DH/DS at Pitt-Greensburg: # An XSLT Stylesheet to Transform XML to HTML

```
?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:math="http://www.w3.org/2005/xpath-functions/math" exclude-result-prefixes="xs math"
  xmlns="http://www.w3.org/1999/xhtml" version="3.0">
```

<!--ebb: The <xsl:stylesheet> is our root element and indicates the relevant namespaces that we need to declare to a) match the XML we're working with, and b) output in the appropriate way for XHTML. The URLs are the official namespaces we need. -->

```
  <xsl:output method="xhtml" indent="yes"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"/>
```

<!--ebb: Next we have the <xsl:output> statement which indicates the kind of output we're making. We added an indent statement here to indent our elements in the output, and this statement will include the required doctype declaration we need for XHTML Strict syntax.-->

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

<!--ebb: Everything inside this template rule starts from the root node and happens just ONCE. It triggers other template rules that match on multiples nested inside. The basic superstructure of our HTML file is written to MATCH the root node of the XML: This template rule says, where you see the root node in the XML, write the following code to transform it to HTML.-->

```
    <html>
      <head><title>Shakespeare's Sonnets</title></head>

      <body>
        <h1>Sonnets</h1>

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

<!--ebb: This says, go look for a template to fire based on the particular XPath destination I want to be matched here. -->

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

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

<!--ebb: This is is the template that will fire next in the sequence of events we're setting up. NOTE: We don't write an XPath expression for @match: templates simply look for nodes of a particular name to match. This says, match me up to any <sonnet> nodes in the document, wherever they are. It's basically going to fire whenever a <sonnet> element turns up: the template rule is there waiting for the eventuality of there being a <sonnet> element anywhere in the document. -->

<!--ebb: OK, let's say we want to remix our XML so we actually output a Sonnet header with the word Sonnet followed by its number, which we need to pull out of those <sonnet number="Roman_Numeral"> elements. Here's what we do. I'm going to use the html <h2> element:-->

<h2>Sonnet <xsl:apply-templates select="@number"/> </h2>
<xsl:apply-templates select="line"/>

<!--ebb: In the <xsl:apply-templates/> rules, notice THREE THINGS:
1) These are usually self-closing elements. (Sometimes, if we want to sort results or process them somehow, we open and close the <xsl:apply-templates> and put a rule inside like this: <xsl:apply-templates select="line">
<xsl:-sort/>
</xsl:apply-templates/>
You'll learn more about sorting later. For now, we're just familiarizing you with the syntax!)
2) In the @select attribute, we expect to see an XPath expression. Here you might be wondering where the XPath is, but it's there. We don't have to specify a path step here because the default expectation is that we are looking at immediate children of the <sonnet> element in this template rule. And in the <h2> element, we just stepped over onto the attribute axis to pick up the @number on sonnet. The path steps were simple. If we were processing descendants, say the grandchildren of <sonnet> called <note>, we would need to use **@select=".//note"**, using **the "dot"** to indicate we want to start from **this** particular instance, and specifying the descendent axis to look down the tree from here with // .
3) We can sometimes just write <xsl:apply-templates/> with no @select attribute at all. We use the @select attribute when we want something to be positioned precisely: Here, we know we want to go and match the <line> element, and we want the <line> element to be transformed right after our new headings that give a title to each sonnet. -->

</xsl:template>

<xsl:template match="line">
<!--ebb: Again, these template matches typically are not XPath expressions: they're just hunting for nodes of a particular name all through the XML file. We told the parser to go apply templates on the child <line> elements of <sonnet>, and those will be processed with this template rule (and any others we write that apply to it.)-->

** <xsl:value-of select="@n"/><xsl:text> </xsl:text><xsl:apply-templates/> **

<!--ebb: Here, we're going to output line numbers and lines, and we've chosen to format the line numbers with help from a element, because that way we can go style those elements in CSS and make them, say, tiny red numbers with lots of padding to set off from the text.
Notice that we used <xsl:value-of> to output the value of @n. We could have used <xsl:apply-templates select="@n">, and the output is really the same, but we wanted to show you this, because it's often used for numerical values, or to give the results of an XPath function that performed a mathematical calculation. Sometimes <xsl:value-of> and <xsl:apply-templates> are interchangeable, as they are here.
Notice that we need TWO statements here to output first, the line number (because we need to go get the attribute value on <line>), and because we then need to simply output the line of text itself. Try commenting out that last <xsl:apply-templates/> and you'll see why we need it: it outputs the text() (or text node) of the element <line>, and that's really all we had left to match from the XML document. Since we aren't writing any more template rules for this transformation, what happens here with this last <xsl:apply-templates/> is that XSLT fires a "built in" rule to process the text nodes and anything left over in the document.-->
</xsl:template>
</xsl:stylesheet>