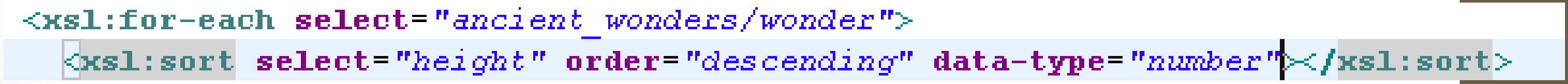
XSLT

Sorting:

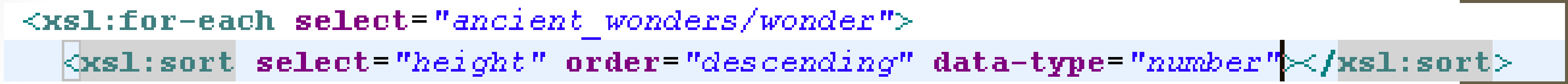
By default, nodes are processed in the order in which they appear in the XML source document.
If you wish to process nodes in some other order, you can add an xsl:sort element when you use xsl:for-each.

Directly after a xsl:for-each element you add :



```
<xsl:for-each select="ancient_wonders/wonder">
   <xsl:sort select="height" order="descending" data-type="number"></xsl:sort>
```

XSLT



```
<xsl:for-each select="ancient_wonders/wonder">
   <xsl:sort select="height" order="descending" data-type="number"></xsl:sort>
```

The select criteria can be an expression that specifies the node key on which the source nodes should be sorted. In this case we are using the node height to sort the wonders.
The sort can be descending or ascending. In this case we are sorting them in a descending order, so the highest wonder first.
If desired, you can specify the data type, the default is text.
Be sure to specify the correct data type, sorting numbers as text gives errors and vice versa.
You can nest xsl:sort elements within other xsl:sort elements.

XSLT

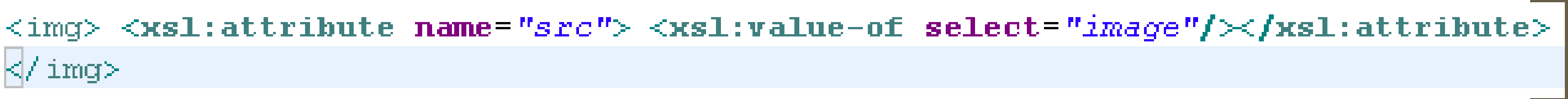
Generating output attributes:

When you are transforming your XML source document to an HTML or XML document, it is often useful to be able to add attributes and values to a given output element. For example, if you are creating an <img> element or an <a> element, you might need to include the src, width and height attributes to the image element, or the href and target attributes to an anchor element.

XSLT

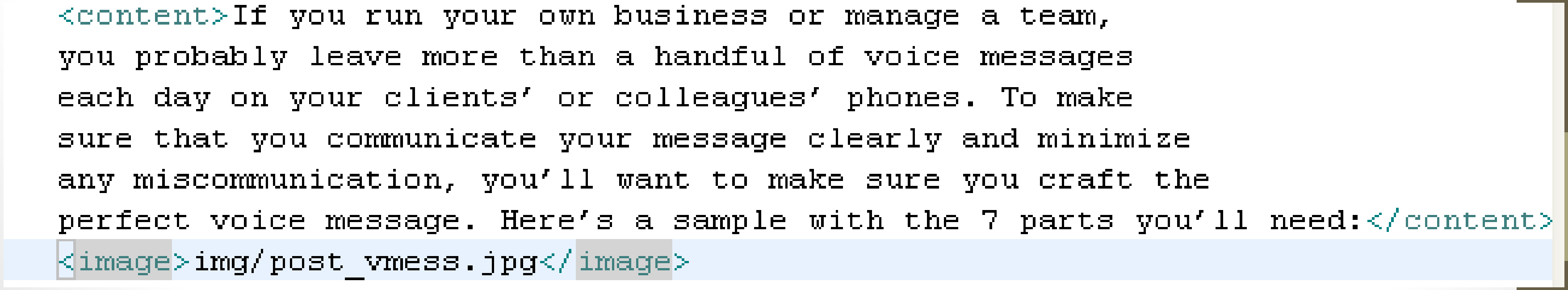
Adding the src attribute to an image element:

XSL file:



