Apuntes básicos del Tema 9

Introducción a XSLT

Contenidos:

9.1- Transformaciones XSL: el lenguaje XSLT

Siguiendo con las distintas tecnologías basadas en XML que hemos analizado en los temas anteriores, llegamos ahora a **XSL** (Extensible Stylesheet Language) que podemos traducir como *lenguaje extensible de hojas de estilo*, pero que en realidad es un metalenguaje, por estar formado por una familia de especificaciones o recomendaciones oficiales del W3C y que son:

- XSLT, que es el verdadero lenguaje de transformación (y que será el objeto principal de este tema)
- XPATH, ya tratado en un tema anterior y cuyo cometido es hacer búsquedas y seleccionar partes del documento XML
- XSL-FO, que se trata de un lenguaje de formateo de objetos para crear una presentación de texto, y usado principalmente para convertir documentos a PDF. Esta parte tiene una cierta complejidad y no la veremos ya que excede del ámbito del presente curso.

9.2 XSLT frente a CSS

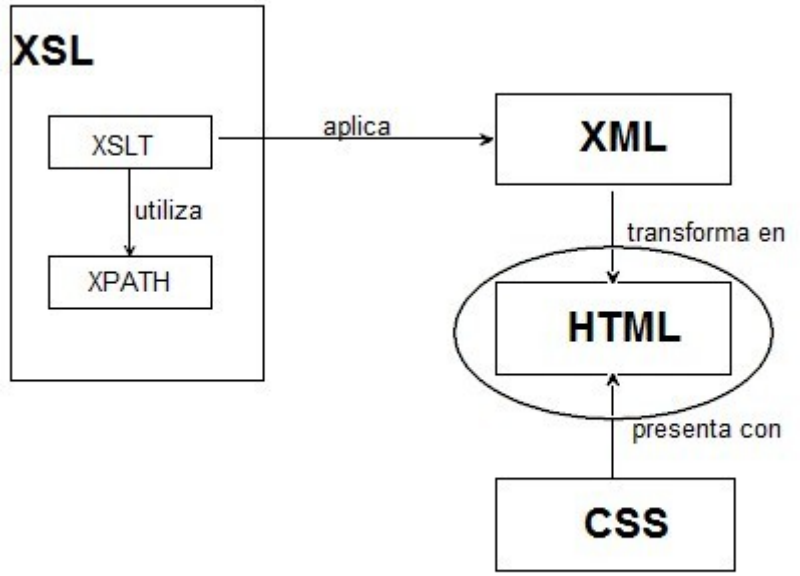
Algunos autores opinan que XSL es en XML el equivalente del CSS en HTML. Por eso se le llama hoja de estilo de XML pero no podemos confundirlo,aunque tengan parte del nombre en común, y función similar basada en la presentación de datos XML en un formato legible.

Desde el principio del uso de XML hubo una gran preocupación por controlar la presentación de los documentos en la WEB y por esa razón se adaptó las tecnologías existentes como las CSS, y que nosotros ya hemos tratado también en temas anteriores. Pero aunque hemos podido comprobar su buen funcionamiento, CSS tiene unas limitaciones marcadas por la falta de construcciones de sentencias de control y filtros adecuados que lo hacen insuficiente cuando entramos de llenos en el mundo de la transformación de documentos.

Es una realidad que XSL permite definir hojas de estilo más adecuadas para los documentos XML frente a CSS que utiliza un método de presentación adaptado al mundo de HTML.

Pero eso no quiere decir que XSL sustituya a las CSS, ya que en la práctica XSL, HTML y CSS son complementarios, de forma que el proceso habitual consiste en que los datos de los documentos XML se utilizan en una página Web mediante su transformación a HTML con XSLT y la ayuda de XPATH, para finalmente presentar el documento HTML utilizando una CSS, como

se representa en el siguiente gráfico:



9.3 XSLT

XSLT es la especificación concreta que dentro del metalenguaje XSL desarrolla el lenguaje de transformación. No obstante, y aclaradas las diferencias, entenderemos que ambos términos (XSL y XSLT) se usan en la práctica como equivalentes para referirse al propio proceso de transformación, siendo simplemente una distinción conceptual a tener en cuenta.

Para hacer las transformaciones se hace uso de la especificación XPath, que aunque en origen fue diseñada para ser utilizada de forma independiente (como hemos comprobado en el tema en que lo hemos tratado) tiene su completa utilidad embebido en una aplicación de transformación creando los filtros adecuados para obtener los elementos que queremos traspasar a los distintos documentos.

Las transformaciones XSL se utilizan principalmente para extraer información de documentos XML y generar archivos XML o HTML.

