

Extensible Stylesheet Language (XSL & XSLT)

What is XSL?

- XSL = eXtensible Stylesheet Language
- XSL consists of
 - XPath (navigation in documents)
 - XSLT (T for *transformations*)
 - XSLFO (FO for *formatting objects*)
 - This is a rather complex language for typesetting (i.e., preparing text for printing)
 - Will not be covered in this course

What is XSLT

Used for Transforming XML documents into other XML documents

XSLT Stylesheet

- An XSLT stylesheet is a program that transforms an XML document into another XML document
- For example:
 - Transforming XML to XHTML (HTML that conforms to XML syntax)
 - Transforming an XML document to WML (a format of XML that cellular phones can display)

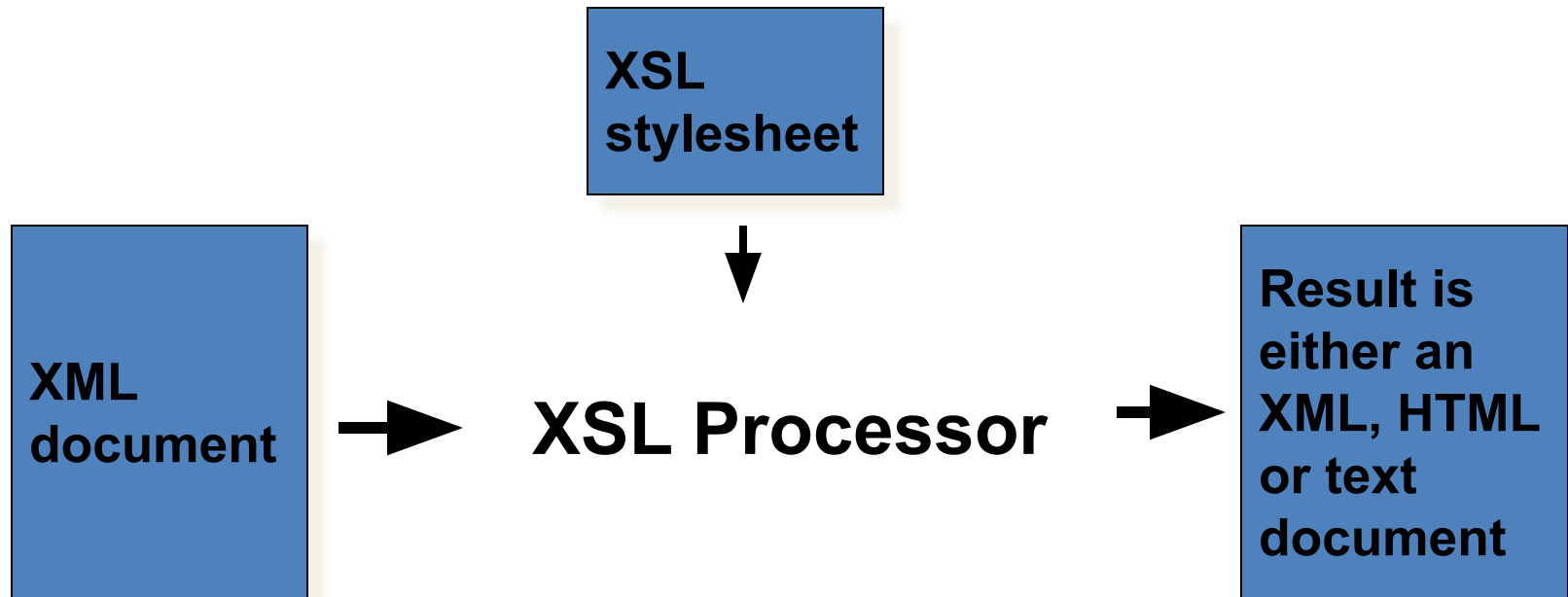
XSL Features

- XSL is a high-level, functional language
- An XSL style sheet is a valid XML document
(i.e Valid with respect to the XSL namespace)
- Commands in XSL are *XSL elements*

Applying XSLT Stylesheets to XML Documents

- There are three ways of applying an XSLT stylesheet to an XML document
 - Directly applying an *XSLT processor* to the XML document and the XSLT stylesheet
 - Calling an XSLT processor from within a (Java) program
 - Adding to the XML document a link to the XSL stylesheet and letting the browser do the transformation

Using an XSL Processor



Directly applying the Xalan XSL processor

```
java org.apache.xalan.xslt.Process  
-IN myXmlFile.xml -XSL myXslFile.xsl  
-OUT myOutputFile.html
```