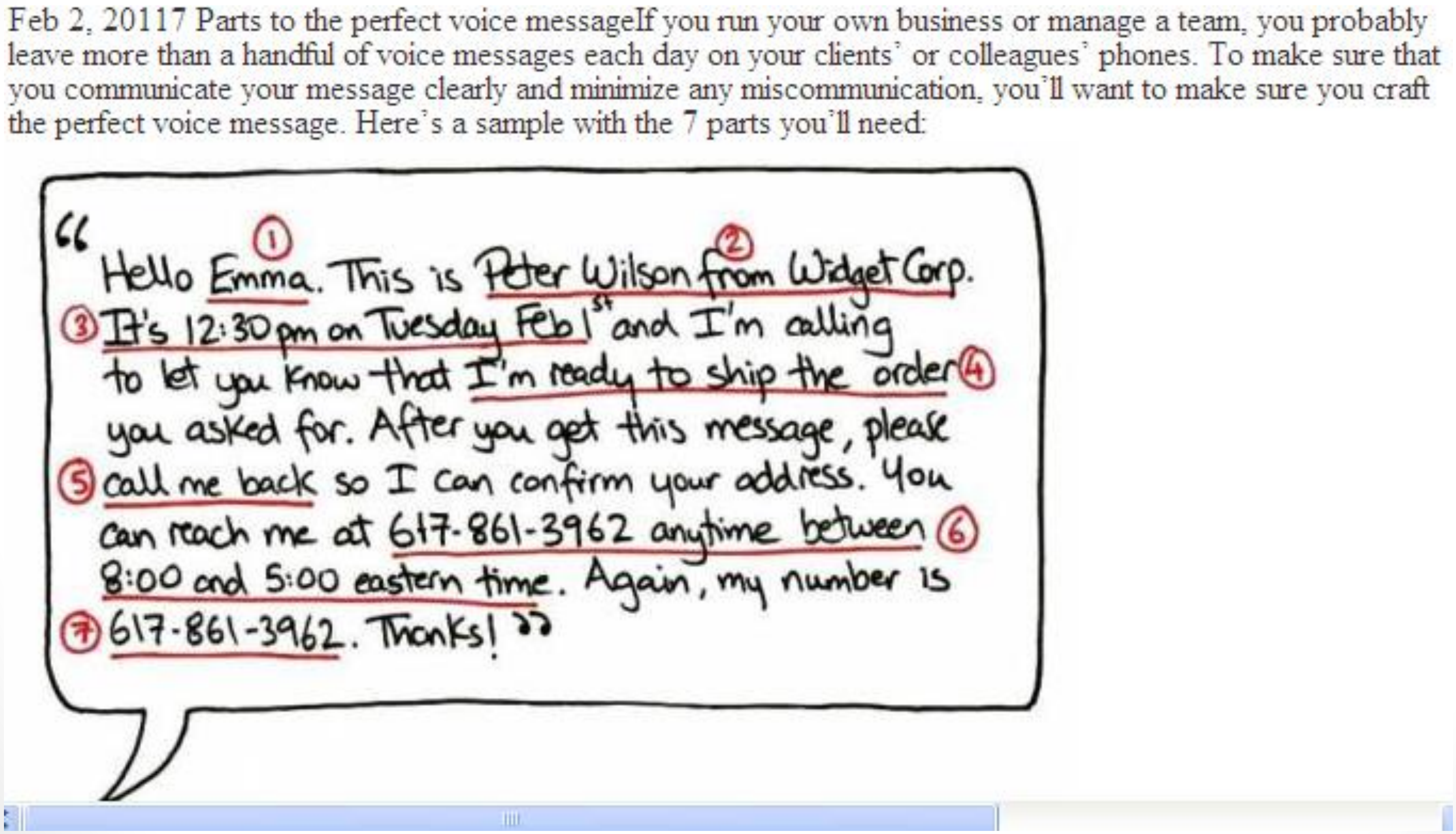
```
<img> <xsl:attribute name="src"> <xsl:value-of select="image"/></xsl:attribute>
</img>
```

XML file:



```
<content>If you run your own business or manage a team,
you probably leave more than a handful of voice messages
each day on your clients' or colleagues' phones. To make
sure that you communicate your message clearly and minimize
any miscommunication, you'll want to make sure you craft the
perfect voice message. Here's a sample with the 7 parts you'll need:</content>
<image>img/post_vmess.jpg</image>
```

