <html>
      <head><title>Zoo</title></head>
      <body>
        <xsl:apply-
        templates
        select="*" />
      </body>
      </html>
   </xsl:template>
```

```
<xsl:template
 match="birds | mammals">
   <h1><xsl:value-of select="name(.)"
     /></h1>
   <ul>
       <xsl:for-each select="*">
           <li><xsl:value-of
             select="name(.)" />
             (<xsl:value-of
             select="@pop"/>)</li>
       </xsl:for-each>
   </ul>
</xsl:template>
</xsl:stylesheet>
```

XSLT Decisions: if

XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

XSLT

```
…
<xsl:template match="birds | mammals">
   <h1><xsl:value-of select="name(.)" /> </h1>
   <p>We have more than 2...</p>
   <xsl:if test="*[@pop &gt; 2]">
      <ul>
         <xsl:for-each select="*[@pop &gt; 2]">
            <li><xsl:value-of select="name(.)" /></li>
         </xsl:for-each>
      </ul>
   </xsl:if>
</xsl:template>
…
```

XSLT Decisions: if

Sample XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

Result HTML

```
<html><head><title>Zoo</title></head><body>
    <h1>birds</h1>
    <p>We have more than 2...</p>
    <ul>
      <li>albatross</li>
      <li>chickadee</li>
    </ul>
    <h1>mammals</h1>
    <p>We have more than 2...</p>
    <ul>
      <li>aardvark</li>
      <li>bat</li>
    </ul>
</body></html>
```

XSLT Decisions: choose

XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

XSLT fragment

```
<xsl:template match="birds | mammals">
  <ul><xsl:for-each select="*">
    <li><xsl:value-of select="name(.)" />
      (<xsl:choose>
        <xsl:when test="@pop = 2">a
         couple</xsl:when>
        <xsl:when test="@pop &lt;= 5">a
         few</xsl:when>
        <xsl:otherwise>many</xsl:otherwise>
      </xsl:choose>)
    </li>
  </xsl:for-each></ul>
</xsl:template>
```

XSLT Decisions: choose

Sample XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

Result HTML

```
<html><head><title>Zoo</title></head><body>
    <ul>
      <li>albatross (a few)</li>
      <li>buzzard (a couple)</li>
      <li>chickadee (many)</li>
    </ul>
    <ul>
      <li>aardvark (a few)</li>
      <li>bat (many)</li>
      <li>cheetah (a couple)</li>
    </ul>
</body></html>
```

XSLT Variables: variable

XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

XSLT fragment

```
<xsl:template match="birds | mammals">
  <xsl:variable name="total-animals"
   select="sum(*/@pop)" />
  <ul><xsl:for-each select="*">
    <li><xsl:value-of select="name(.)" />
      (<xsl:value-of select="round(100 * @pop
       div $total-animals)"/>% of all <xsl:value-
       of select="name(..)"/>)
    </li>
  </xsl:for-each></ul>
</xsl:template>
```

XSLT Variables: variable

Source XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

Result HTML

```
<html><head><title>Zoo</title></head><body>
    <ul>
      <li>albatross (22% of all birds)</li>
      <li>buzzard (11% of all birds)</li>
      <li>chickadee (67% of all birds)</li>
    </ul>
    <ul>
      <li>aardvark (2% of all mammals)</li>
      <li>bat (97% of all mammals)</li>
      <li>cheetah (1% of all mammals)</li>
    </ul>
</body></html>
```

XSLT Variables: parameter

XSLT fragments

```
<xsl:template match="birds |
 mammals">
  <xsl:variable name="total-
   animals"
   select="sum(*/@pop)" />
  <ul><xsl:for-each
   select="*">
    <xsl:call-template
     name="animal">
     <xsl:with-param
     name="total"
     value="$total-animals">
    </xsl:call-template>
  </xsl:for-each></ul>
```

```
<xsl:template name="animal">
  <xsl:param name="total" />
  <li>
    <xsl:value-of select="name(.)" />
     (<xsl:value-of select="round(100 *
     @pop div $total)"/>%)
  </li>
</xsl:template>
```

XSLT Variables: param

XSLT fragments

```
<xsl:template match="birds |
 mammals">
  <xsl:variable name="total-
   animals"
   select="sum(*/@pop)" />
  <ul><xsl:for-each
   select="*">
    <xsl:call-template
     name="animal">
     <xsl:with-param
     name="total"
     value="$total-animals">
    </xsl:call-template>
  </xsl:for-each></ul>
</xsl:template>
```

```
<xsl:template name="animal">
   <xsl:param name="total" />
   <li>
      <xsl:value-of select="name(.)" />
        (<xsl:value-of select="round(100 *
        @pop div $total)"/>%)
   </li>
</xsl:template>
```

XSLT Variables: param

XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

XSLT fragment