Online Tool

XML Transform online Tool

Example of an XML document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="UK">
    <title>Dark Side of the Moon</title>
    <artist>Pink Floyd</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Space Oddity</title>
    <artist>David Bowie</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Aretha: Lady Soul</title>
    <artist>Aretha Franklin</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

Letting a Browser Perform the Transformation

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

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

```
<catalog>
```

```
  <cd country="UK">
```

```
    <title>Dark Side of the Moon</title>
```

```
    <artist>Pink Floyd</artist>
```

```
    <price>10.90</price>
```

```
  </cd>
```

```
  ...
```

```
</catalog>
```



A diagram illustrating the transformation process. A blue box at the bottom contains the text "A link to the stylesheet". An arrow points from this box to the `href="catalog.xsl"?` part of the `<?xml-stylesheet>` declaration in the XML code above.

The Root of the XSL Document

- The Root of the XSL document should be one of the following lines:

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

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

The namespace allows the XSL processor to distinguish between XSL tags and tags of the result document

How Does XSLT Work?

- An XSL stylesheet is a collection of *templates* that are applied to *source nodes* (i.e., nodes of the given XML document)
- Each template has a *match* attribute that specifies to which source nodes the template can be applied
- The *current* source node is *processed* by applying a template that matches this node
- Processing always starts at the root (/)

Templates

- A template has the form

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

`...`

`</xsl:template>`

- The content of a template consists of
 - XML elements and text that are copied to the result
 - XSL elements that are actually instructions
- The pattern syntax is a subset of XPath

```
<?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>
```

```
<h1>Hello World</h1>
```

```
</body>
```

```
</html>
```

```
</xsl:template>
```

```
</xsl:stylesheet>
```



```
<html>  
<body>  
<h1>Hello World</h1>  
</body>  
</html>
```

Applying a browser to catalog.xml
(catalog.xml has a link to catalog.xsl)

The Element

`<xsl:apply-templates>`

- Processing starts by applying a template that matches the root (/)
 - If the given XSL stylesheet does not have a template that matches the root, then one is inserted by default (see the slide on “Default Templates”)
- The XSL stylesheet must specify explicitly whether templates should be applied to descendants of the root
- It is done by putting inside a template the instruction:
`<xsl:apply-templates select="xpath" />`
- Without the select attribute, this instruction processes all the children of the current node



```
<?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>
      <xsl:apply-templates select="catalog/cd"/>
    </body>
  </html>
</xsl:template>
<xsl:template match="cd">
  <h2>A CD!</h2>
</xsl:template>
</xsl:stylesheet>
```

```
<html>
<body>
<h2>A CD!</h2>
<h2>A CD!</h2>
<h2>A CD!</h2>
</body>
</html>
```



