XSL (eXtensible Stylesheet Language)

- XSL is a high-level functional language used to transform XML documents into various formats (XML, HTML etc.)
- XSL program consists of a set of TEMPLATE rules.
- Each rule consists of a pattern and a template.
  - pattern (XPath expression) => where clause
  - template => construct clause

- XSL processor starts from the root element and tries to apply a pattern to that node; If it succeeds, it executes the corresponding template.
- The template, when executed, usually instructs the processor to produce some XML result and to apply the templates
- Recursively on the node's children.
- An XSL style sheet is a valid XML document

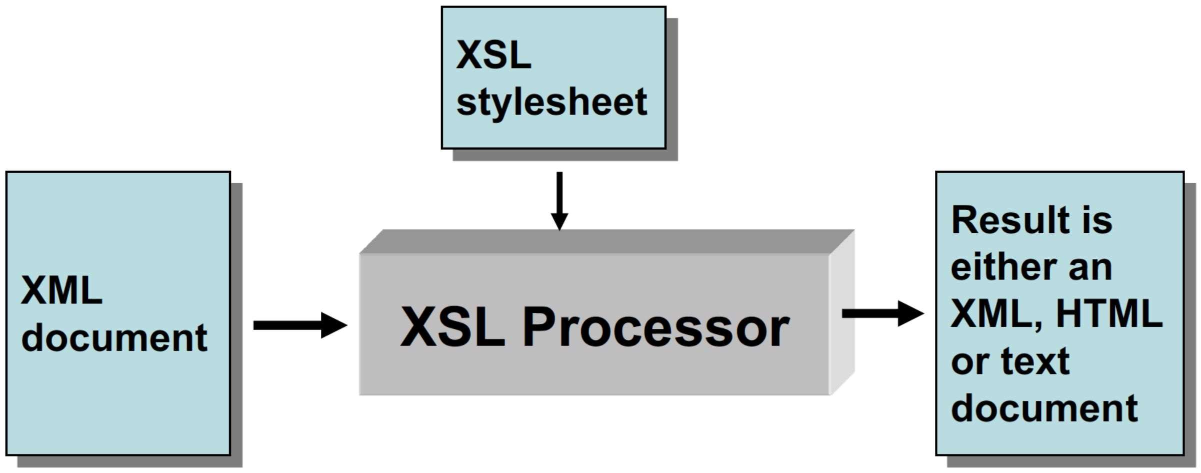
Sample XML Document

`catalog.xml`

Markup

```xml
<?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>
```

Applying XSLT Stylesheets to XML Documents



There are three ways of applying an XSLT stylesheet to an XML document:

1. Directly applying an XSLT processor to the XML document and the XSLT stylesheet; e.g. on command line (`libxml2` tool shown here):

   Bash

   ```bash
   $ xsltproc page1.xsl bib.xml
   ```

2. Calling an XSLT processor from within a (Python or Java) program

```bash
macbook-pro:xsl raj$ more xslTransform.py
import sys
from lxml import etree

def xslTransform(xsl,xml):
  xslt_root = etree.parse(xsl)
  transform = etree.XSLT(xslt_root)
  xml_root = etree.parse(xml)
  result = transform(xml_root)
  return result

def main():
  print(xslTransform(sys.argv[1],sys.argv[2]))

main()
$ python3 xslTransform.py page1.xsl bib.xml
```

3. Adding to the XML document a link to the XSL stylesheet and letting the browser do the transformation

```markup
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="page1.xsl"?>

<Journals>
  <Journal>
  ...
  ...
  </Journal>
</Journals>
```

The Root of the XSL Document (program)

The root element of the XSL document (program) should be one of the following:

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

or

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

The `xsl` 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">
  ... Template content ...
</xsl:template>
```

The content of a template consists of

- XML elements and text (HTML etc) that are copied to the result
- XSL elements that are actually instructions

The pattern syntax is a subset of XPath

Simple Example

Hello World! (`p1.xsl`)

Markup
```
<?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>
```

Test this and subsequent examples on Python's http server:

Bash
```
$ python3 -m http.server
```

XSL Processing

- 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 ("Default Templates")
- The XSL stylesheet must specify explicitly whether templates should be applied to descendants of the root/node
- It is done by putting inside a template the following instruction:

Markup
```
<xsl:apply-templates select="xpath"/>
```

- Without the select attribute, this instruction processes all the children of the current node

Example with `xsl:apply-templates` (`p2.xsl`)

Markup
```
<?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>
```

Default Templates

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

Markup
```
<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)

Interaction of Default and User Defined Templates (`p3.xsl`)

Markup
```
<?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/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="cd[title='Space Oddity']">
    <h1>Hello World</h1>
  </xsl:template>

</xsl:stylesheet>
```

- The default templates print the text in the leaves of the first and third CD
- The second template above prints the "Hello World" (this replaces the default template for this node!)

The Most Frequently Used Elements of XSL

- The `value-of` element extracts the value of a node from the nodelist located by xpath-expression:

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

- The `for-each` element loops over all the nodes in the nodelist located by xpath-expression

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

- The `if` element is for conditional processing

Markup
```
<xsl:if test="xpath-expression"/>
<xsl:if test="xpath-expression=value"/>
```

The `<xsl:value-of>` Element

Markup

```
<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

Example for `value-of` (`p4.xsl`)

Markup

```
<?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

Markup
```
<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

Example for `for-each` (`p5.xsl`)

Markup
```
<?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>
        <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 if we change

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

to

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

we will get only CDs which have a price less than 10.

Example for `for-each` (`p6.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>A CD Catalog</h2>
        <table border="1">
        <tr bgcolor="yellow">
        <th>Title</th>
        <th>Artist</th>
        </tr>
        <xsl:for-each select="catalog/cd[price &lt; 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>
```

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

Example for `sort` (`p7.xsl`)

Markup

```
<?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>
        <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 `<xsl:if>` Element

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

  Markup

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

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

Example for `if` (`p8.xsl`)

Markup

```
<?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>
        <xsl:for-each select="catalog/cd">
          <xsl:if test="price &gt; 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>
```

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

Markup

```
<?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

Recursive Template Application

- 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!

Markup

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

  <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>
```

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