9.4 Un ejemplo sencillo

Si partimos como ejemplo de un sencillo documento XML como el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<web_cine>
    <nombre>Información de cine</nombre>
    <url>http://www.filmaffinity.com</url>
</web_cine>
```

Para poder transformarlo a HTML,simplemente tenemos que insertar entre el prólogo y el elemento raíz la instrucción de procesamiento *<?xml-stylesheet href=".....>* (similar a la que ya utilizábamos en temas anteriores para relacionar la CSS, pero en este caso el atributo *type* contendrá *text/**xsl***) que nos incorpora el acceso al fichero xsl . Guardamos este fichero como *ejemplo1**xml***

```
<?xml version="1.0" encoding="UTF-8""?>
<?xml-stylesheet href="ejemplo1.xsl" type="text/xsl"?>
<web_cine>
    <nombre>Información de cine</nombre>
    <url>http://www.filmaffinity.com</url>
</web_cine>
```

ejemplo1.xml

y por otro lado confeccionaremos un fichero XSL, que guardaremos como *ejemplo1**xsl*** y que contendrá el código de las transformaciones necesarias:

```
<?xml version="1.0" encoding="UTF-8""?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match='/'>
    <html>
    <head>
        <title>Ejemplo</title></head>
     <body>
        <h1> <xsl:apply-templates /> </h1>
    </body>
    </html>
  </xsl:template>
</xsl:stylesheet
```

ejemplo1.xsl

Si ambos ficheros están situados en la misma carpeta, al abrir con el navegador el fichero *ejemplo1.xml* nos visualizará directamente los datos contenidos en el elemento, ya que aplica su hoja de estilo (ejemplo1.xsl) y pasa a ser visualizada como un fichero html.

Fijémonos en que el título de la página es *Ejemplo*, y que todo el contenido del fichero XML, formado por el valor de las etiquetas <*nombre*> y <*url*> , ambas anidadas en la etiqueta raíz <*web_cine*>, se visualizan con formato estándar del navegador para la etiqueta <*h1*>, que es la única que hemos empleado dentro de <*body*> para hacer más simple la presentación del ejemplo.

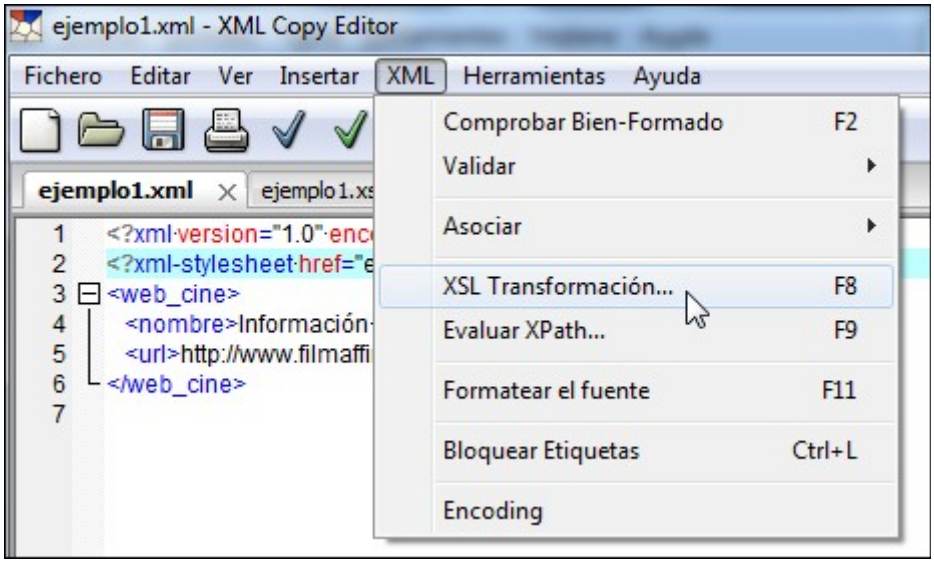
Destacando ya algunas cosas interesantes e imprescindibles para entender la sintaxis de la transformación, resaltaremos ahora los puntos básicos del fichero xsl, aunque luego serán tratados de nuevo en su apartado correspondiente.

- La sintaxis que utiliza los ficheros xsl es sintaxis XML, y debe respetar las restricciones que ya conocemos, como la de ser un documento *bien formado*.
- El elemento raíz será siempre <xsl:stylesheet.....> que contiene la versión que se está usando y la referencia al namespace declarado como ***xmlns:xsl="http://www.w3.org/1999/XSL/Transform"*** por lo que se usará, en el resto de etiquetas "propias" el prefijo ***"xsl:"***
- La etiqueta ***<xsl:template match='/'>*** hace referencia al elemento *template* cuyo atributo *match* especifica los nodos del documento origen que se utilizarán (usando para ello una expresión XPath), por lo que en este caso se refiere al elemento raíz y abarcará todo el documento, que será luego empleado al invocarlo mediante la orden ***<xsl:apply-templates />***.En la práctica, por ser este caso demostrativo, lo que hemos hecho es simplemente decirle dónde tiene que incluir (en la etiqueta <*h1*>) los valores del documento original
- En este caso, y como aclararemos posteriormente, la salida ha sido un documento html ya que por defecto, si no se dice otra cosa, si la primera etiqueta que no lleva el prefijo *xsl:* es <html> entonces el documento se convierte a HTML.

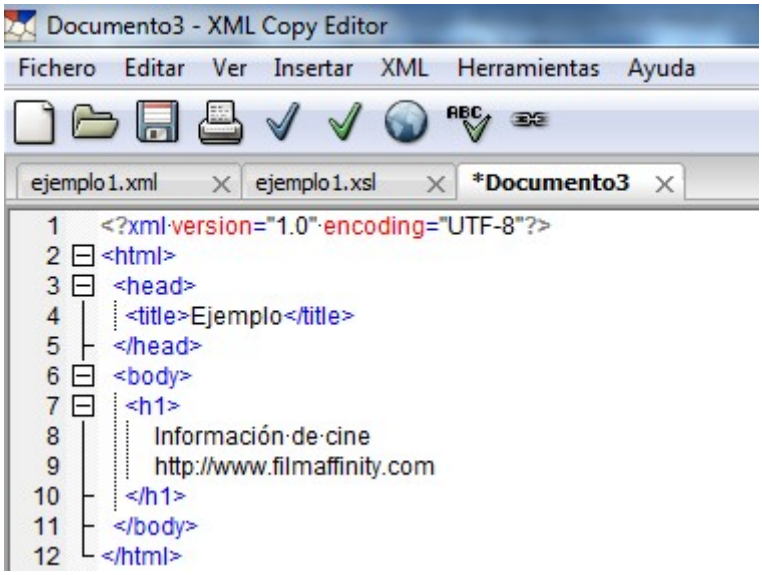
La transformación puede llevarse a cabo en el momento de la visualización de documento XML en un navegador, como acabamos de describir, pero también puede ser interesante obtener y crear un documento destino que sea el resultado de dicha transformación, especialmente en el caso de

conversión a (X)HTML, para aplicar y presentar con su CSS correspondiente. Existen varios programas que realizan esta tarea, y entre ellos el XML CopyEditor que ya conocemos, y que nos sirve para documentos sencillos.

La transformación se realiza siguiendo los pasos:

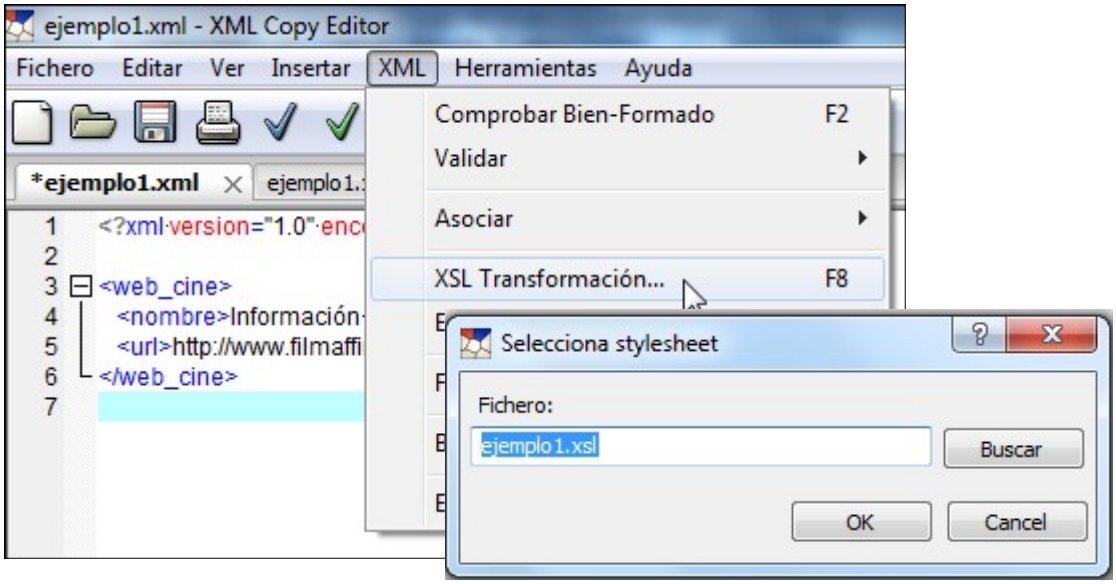


Obteniendo una nueva pestaña con el documento destino de la transformación



Que podremos guardar con el nombre que nos interese, **añadiendo la extensión .html.**

Incluso podemos (aunque no es recomendable como uso generalizado) partir de un documento XML en el que no se referencie internamente el fichero xsl de transformación, en cuyo caso al elegir la opción correspondiente se nos pedirá y podremos utilizarlo con el mismo resultado:



9.5 Qué son las plantillas

Una plantilla es un bloque utilizado con el elemento

<xsl:template match="expresión XPath">

que delimita una serie de contenidos y reglas XSL como centro de cualquier transformación.

Su importancia dentro de XSLT es de primer nivel, pues la transformación consistirá simplemente en aplicar una colección de esas plantillas al documento de entrada para obtener el correspondiente documento de salida.

El atributo **match** asociará la plantilla con un elemento XML, y es una expresión XPath (que ya vimos en el tema anterior). En el ejemplo hemos usado la etiqueta <xsl:template match='/'> por lo que hemos seleccionado el documento completo, al utilizar el símbolo del elemento raíz.