Default Templates

- XSL provides implicit built-in templates that match every element and text nodes

```
<xsl:template match="/ | *">
  <xsl:apply-templates/>
</xsl:template>

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

- Templates we write always override these built-in templates (when they match)

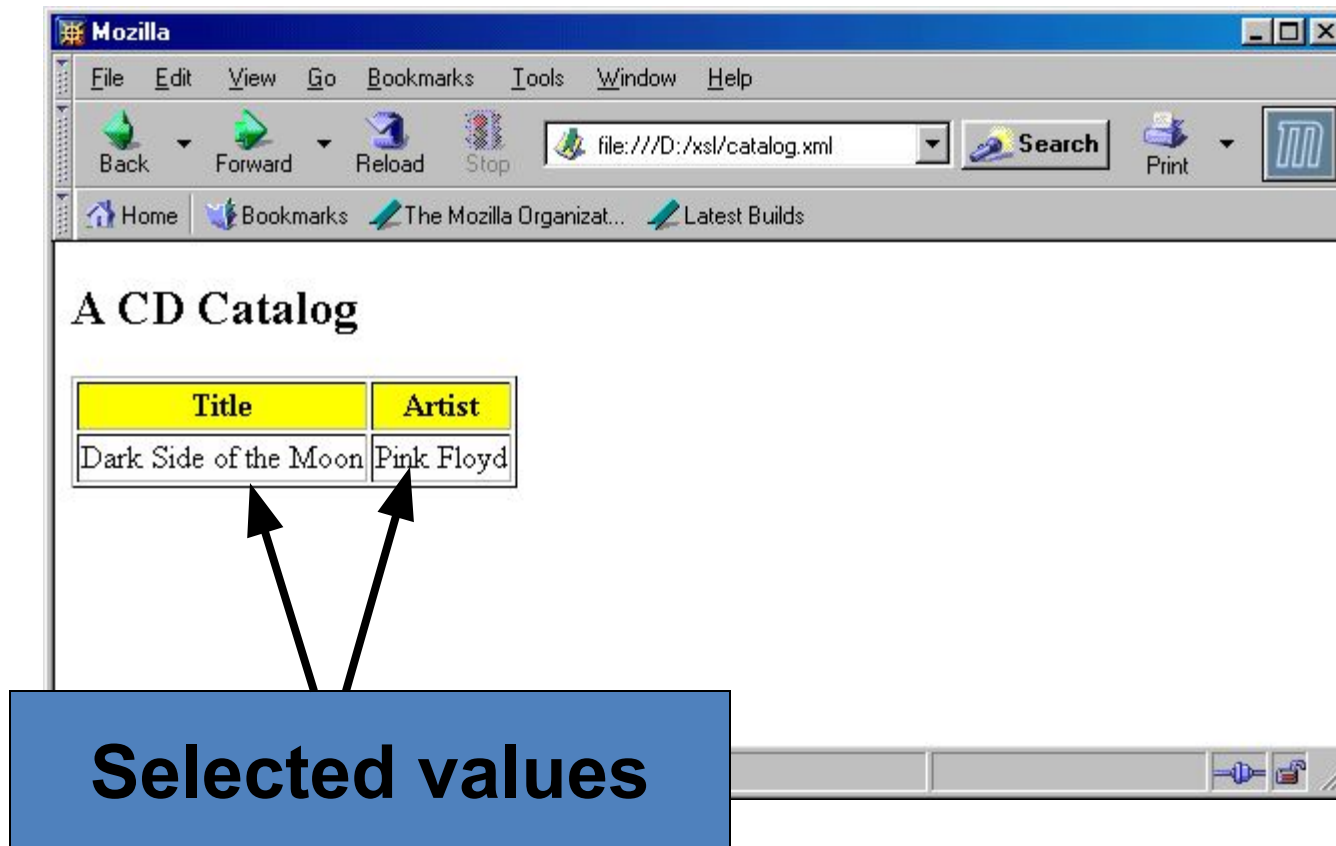
The Most Frequently Used Elements of XSL

- `<xsl:value-of select="xpath-expression"/>`
 - This element extracts the value of a node from the nodelist located by *xpath-expression*
- `<xsl:for-each select="xpath-expression"/>`
 - This element loops over all the nodes in the nodelist located by *xpath-expression*
- `<xsl:if test="xpath-expression"/>`, `<xsl:if test="xpath-expression=value"/>`, etc.
 - This element is for conditional processing

The `<xsl:value-of>` Element

```
<xsl:value-of select="xpath-expression"/>
```

- The XSL element `<xsl:value-of>` can be used to extract the value of an element that is selected from the source XML document
- The extracted value is added to the output stream
- The selected element is located by an XPath expression that appears as the value of the *select* attribute




```
<?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>A CD Catalog</h2>
    <table border="1">
      <tr bgcolor="yellow">
        <th>Title</th>
        <th>Artist</th>
      </tr>
```

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

</xsl:stylesheet>
```