XSLT



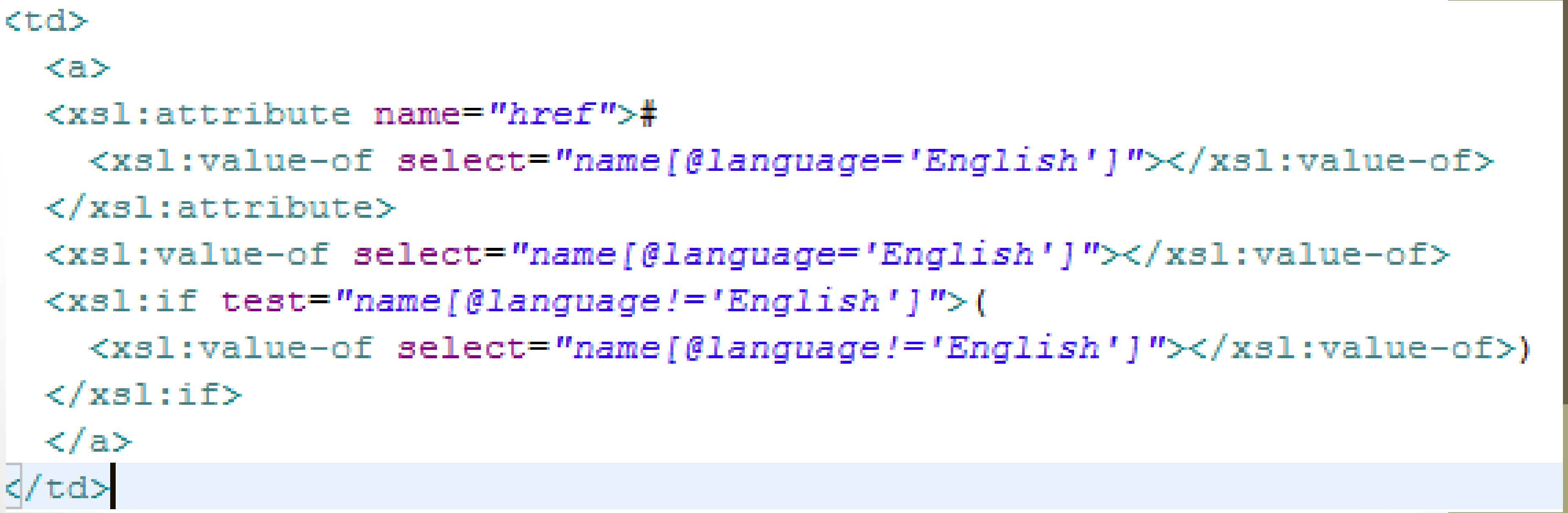
Feb 2, 20117 Parts to the perfect voice messageIf you run your own business or manage a team, you probably leave more than a handful of voice messages each day on your clients' or colleagues' phones. To make sure that you communicate your message clearly and minimize any miscommunication, you'll want to make sure you craft the perfect voice message. Here's a sample with the 7 parts you'll need:

"Hello Emma. (1) This is Peter Wilson from Widget Corp. (2) (3) It's 12:30pm on Tuesday Feb 1st and I'm calling to let you know that I'm ready to ship the order (4) you asked for. After you get this message, please (5) call me back so I can confirm your address. You can reach me at 617-861-3962 anytime between (6) 8:00 and 5:00 eastern time. Again, my number is (7) 617-861-3962. Thanks!"