```
<xsl:template match='/'>
  <html>
  <head>
      <title>Ejemplo</title></head>
   <body>
      <h1> <xsl:apply-templates /> </h1>
  </body>
  </html>
</xsl:template>
```

Nuestra plantilla esta formada por contenidos que queremos incluir en nuestro documento de salida, como las distintas etiquetas html o en su caso el valor correspondiente ("Ejemplo" para <title>), y por *instrucciones* propias de xslt, que son las que comenzarán con el prefijo declarado en *namespace* y que en este caso es *'sxl: '*

Es decir, todo lo que está dentro de la plantilla formará la salida o resultado, trasladando todos lo elementos normales o texto tal cual están, mientras que los que tienen el prefijo *'xsl:'* indicarán al procesador que se deberá hacer alguna cosa con ellos.

En nuestro ejemplo básico anterior tenemos una única plantilla, pero una hoja de estilo puede tener todas las plantillas que se considere necesario. Su funcionamiento estará basado en el recorrido en forma de árbol de los nodos que coinciden con la expresión XPath de las sucesivas plantillas, incorporándolos en el documento de destino.

9.6 Tipos de archivo de destino

Las transformaciones XSL pueden obtener archivos de tipo ***xml*** o ***html*** (también de tipo *txt* pero no lo trataremos en este curso). Se puede especificar el método a utilizar si incluimos el elemento ***xsl:output*** con su atributo ***method***

En nuestro ejemplo anterior podríamos (y es muy conveniente) haber incluido el elemento output de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" />
   <xsl:template match='/'>
      <html>
      <head>
          <title>Ejemplo</title></head>
       <body>
          <h1> <xsl:apply-templates /> </h1>
      </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

Ejemlo1a(con elemento output especificando el metodo)

Cuando no se especifica el método de salida, como hemos hecho al principio en el ejemplo1, se sigue la siguiente norma:

- -El método por defecto cuando no se especifica ningún método, será **xml**
- -No obstante la norma anterior, cuando el primer elemento que se especifica (que no sea directamente un elemento *xsl*, es decir, que no lleva el prefijo *xsl:*) sea <html> el documento resultante será **html**

Otro atributo optativo del elemento *output* es *indent* que puede contener los valores "*yes*" o "*no*". Su valor por defecto es "*no*", por lo que si queremos que la salida se formatee con cada elemento en una línea y los elementos hijos identandose automáticamente, tendremos que poner su valor a "yes".

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

9.7 Elementos XSLT

Además de los elementos ya tratados en los puntos anteriores para explicar las declaraciones de documento y las plantillas, existen muchos otros elementos pertenecientes a la tecnología XSLT de los cuales solo veremos una selección de los mismos, correspondientes a los mas utilizados.

Para ello partiremos de otro ejemplo, también muy simple pero con alguna etiqueta más, que guardaremos como ejemplo2.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="ejemplo2.xsl" type="text/xsl"?>
<grupo>
   <nombre>Pepe</nombre>
   <nombre>Juan</nombre>
   <nombre>Luis</nombre>
   <nombre>Carmen</nombre>
</grupo>
```