```
<xsl:param name="minimum-population" select="5" />
<xsl:template match="birds | mammals">
  <ul>
  <xsl:for-each select="*[@pop &gt;=
   $minimum-population]">
    <li><xsl:value-of select="name(.)" />
      (<xsl:value-of select="@pop"/>)
    </li>
  </xsl:for-each></ul>
</xsl:template>
```

XSLT Variables: param

Source XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

Result HTML

```
<html><head><title>Zoo</title></head><body>
    <ul>
      <li>chickadee (12)</li>
    </ul>
    <ul>
      <li>aardvark (5)</li>
      <li>bat (200)</li>
    </ul>
</body></html>
```

XSLT Extras: sort

XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

XSLT fragment

```
<xsl:template match="birds | mammals">
  <ul>
  <xsl:for-each select="*">
    <xsl:sort select="@pop" order="descending" data-
      type="number" />
    <li><xsl:value-of select="name(.)" />
      (<xsl:value-of select="@pop"/>)
    </li>
  </xsl:for-each></ul>
</xsl:template>
```

XSLT Extras: sort

Source XML

```
<zoo>
  <birds>
    <albatross pop="4" />
    <buzzard pop="2" />
    <chickadee pop="12" />
  </birds>
  <mammals>
    <aardvark pop="5" />
    <bat pop="200" />
    <cheetah pop="2" />
  </mammals>
</zoo>
```

Result HTML

```
<html><head><title>Zoo</title></head><body>
    <ul>
      <li>chickadee (12)</li>
      <li>albatross (4)</li>
      <li>buzzard (2)</li>
    </ul>
    <ul>
      <li>bat (200)</li>
      <li>aardvark (5)</li>
      <li>cheetah (2)</li>
    </ul>
</body></html>
```

XSLT Extras: copy-of

Source XML

```
<person>
    <name first="Neil"
    last="Armstrong" />
    <quote>...one giant leap for
    mankind. </quote>
</person>
```

XML Output

```
<quote><name first="Neil"
    last="Armstrong"/>
    <text>…one giant leap for
    mankind.</text>
</quote>
```

Sample XSLT ➔ XML

```
<xsl:stylesheet version="1.0" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" />
    <xsl:template match="/person">
        <quote><xsl:copy-of select="name"
          />
        <text><xsl:value-of select="quote"
          /></text>
        </quote>
    </xsl:template>
</xsl:stylesheet>
```

XSLT Extras: more elements

- xsl:text – writes literal text to the output (useful for controlling whitespace or forcing exact, unescaped output)

- xsl:import and xsl:include – used to combine stylesheets (useful for XSLT libraries)

XSLT Extras: more functions

- document(*url*) – opens an XML document at the given location, returning its nodes as data that can be used by the template
  Example:
  document('zoo.xml')/zoo//*[@pop = 2]
- current() – similar to "." except that it always refers to the current node, even when used inside predicates
  Example:
  //*[@pop = current()/@pop]

Why/When XSLT?

- Web Standard
  - XSLT v 1.0 Recommended by W3C almost 10 years ago (it's *stable* and *well understood*)
  - Dozens of implementations and host languages (Java, .NET, PHP, C++…)
  - Wide tool support (Firefox, IE, Visual Studio, many text editors, full IDEs…)
  - No vendor lock-in

Why/When XSLT?

- Very good at converting XML to XML, XHTML/HTML, or plain text
  - Makes it easy to keep things well-formed
  - Uses XML syntax, so the only real new syntax is XPath (which is also used elsewhere)
  - XPath is far more compact than similar DOM code in JS/Java/C#/etc.
  - Can be interpreted (for quick development) or compiled (for maximum performance)

References

- Inside XML, Chapter 13: “XSL Transformations”.
- “XSL Transformations (XSLT)”, version 1.0:

  http://www.w3.org/TR/xslt
- “XML Path Language (XPath)”, version 1.0:

  http://www.w3.org/TR/xpath
- Nancy McCracken, Ozgur Balsoy
  - http://aspen.csit.fsu.edu/webtech/xml/

XML and Related Acronyms

- ***Document Type Definition* (DTD),** which defines the tags and their relationships
- ***Extensible Style Language* (XSL)** style sheets, which specify the presentation of the document
- **Cascading Style Sheets(CSS)** less powerful presentation technology without tag mapping capability
- ***XPATH*** which specifies location in document
- ***XLINK and XPOINTER*** which defines link-handling details
- ***Resource Description Framework* (RDF),** document metadata
- ***Document Object Model* (DOM),** API for converting the document to a tree object in your program for processing and updating
- ***Simple API for XML* (SAX),** "serial access" protocol, fast-to-execute protocol for processing document on the fly
- ***XML Namespaces***, for an environment of multiple sets of XML tags
- ***XHTML***, a definition of HTML tags for XML documents (which are then just HTML documents)
- ***XML schema***, offers a more flexible alternative to DTD