XSLT

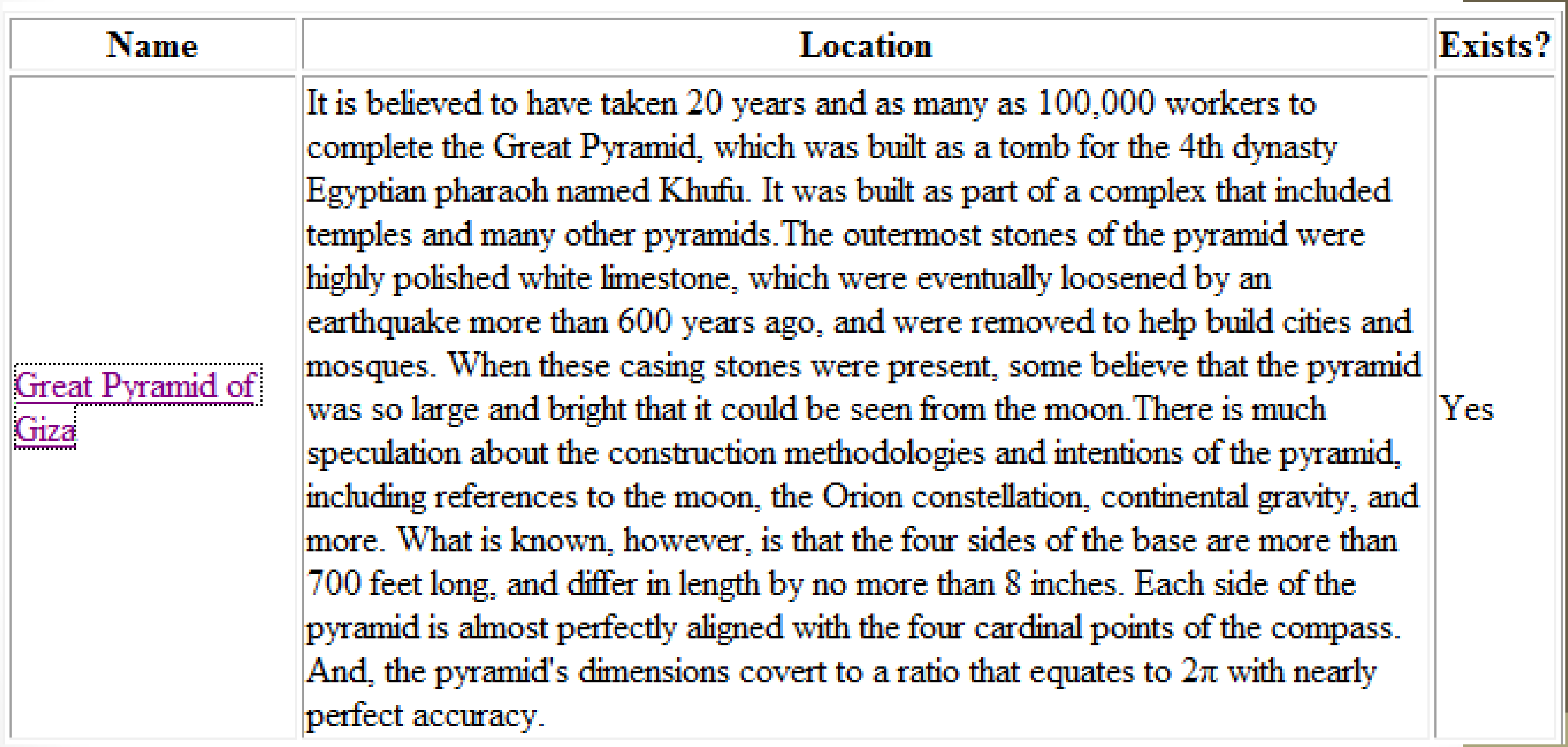
Adding an href attribute to an anchor <a> element. The href value will be set to # and the name of the wonder being processed. In this way each wonder will be linked to its own named reference on the page.

XSL file:



```
<td>
  <a>
  <xsl:attribute name="href">#
    <xsl:value-of select="name[@language='English']"></xsl:value-of>
  </xsl:attribute>
  <xsl:value-of select="name[@language='English']"></xsl:value-of>
  <xsl:if test="name[@language!='English']">(
    <xsl:value-of select="name[@language!='English']"></xsl:value-of>)
  </xsl:if>
  </a>
</td>
```

XSLT



Name	Location	Exists?
Great Pyramid of Giza	It is believed to have taken 20 years and as many as 100,000 workers to complete the Great Pyramid, which was built as a tomb for the 4th dynasty Egyptian pharaoh named Khufu. It was built as part of a complex that included temples and many other pyramids.The outermost stones of the pyramid were highly polished white limestone, which were eventually loosened by an earthquake more than 600 years ago, and were removed to help build cities and mosques. When these casing stones were present, some believe that the pyramid was so large and bright that it could be seen from the moon.There is much speculation about the construction methodologies and intentions of the pyramid, including references to the moon, the Orion constellation, continental gravity, and more. What is known, however, is that the four sides of the base are more than 700 feet long, and differ in length by no more than 8 inches. Each side of the pyramid is almost perfectly aligned with the four cardinal points of the compass. And, the pyramid's dimensions covert to a ratio that equates to 2π with nearly perfect accuracy.	Yes

XSLT

Templates:

The root template is the first thing processed in an XSLT style sheet. This template is the set of rules applied to the root node of the XML source document.
XSLT allows you to create more templates than just the root template. This allows you to create different sets of processing rules to apply to different parts of your XML.
One of the main benefits of using templates is the ability to reuse a template for other nodes in your document. In the same way that one can use functions in most programming languages, you would create a template, and simply apply that template whenever necessary. This eliminates the need to rewrite the exact same processing instructions each time.