ejemplo2.xml

9.7.1 apply-templates

Como hemos visto ya en el apartado 9.5 anterior, el elemento ***<xsl:apply-templates>*** se utiliza desde dentro de una plantilla para llamar a otras, y si no hay otras se aplica la plantilla por defecto que lo único que hace es incluir el contenido de las etiquetas en el documento de salida .

Su sintaxis básica es:

<xsl:apply-templates select="expresión XPath">

Si se especifica el atributo *select*, entonces se evaluará la expresión y el resultado se utilizará como nodo de contexto, y si no es especifica, se asume el nodo contexto vigente.

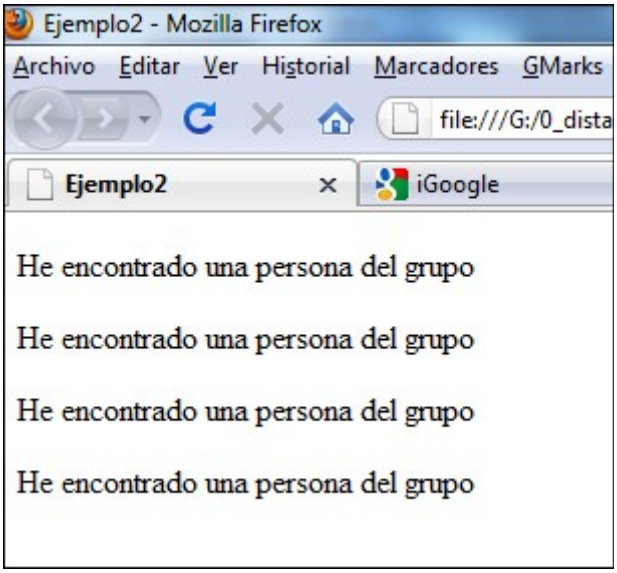
Creamos el fichero *xsl* para el documento *xml* anterior, en el que utilizara una plantilla para los elementos principales y otra plantilla se encargará de los elementos <nombre>

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match='/'>
      <html>
      <head>
          <title>Ejemplo2 </title></head>
       <body>
           <xsl:apply-templates />
      </body>
      </html>
   </xsl:template>

   <xsl:template match="nombre">
       <p> He encontrado una persona del grupo</p>
   </xsl:template>
</xsl:stylesheet>
```

ejemplo2a.xls

Al visualizar el resultado en el navegador obtenemos:



Lo que ha ocurrido es que la primera plantilla ha generado la estructura de la página, con el título en <head> y el cuerpo principal de <body>. A continuación comienza a aplicar la siguiente plantilla, por lo que creará un párrafo cada vez que se encuentre con el elemento <name> y escribirá el texto previsto para este caso.

9.7.2 value-of

En el ejemplo anterior solo queríamos resaltar el funcionamiento de las plantillas, y el ejemplo se ha limitado a enviar un mensaje cada vez que encontraba el nodo "nombre", pero mediante el elemento *value-of select* podemos utilizar el valor que se encuentra en cada nodo.

La sintaxis será:

```
<xsl:value-of select=”expresión XPath”>
```

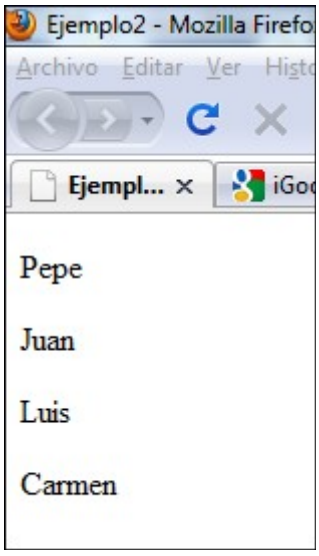
El valor del atributo **select** es una expresión **XPath**. La expresión Xpath utilizada puede servir para devolver tanto el valor del texto asociado al nodo como el del atributo correspondiente (recordemos que el atributo deber ir precedido de @).

En nuestro ejemplo anterior, podemos ahora especificar:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match='/'>
      <html>
      <head>
          <title>Ejemplo2 </title></head>
       <body>
           <xsl:apply-templates />
      </body>
      </html>
    </xsl:template>

    <xsl:template match="nombre">
        <p> <xsl:value-of select="." /> </p>
    </xsl:template>
</xsl:stylesheet>
```

que nos producirá el siguiente resultado:



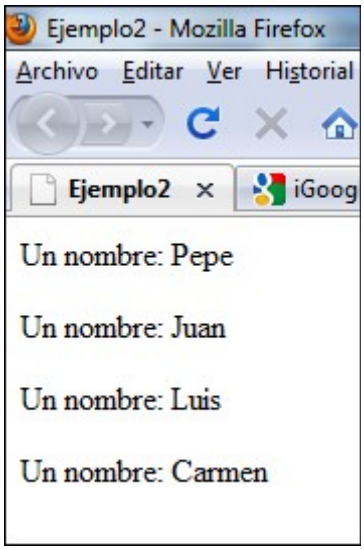
9.7.3 text

El elemento ***<xsl:text>*** inserta texto, aunque no es necesario ya que en las plantillas podemos escribir directamente el texto que queramos que se escriba en el árbol de resultado.

Aplicamos a nuestro ejemplo y modificando la segunda plantilla:

```
<xsl:template match="nombre">
      <p>
      <xsl:text > Un nombre: </xsl:text>
      <xsl:value-of select="." />
       </p>
      <xsl:text></xsl:text>
  </xsl:template>
```

produciéndose el siguiente resultado si lo visualizamos en el navegador



Aunque como hemos comentado hubiésemos obtenido el mismo resultado sin usar este elemento, simplemente insertando el texto:

```
  <xsl:template match="nombre">
     <p>
        Un nombre:
        <xsl:value-of select="." />
      </p>
     <xsl:text></xsl:text>
  </xsl:template>
```

No obstante el uso de este elemento puede ser interesante si se utiliza con el atributo que permite usar la salida de caracteres de escape *disable-output-escaping="yes"* ya que algunos editores pueden causar problemas al insertar directamente el carácter que no es PCDATA.

Por ejemplo:

```
<xsl:text disable-output-escaping="yes"> Un &amp;quot; nombre &amp;quot;: </xsl:text>
```

Permitirá conseguir la salida:

```
Un " nombre ": Pepe
Un " nombre ": Juan
Un " nombre ": Luis
Un " nombre ": Carmen
```

9.7.4 element

Aunque ya hemos visto como insertar directamente los elementos que nos interesa, podemos utilizar el elemento

```
<xsl:element name=... >
```

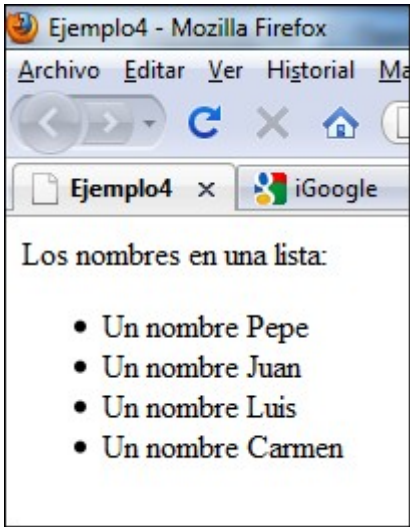
para la misma función, pero también para crear etiquetas de forma más dinámica, relativas al contexto.