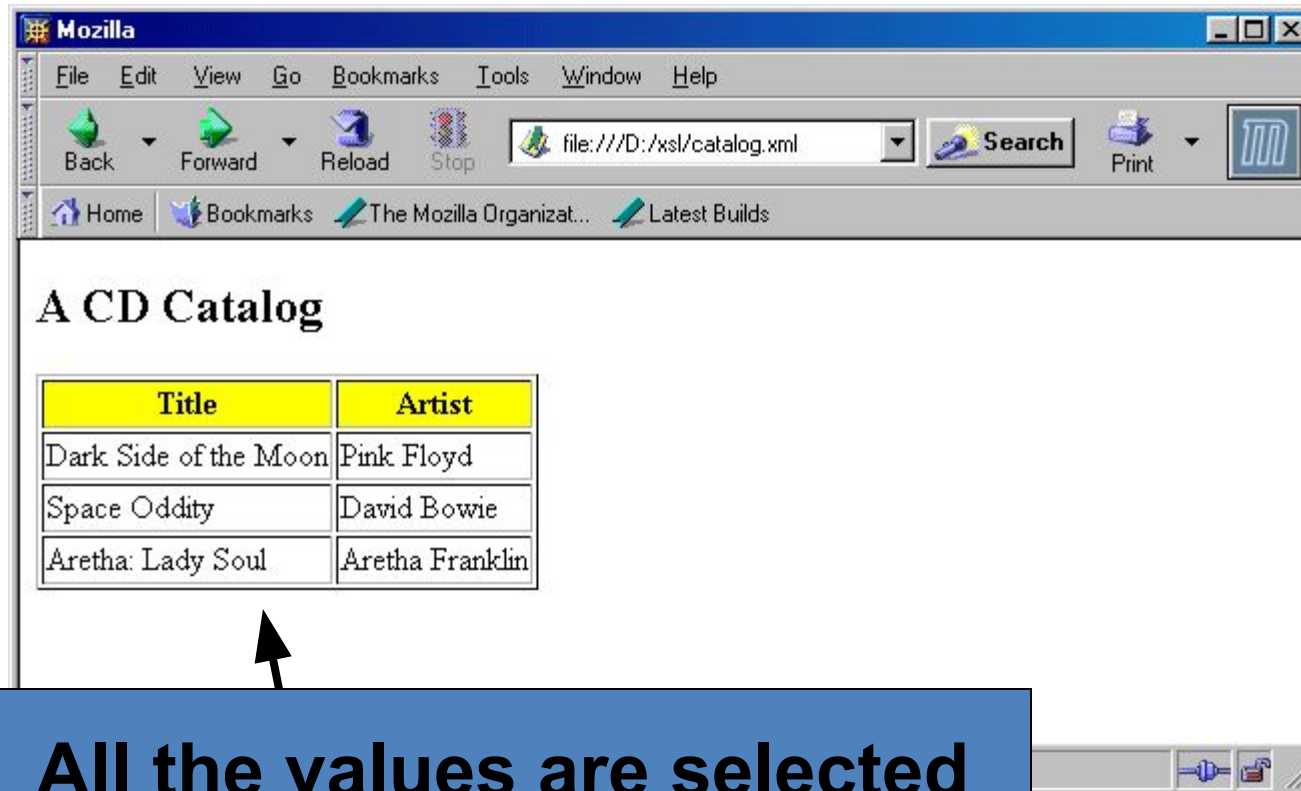


Note that only the first matched element is retrieved for each <xsl:value of>

The **<xsl:for-each>** Element

```
<xsl:for-each select="xpath-expression"/>
```

- The `<xsl:for-each>` element loops over all the nodes in the nodelist of the XPath expression that appears as the value of the *select* attribute
- The value of each node can be extracted by an `<xsl:value-of>` element



```
<?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>A CD Catalog</h2>
    <table border="1">
      <tr bgcolor="yellow">
        <th>Title</th>
        <th>Artist</th>
      </tr>
```

**As in the
previous
example**

```
<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>

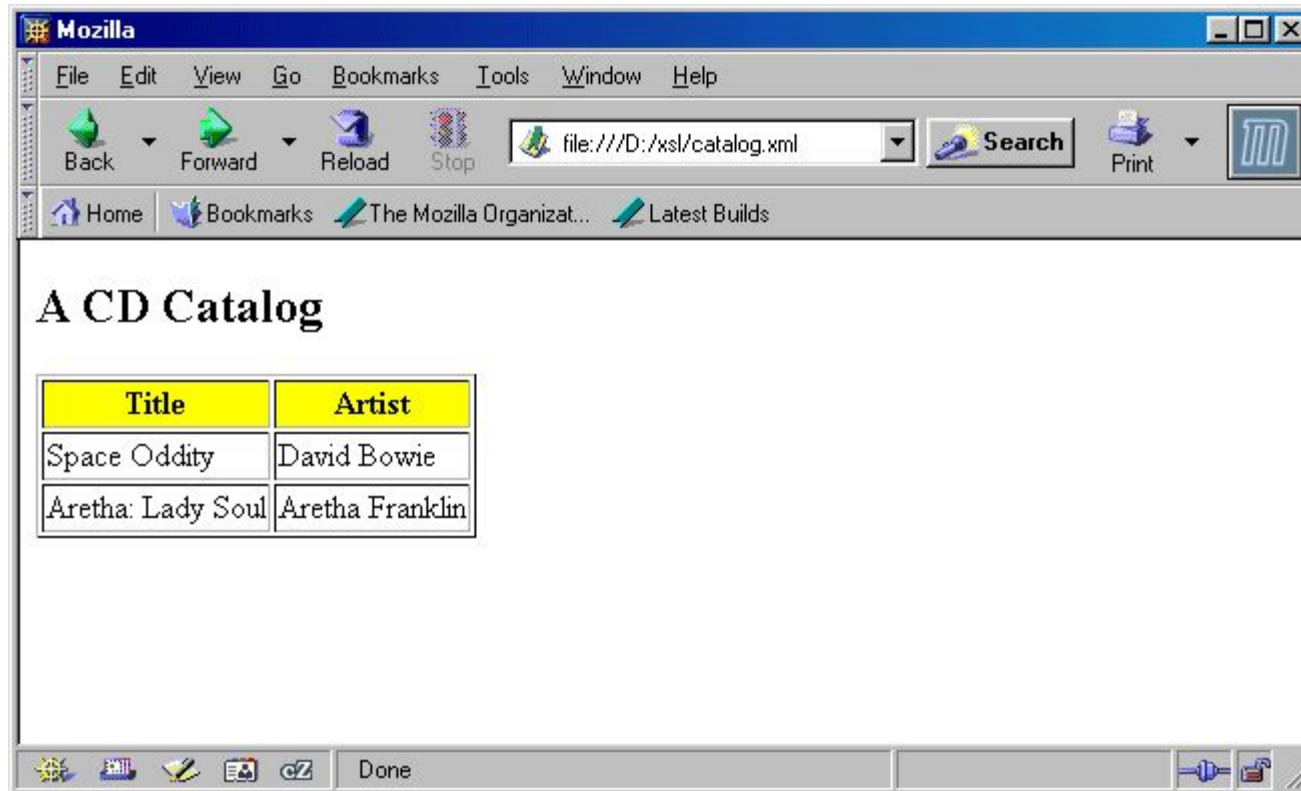
</xsl:stylesheet>
```