XSLT

To create a template:

Type

```
<xsl:template match=“name[@language!=‘English’]”>
(<em>
  <xsl:value-of select=“.”>
</em>)
</xsl:template>
```

This new template will italicize the output for the name of each wonder that is not English. It does so by using xsl:value-of select=“.” which returns the value of the current node.

XSLT

The root template is simply a template with a pattern that matches the root node.

Only the root template is called automatically. All other templates must be applied manually. Otherwise they are ignored.

So how do you apply a template to a specific node in your XML document?

You must use xsl:apply-templates.
This is how you control where and when the transformation described by the template is used in the final document.

XSLT

Applying a template:

Within any template, you can include the following:

<xsl:apply-templates select=“expression”/>

Expression identifies the node(s) of the XML document whose templates should be applied. The template created previously is now used:

XSLT

Our code originally had the following:

```
<xsl:if test=“name[@language!=‘English’]”>
(<em><xsl:value-of select=“name[@langugage!=‘English’]”/></em>)
</xsl:if>
```

We can replace it with the following:

```
<xsl:apply-templates select=“name[@language!=‘English’”/>
```

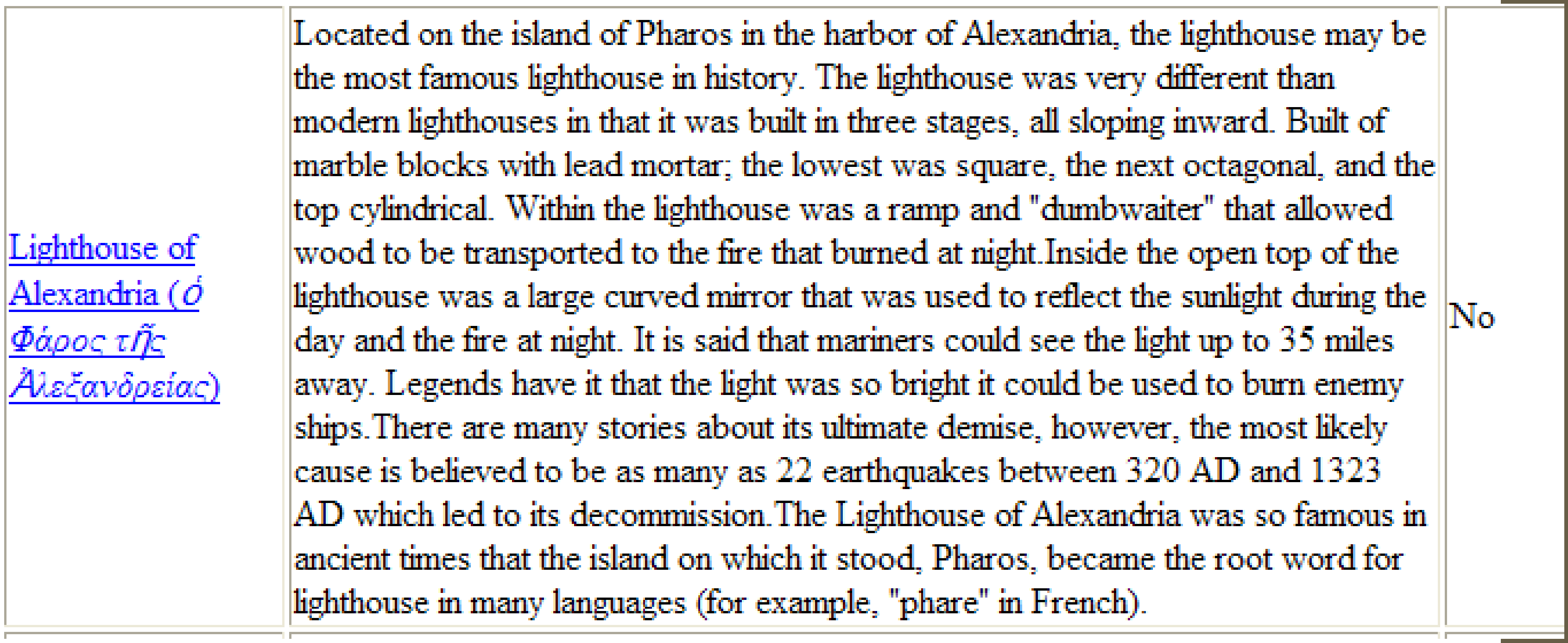
The template is applied which outputs in italics the current node that meets the criteria i.e. A name that is not in English.