Si transformamos nuestro ejemplo anterior de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match='/'>
      <html>
      <head>
          <title>Ejemplo4 </title></head>
       <body>
           Los nombres en una lista:
           <xsl:element name="ul" >
             <xsl:apply-templates />
           </xsl:element>
      </body>
      </html>
   </xsl:template>

   <xsl:template match="nombre">
      <xsl:element name="li">
          <xsl:text > Un nombre  </xsl:text>
          <xsl:value-of select="." />
      </xsl:element>
   </xsl:template>
</xsl:stylesheet>
```

obtenemos un fichero que podemos visualizar como:



aunque hubiésemos llegado a la misma situación en caso de poner directamente las etiquetas correspondientes:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match='/'>
      <html>
      <head>
          <title>Ejemplo4-sin element </title></head>
       <body>
           Los nombres en una lista:
          <ul>
            <xsl:apply-templates />
          </ul>
      </body>
      </html>
   </xsl:template>

   <xsl:template match="nombre">
        <li>
          <xsl:text > Un nombre  </xsl:text>
          <xsl:value-of select="." />
        </li>
   </xsl:template>
</xsl:stylesheet>
```

Sin embargo consideremos el caso de una simple transformación **no** a html **sino a XML** (recordemos que podemos utilizar el elemento *output* comentado en el apartado 9.6 o simplemente dejarlo por defecto, al reconocer que la primera etiqueta no es <html>), en la que queremos convertir el texto encontrado en una nueva etiqueta. Para ello utilizaríamos las llaves **{ }** :

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

    <xsl:template match="nombre">
       <xsl:element name="{.}">Este texto pertenece a la etiqueta generada automáticamente</xsl:element>
    </xsl:template>
</xsl:stylesheet>
```

La transformación resultante sería el siguiente fichero xml bien formado:

```
<?xml version="1.0" encoding="UTF-8"?>
<mis_etiquetes>
   <Pepe>Este texto pertenece a la etiqueta generada automáticamente</Pepe>
   <Juan>Este texto pertenece a la etiqueta generada automáticamente</Juan>
   <Luis>Este texto pertenece a la etiqueta generada automáticamente</Luis>
   <Carmen>Este texto pertenece a la etiqueta generada automáticamente</Carmen>
</mis_etiquetes>
```

con lo que hemos visto un ejemplo de la creación de elementos de forma dinámica.

9.7.5 attribute

De la misma forma que hemos utilizado element, para los atributos podemos emplear:

```
<xsl:attribute name=... >
```

de forma fija o dinámica. En cualquier caso, el elemento<xsl:atribute...> deberá ir siempre detrás de la etiqueta de apertura del elemento al que pertenece

Por ejemplo, si volvemos a nuestro ejemplo2 inicial y lo modificamos así:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match='/'>
      <html>
      <head>
           <title>Ejemplo 5 </title></head>
       <body>
            <xsl:apply-templates />
      </body>
      </html>
    </xsl:template>

    <xsl:template match="nombre">
       <p><xsl:attribute name="class"> destacado </xsl:attribute>
            He encontrado una persona del grupo
       </p>
    </xsl:template>
</xsl:stylesheet>
```

Obtendremos el siguiente código que nos permitirá aplicar un formato determinado si confeccionamos una CSS:

```
?xml version="1.0" encoding="UTF-8"?>
<html>
  <head>
    <title>Ejemplo2 </title>
  </head>
  <body>
    <p class=" destacado ">
            He encontrado una persona del grupo
        </p>

    <p class=" destacado ">
            He encontrado una persona del grupo
        </p>

    <p class=" destacado ">
            He encontrado una persona del grupo
        </p>

    <p class=" destacado ">
            He encontrado una persona del grupo
        </p>
</body>
</html>
```

Y también podríamos aprovechar este elemento para crear atributos de forma dinámica, por ejemplo para crear un documento xml con nombres de atributos distintos en cada elemento:

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

    <xsl:template match="nombre">
        <xsl:element name="UnNombre">
            <xsl:attribute name="{.}">valor_atributo</xsl:attribute>
        Este texto pertenece a la etiqueta generada automáticamente
        </xsl:element>
    </xsl:template>
</xsl:stylesheet>
```

Obteniendo el siguiente fichero xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<mis_etiquetes>
   <UnNombre Pepe="valor_atributo">
      Este texto pertenece a la etiqueta generada automáticamente
      </UnNombre>

   <UnNombre Juan="valor_atributo">
      Este texto pertenece a la etiqueta generada automáticamente
      </UnNombre>

   <UnNombre Luis="valor_atributo">
      Este texto pertenece a la etiqueta generada automáticamente
      </UnNombre>

   <UnNombre Carmen="valor_atributo">
      Este texto pertenece a la etiqueta generada automáticamente
      </UnNombre>
</mis_etiquetes>
```

O esta otra versión, para obtener el valor del atributo relacionado con el nodo de contexto:

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

    <xsl:template match="nombre">
       <xsl:element name="UnNombre">
            <xsl:attribute name="pertenece">
                 <xsl:value-of  select="." />
        </xsl:attribute>
       ……..Este texto pertenece a la etiqueta generada automáticamente
       </xsl:element>
    </xsl:template>
</xsl:stylesheet>
```

cuyo resultado de transformación sería:

```
<?xml version="1.0" encoding="UTF-8"?>
<mis_etiquetes>
   <UnNombre pertenece="Pepe">
      Este texto pertenece a la etiqueta generada automáticamente
      </UnNombre>

   <UnNombre pertenece="Juan">
      Este texto pertenece a la etiqueta generada automáticamente
      </UnNombre>

   <UnNombre pertenece="Luis">
      Este texto pertenece a la etiqueta generada automáticamente
      </UnNombre>