Note that all the /catalog/cd elements are retrieved

Consider the following change in the select attribute:

```
<xsl:for-each  
  select="catalog/cd[price<10]">  
  <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>
```

Only elements that satisfy
/catalog/cd[price<10]
are retrieved



The **<xsl:sort>** Element

- The **<xsl:sort>** element is used to sort the list of nodes that are looped over by the **<xsl:for-each>** element
- Thus, the **<xsl:sort>** must appear inside the **<xsl:for-each>** element
- The looping is done in sorted order

Mozilla

File Edit View Go Bookmarks Tools Window Help

Back Forward Reload Stop file:///D:/xsl/catalog.xml Search Print

Home Bookmarks The Mozilla Organizat... Latest Builds

A CD Catalog

Title	Artist
Aretha: Lady Soul	Aretha Franklin
Space Oddity	David Bowie
Dark Side of the Moon	Pink Floyd

Sorted by the name of the artist

```
<xsl:for-each select="/catalog/cd">
  <xsl:sort select="artist"/>
  <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>
```

The /catalog/cd elements
are sorted according to the
value of the artist element

The `<xsl:if>` Element

- The `<xsl:if>` element is used for conditional processing
- The condition appears as the value of the *test* attribute, for example:

```
<xsl:if test="price > 10">  
    some output ...  
</xsl:if>
```

- The elements inside the `<xsl:if>` element are processed if the condition is true

Note

- Processing the inside elements means
 - Copying them into the output stream if they are not XSL elements, and
 - Evaluating them if they are XSL elements
- If the value of the test attribute is just an XPath expression (i.e., without any comparison), then the test is satisfied if the nodelist of this XPath expression is not empty

```
<?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>A CD Catalog</h2>
    <table border="1">
      <tr bgcolor="yellow">
        <th>Title</th>
        <th>Artist</th>
      </tr>
```