XSLT

The template applies the style to the current node.

```
<xsl:template match=“name[@language!=‘English’]”>
(<em>
  <xsl:value-of select=“.”>
</em>)
</xsl:template>
```

XSLT



Lighthouse of Alexandria (ὁ Φάρος τῆς Ἀλεξανδρείας)	Located on the island of Pharos in the harbor of Alexandria, the lighthouse may be the most famous lighthouse in history. The lighthouse was very different than modern lighthouses in that it was built in three stages, all sloping inward. Built of marble blocks with lead mortar; the lowest was square, the next octagonal, and the top cylindrical. Within the lighthouse was a ramp and "dumbwaiter" that allowed wood to be transported to the fire that burned at night.Inside the open top of the lighthouse was a large curved mirror that was used to reflect the sunlight during the day and the fire at night. It is said that mariners could see the light up to 35 miles away. Legends have it that the light was so bright it could be used to burn enemy ships.There are many stories about its ultimate demise, however, the most likely cause is believed to be as many as 22 earthquakes between 320 AD and 1323 AD which led to its decommission.The Lighthouse of Alexandria was so famous in ancient times that the island on which it stood, Pharos, became the root word for lighthouse in many languages (for example, "phare" in French).	No

XSLT

Tips:

If you have multiple templates in your XSL file, the order of the xsl:apply-templates determines the order in which the templates are processed.

If you don't specify the select attribute the processor will look for and apply a template to each of the current nodes children.

When using xsl:apply-templates, if there is no matching template a template built into the XSLT processor is automatically used.

The xsl:apply-templates element may contain an xsl:sort element.

XSLT

To output the story for each wonder in a particular format we can use a template.

The template is to output the story in a paragraph, with font face=“Times New Roman” size=“5” and color=“#000080”

How do we do this?

XSLT

```
<xsl:for-each select="ancient_wonders/wonder">
    <xsl:sort select="height" order="descending" data-type="number" ></xsl:sort>
    <tr>
    <td>
      <a>
      <xsl:attribute name="href">#
        <xsl:value-of select="name[@language='English']"></xsl:value-of>
      </xsl:attribute>
      <xsl:value-of select="name[@language='English']"></xsl:value-of>
      <xsl:apply-templates select="name[@language!='English']">
              </xsl:apply-templates>
      </a>
    </td>
     <td><xsl:value-of select="history/story"></xsl:value-of></td>
     <td><xsl:choose>
      <xsl:when test="history/year_destroyed!=0">No</xsl:when>
      <xsl:otherwise>Yes</xsl:otherwise>
    </xsl:choose></td>
    </tr>
  </xsl:for-each>
```

XSLT

Within the for-each loop of the ancient_wonders/wonder node we want to output the story in a particular format, to do this we create a new template for the history/story node.

XSLT

```
<xsl:template match="history/story">
  <p>
   <font color="#000080" face="Times New Roman"
            size="5">
    <xsl:value-of select="."></xsl:value-of>
    </font>
  </p>
</xsl:template>
```