   <UnNombre pertenece="Carmen">
      Este texto pertenece a la etiqueta generada automáticamente
      </UnNombre>
</mis_etiquetes>
```

9.7.6 copy-of

Especialmente cuando queremos transformar un XML en otro documento también XML, si hay secciones largas en las que el contenido es exactamente igual, esta instrucción nos permite tomar fragmentes del origen y traspasarlos íntegramente al al destino, sin tener que crear los elementos.

Su sintaxis básica es:

```
<xsl:copy-of select=”expresión XPath”/>
```

Por ejemplo, aplicando la siguiente transformación al fichero XML original que estamos usando desde el principio, podemos hacer:

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

Con lo que obtenemos un fichero idéntico al anterior.

Este elemento tendrá más sentido cuando se utilice dentro de los elementos condicionales.

9.7.7 copy

Este elemento, a diferencia del anterior, efectua una copia "hueca" ya que solo copia la etiqueta del elemento, mientras que los atributos y nodos hijos son ignorados. No tiene atributo *select* ya que siempre copia el elemento actual dentro de un "*template*" o como veremos más adelante, en un bucle o elemento de control. Además resulta útil cuando queremos transformar un elemento a un XML similar pero eliminando los atributos.

Si aplicamos el siguiente ejemplo al al fichero XML original que estamos usando desde el principio:

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

    <xsl:template match="grupo">
            <xsl:copy />
    </xsl:template>
</xsl:stylesheet>
```

El fichero obtenido contendrá simplemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<grupo/>
```

Pero si incorporamos **<xsl:value-of select=".">** se obtiene el contenido del nodo de contexto **más el de todos los nodos descendientes**:

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

    <xsl:template match="grupo">
              <xsl:copy>
                  <todos_mis_amigos>
                        <xsl:value-of select=".">
                  </todos_mis_amigos>
              </xsl:copy>
    </xsl:template>
</xsl:stylesheet>
```

y ya a podemos utilizar el contenido de los nodos hijos del elemento grupo, obteniendo un xml con el siguiente contenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<grupo>
      <todos_mis_amigos>
            Pepe
            Juan
            Luis
            Carmen
    </todos_mis_amigos>
</grupo>
```

9.8 Elementos de control (2ª parte)

En XSLT se pueden usar filtros y estructuras que facilitan las operaciones de control del contenido de los documentos. Estas operaciones de control son habituales en los lenguajes de programación tradicionales, y se conocen como:

- Iteraciones (o repeticiones o bucles)
- Selecciones (o alternativas)

La lógica que algoritmicamente corresponde a estas estructuras de control pueden ser complicada si no se ha practicado previamente algún lenguaje de programación, pero en este caso usaremos ejemplos con construcciones muy sencillas para que puedan ser entendidas sin ese requisito previo.

9.8.1 Iteraciones: *for-each*

Este elemento se usa para repetir la búsqueda de los nodos que coinciden con la expresión XPath que se usa en el atributo *select* , de forma que solo escribimos una vez el código que comprende la instrucción, pero se aplica a todos los casos en que se cumpla la expresión.

La sintaxis es:

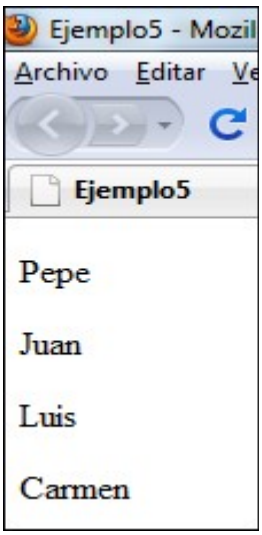
```
<xsl:for-each select=”expresión XPath”>
```

En algunos casos puede ser una alternativa más clara a la llamada de nuevas plantillas como en el ejemplo utilizado en el apartado 7.2, y que podría quedar más simplificado de esta forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet                                                    version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match='/'>
      <html>
      <head>
          <title>Ejemplo5 </title></head>
       <body>
            <xsl:for-each select="grupo/nombre">
                  <p> <xsl:value-of select="." /> </p>
            </xsl:for-each>
      </body>
      </html>
    </xsl:template>
</xsl:stylesheet>
```

En el que simplemente hemos sustituido la segunda plantilla por la estructura *for-each* completa, que ya contiene el elemento seleccionado con *value-of*. Es decir, el contenido de <xsl:for-each ...> hace la función de una plantilla anidada en otra anterior, con lo que se simplifica el

documento. Como podemos ver, el resultado es idéntico al obtenido en el caso anterior:



9.8.2 Selecciones: *if test* y *chose*

Disponemos también de elementos que nos permiten decidir si una acción se realizará o no dependiendo de ciertos criterios, es decir, generar contenido condicionalmente.

La sintaxis básica es

```
<xsl:if test="expression Xpath">
…….....
 </xsl:if>
```

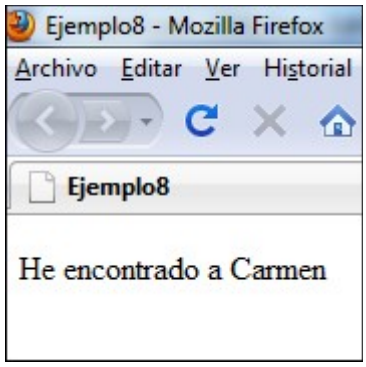
El atributo `test` es obligatorio y contiene, en sintaxis XPath la condición que se tiene que cumplir.