**As in the
previous
examples**



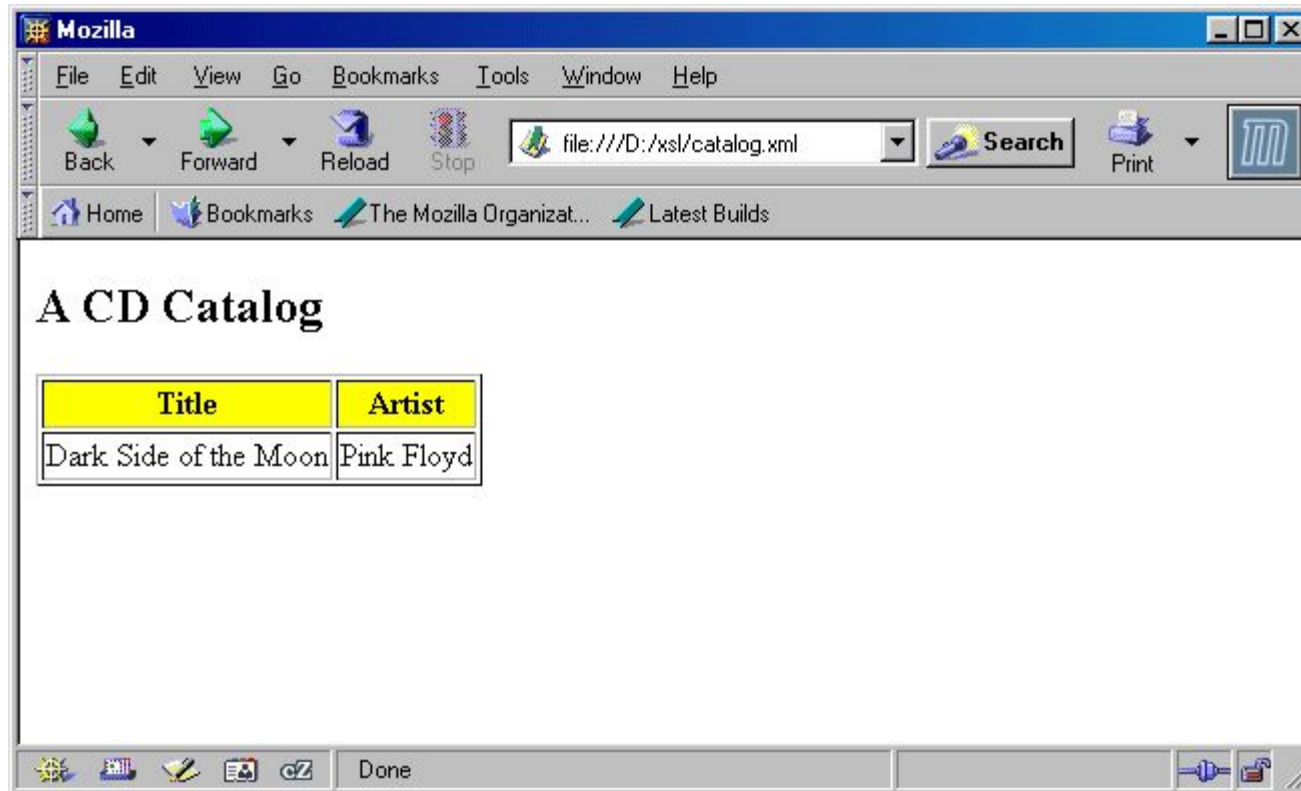
```
<xsl:for-each select="catalog/cd">
  <xsl:if test="price > 10">
    <tr>
      <td><xsl:value-of select="title"/>
    </td>
      <td><xsl:value-of select="artist"/>
    </td>
    </tr>
  </xsl:if>
</xsl:for-each>
```

```
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Only /catalog/cd with
price>10 are retrieved





The **<xsl:choose>** Element

- The **<xsl:choose>** element is used in conjunction with **<xsl:when>** and **<xsl:otherwise>** to express test with multiple conditions
- There can be many **<xsl:when>** inside an **<xsl:choose>** element, but there should be a single **<xsl:otherwise>** inside an **<xsl:choose>** element

Using `<xsl:choose>`

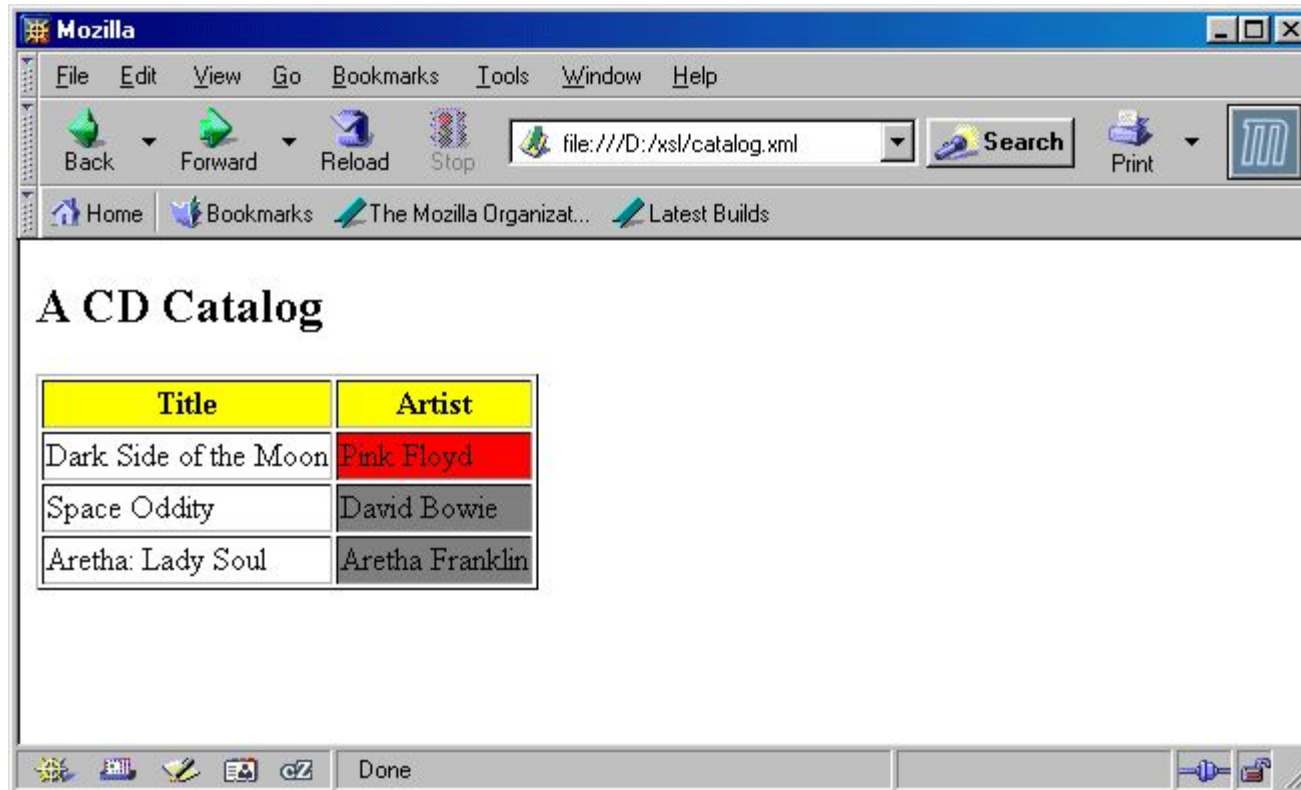
- To insert a conditional choose against the content of the XML file, simply add the `<xsl:choose>`, `<xsl:when>`, and `<xsl:otherwise>` elements to your XSL document like this:

```
<xsl:choose>
  <xsl:when test="price > 10">
    ... some code ...
  </xsl:when>
  <xsl:otherwise>
    ... some code ....
  </xsl:otherwise>
</xsl:choose>
```

```

<xsl:for-each select="catalog/cd"><tr>
<td><xsl:value-of select="title"/></td>
<xsl:choose>
  <xsl:when test="price > 10">
    <td bgcolor="red">
      <xsl:value-of select="artist"/></td>
    </xsl:when>
    <xsl:when
      test="price>9 and price<=10">
      <td bgcolor="gray">
        <xsl:value-of select="artist"/></td>
      </xsl:when>
      <xsl:otherwise>
        <td><xsl:value-of select="artist"/></td>
      </xsl:otherwise>
    </xsl:choose></tr>
</xsl:for-each>

```



Applying Templates Recursively

- The following example shows how to apply templates recursively
- Generally, it is possible (but not in this example) that more than one template matches the current source node
- The specification (www.w3.org/TR/xslt) describes (Section 5.5) which template should be chosen for application

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
```

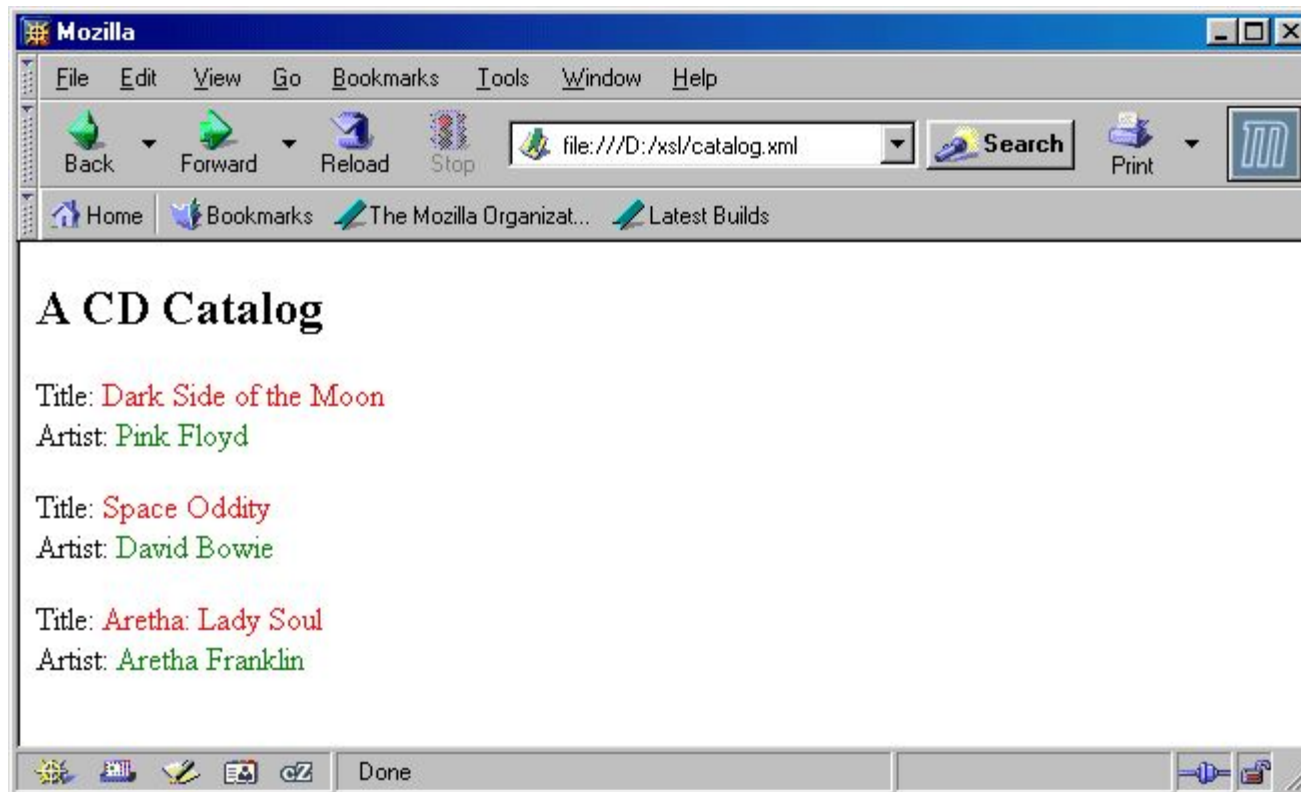
```
<xsl:template match="/">
  <html>
    <body>
      <h2>A CD Catalog</h2>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
```

```
<xsl:template match="cd">
  <p>
    <xsl:apply-templates select="title"/>
    <xsl:apply-templates select="artist"/>
  </p>
</xsl:template>
```

```
<xsl:template match="title">
  Title: <span style="color:red">
    <xsl:value-of select="."/></span>
  <br />
</xsl:template>
```

```
<xsl:template match="artist">
  Artist: <span style="color:green">
    <xsl:value-of select="."/></span>
    <br />
</xsl:template>

</xsl:stylesheet>
```