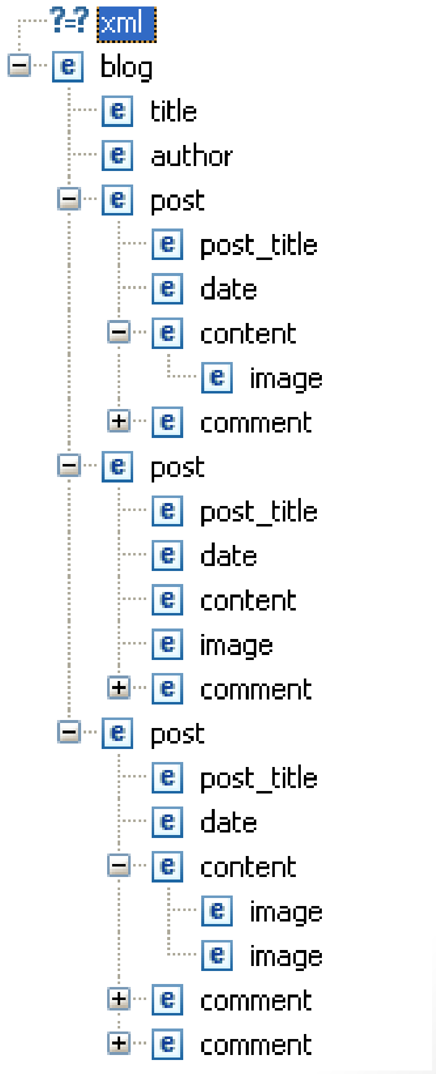
We then use that template by applying it within the for-each loop.

XSLT

```
<xsl:for-each select="ancient_wonders/wonder">
     <xsl:sort select="height" order="descending" data-type="number“></xsl:sort>
     <tr> <td> <a>
       <xsl:attribute name="href">#
         <xsl:value-of select="name[@language='English']"></xsl:value-of>
       </xsl:attribute>
       <xsl:value-of select="name[@language='English']"></xsl:value-of>
       <xsl:apply-templates select="name[@language!='English']">
                 </xsl:apply-templates>
       </a>  </td>
   <td>   <xsl:apply-templates select=“history/story”> </xsl:apply-templates>
</td>
      <td><xsl:choose>
       <xsl:when test="history/year_destroyed!=0">No</xsl:when>
       <xsl:otherwise>Yes</xsl:otherwise>
     </xsl:choose></td>
     </tr>
    </xsl:for-each>
```

XSLT

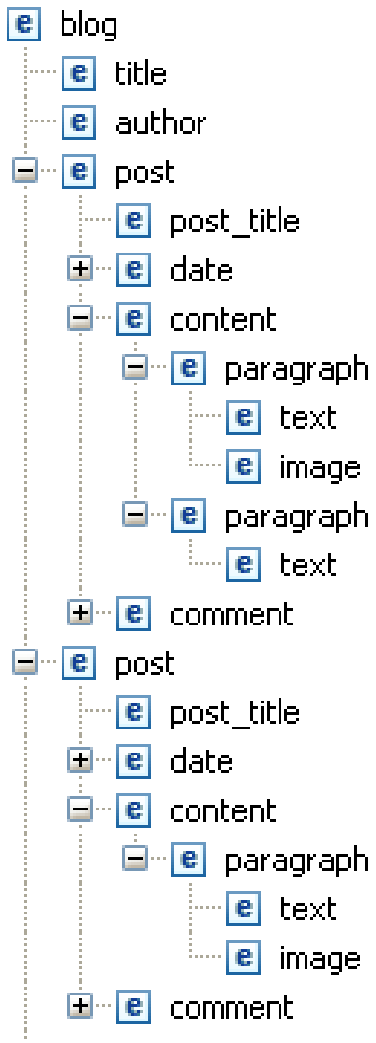
Data elements:
It is very important that the structure of your data is well designed. Where we have data content as well as nested elements it can make it difficult to access content easily using xsl. For the blog you could store the data elements as shown here. However, this would mean that it was difficult to access the both the text in a content element and the image or images in a content element particularly if it varied as to the order of text and images within content. It also isn't clear which text is



XSLT

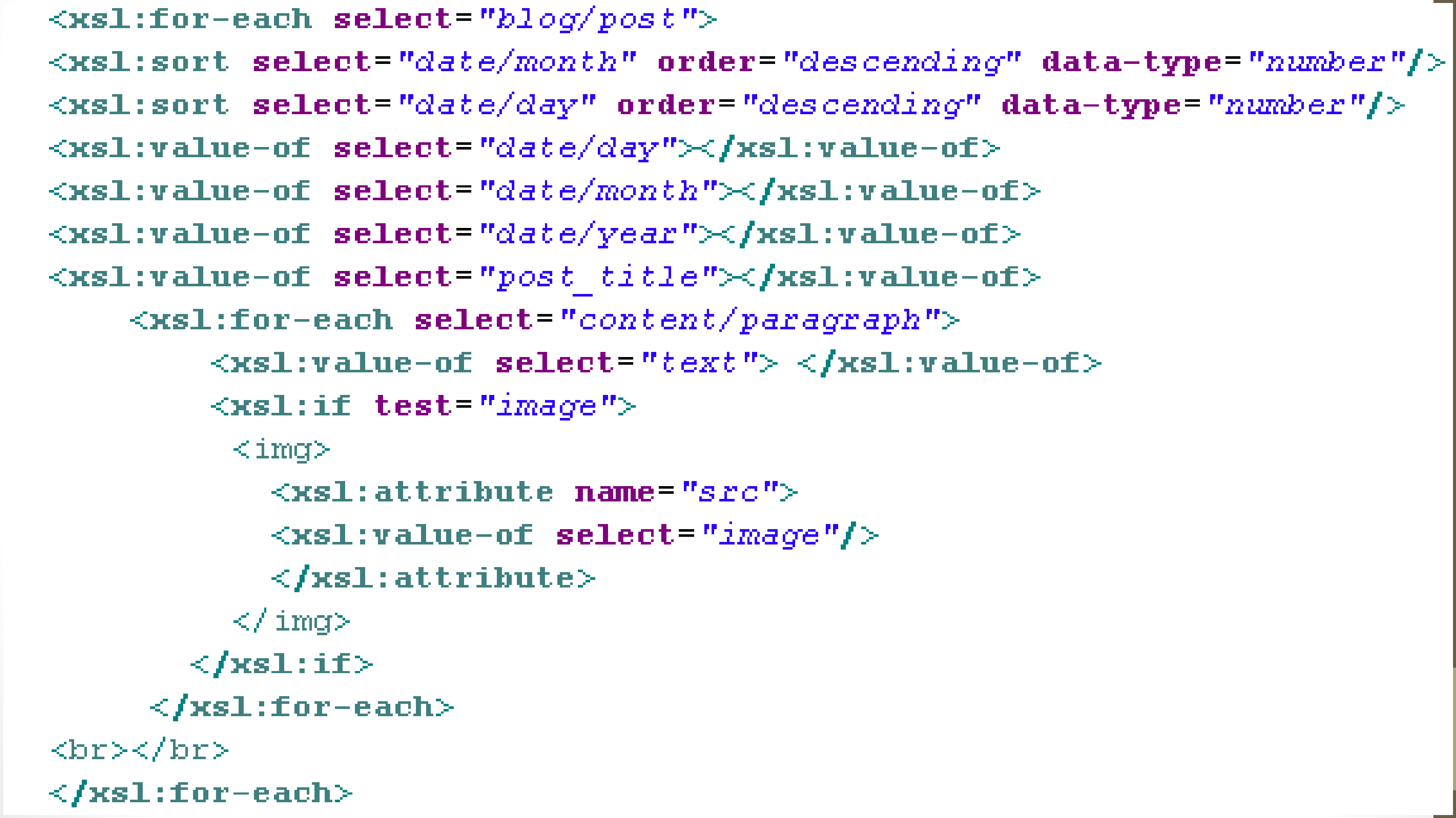
Data elements:

It is more appropriate to store the data in a slightly different format.
This means that a post can have many paragraphs, each with text and in some cases an image. Now we can access and display each paragraphs text and image without difficulty.



```
blog
  title
  author
  post
    post_title
    date
    content
      paragraph
        text
        image
      paragraph
        text
    comment
  post
    post_title
    date
    content
      paragraph
        text
        image
    comment
```

XSLT



```
<xsl:for-each select="blog/post">
<xsl:sort select="date/month" order="descending" data-type="number"/>
<xsl:sort select="date/day" order="descending" data-type="number"/>
<xsl:value-of select="date/day"></xsl:value-of>
<xsl:value-of select="date/month"></xsl:value-of>
<xsl:value-of select="date/year"></xsl:value-of>
<xsl:value-of select="post_title"></xsl:value-of>
    <xsl:for-each select="content/paragraph">
        <xsl:value-of select="text"> </xsl:value-of>
        <xsl:if test="image">
         <img>
           <xsl:attribute name="src">
           <xsl:value-of select="image"/>
           </xsl:attribute>
         </img>
       </xsl:if>
     </xsl:for-each>
<br></br>
</xsl:for-each>
```