Is Recursive Application of Templates Really Needed?

- The output of the previous example can also be generated by an XSL stylesheet that uses only one template that matches the root (and does not use the element **<xsl:apply-templates>**)
- However, some tasks can only be done by applying templates recursively
 - This typically happens when the structure of the source XML document is not known

For example

- Suppose that we want to write an XSL stylesheet that generates an exact copy of the source XML document
 - It is rather easy to do it when the structure of the source XML document is known
- Can we write an XSL stylesheet that does it for every possible XML document?
 - Yes! (see next slide)

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

    <xsl:output method="xml"/>

    <xsl:template match="*">
        <xsl:element name="{name(.)}">
            <xsl:for-each select="@*">
                <xsl:attribute name="{name(.)}">
                    <xsl:value-of select="."/>
                </xsl:attribute>
            </xsl:for-each>
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>

</xsl:stylesheet>
```

Identity Transformation Stylesheet

The `<xsl:output>` Element

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
<xsl:output method="xml" version="1.0"
encoding="iso-8859-1" indent="yes"/>
```

.....

```
</xsl:stylesheet>
```

Tells in what format the output should be: xml/html/text

Some Other XSL Elements

- The `<xsl:text>` element allows to insert free text in the output
- The `<xsl:copy-of>` element creates a copy of the current node
- The `<xsl:comment>` element is used to create a comment node in the result tree
- There are more elements and functions: **look in the specification!** (www.w3.org/TR/xslt)

<xsl:text>

```
<?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>
    <p>Titles:
    <xsl:for-each select="catalog/cd">
      <xsl:value-of select="title"/>
      <xsl:if test="position() < last()-1">
        <xsl:text>, </xsl:text>
      </xsl:if>
      <xsl:if test="position()=last()-1">
        <xsl:text>, and </xsl:text>
      </xsl:if>
```

<xsl:text> (cont'd)

```
<xsl:if test="position()=last()">  
  <xsl:text>!</xsl:text>  
</xsl:if>  
</xsl:for-each>  
</p>  
</body>  
</html>  
</xsl:template>  
  
</xsl:stylesheet>
```

<xsl:copy-of>

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="header">
    <tr bgcolor="#9acd32">
      <th align="left">Title</th>
      <th align="left">Artist</th>
    </tr>
  </xsl:variable>
  <xsl:template match="/">
    <html>
      <body>
        <h2>My CD Collection</h2>
        <table border="1">
          <xsl:copy-of select="$header"/>
          <xsl:for-each select="catalog/cd">
```

<xsl:copy-of> (cont'd)

```
<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>
```

W3Schools Tutorial on XSLT

- The [W3Schools XSLT Tutorial](#) has (among other things) tables that list all the elements and functions of XSLT
- It also has some details about implementations
 - Some browsers may not implement all features or may implement some features differently from the specifications

Summary

- XSLT is a high-level transformation language
- Create core output once in XML format (using Servlets, JSP, etc.)
- Use XSLT to transform the core output as needed