Un ejemplo sencillo ejemplo, restringiendo la búsqueda a un determinado nombre:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet                                                    version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match='/'>
      <html>
      <head>
          <title>Ejemplo8</title></head>
       <body>
           <xsl:for-each select="grupo/nombre">
              <xsl:if test=".='Carmen' ">
                  <p>He encontrado a <xsl:value-of select="." /> </p>
              </xsl:if>
           </xsl:for-each>
      </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

Notar que se han tenido que alternar las comillas ' y “ para abarcar tanto las correspondientes a la expresión propia de la condición de **test** como a la del XPath que contiene, ya que al anidar unas comillas dentro de otras, deben ser distintas.

De esta forma obtenemos la siguiente salida:



Si queremos disponer de alguna alternativa en caso de que no se cumpla la condición, necesitaríamos utilizar la construcción *chose*, que además de poder elegir entre varias alternativas, nos deja planificar la acción a realizar si no se cumple ninguna de ellas.

La sintaxis básica es:

```
<xsl:choose>
        <xsl:when test=”expresión booleana”>
                ….
         </xsl:when>
        <xsl:when test=”expresión booleana”>
                ….
         </xsl:when>
                ….
        <xsl:otherwise>
                ….
         </xsl: otherwise>

</xsl:xhoose>
```

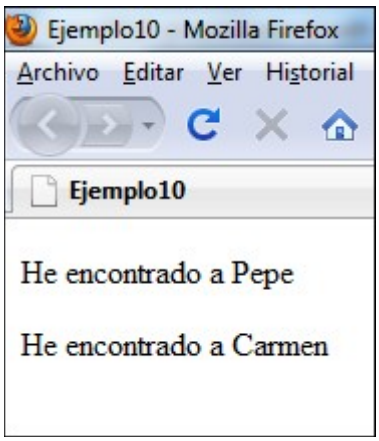
En esta estructura podemos tener tantas opciones *when text=...* como sean necesarias, y el funcionamiento sera de tal forma que se irán comprobando secuencialmente sus valores. Si ninguna de ellas se cumple, y existe el elemento *otherwise*, que es optativo, entonces esta sería la acción que se realizaría.

Veamos una sencilla aplicación en nuestro ejemplo anterior para hacer más de una selección:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet                                                    version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match='/'>
      <html>
      <head>
          <title>Ejemplo10 </title>
       </head>

      <body>
            <xsl:for-each select="grupo/nombre">
                        <xsl:choose>
                        <xsl:when test=".='Carmen'">
                         <p>He encontrado a
                              <xsl:value-of select="." /> </p>
                        </xsl:when>
                         <xsl:when test=".='Pepe'">
                         <p>He encontrado a
                              <xsl:value-of select="." /> </p>
                         </xsl:when>
                        </xsl:choose>
            </xsl:for-each>
      </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

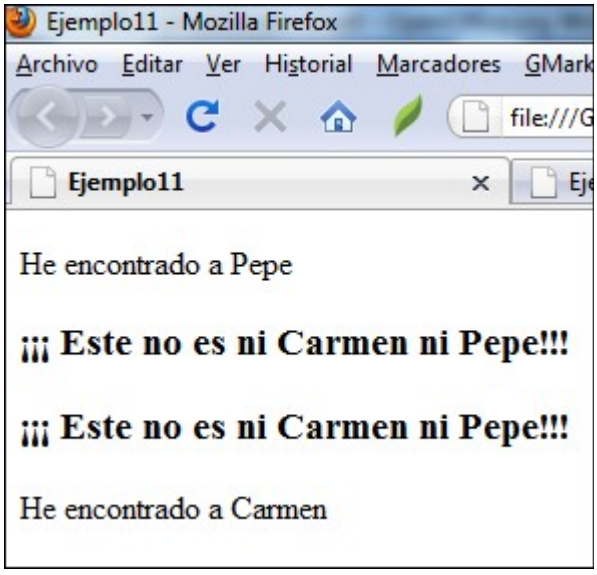
con el que obtenemos:



O aplicado al caso en que queremos dar un mensaje concreto en caso de no encontrar el nombre buscado:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet                                                    version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match='/'>
      <html>
      <head>
          <title>Ejemplo11 </title>
       </head>
      <body>
            <xsl:for-each select="grupo/nombre">
                        <xsl:choose>
                        <xsl:when test=".='Carmen'">
                         <p>He encontrado a
                              <xsl:value-of select="." /> </p>
                        </xsl:when>
                         <xsl:when test=".='Pepe'">
                         <p>He encontrado a
                              <xsl:value-of select="." /> </p>
                         </xsl:when>
                         <xsl:otherwise>
                            <h3>¡¡¡ Este no es ni Carmen ni Pepe!!!</h3>
                         </xsl:otherwise>
                       </xsl:choose>
           </xsl:for-each>
      </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

y obtenemos:



9.9. Ordenación: sort

Este elemento se usa para alterar el orden de presentación de los elementos, siguiendo un criterio de clasificación indicado en el atributo *select,* y *s*e añade anidado dentro de un elemento <*xsl:for-each....*>, como primer hijo de ese elemento.

Su sintaxis completa es:

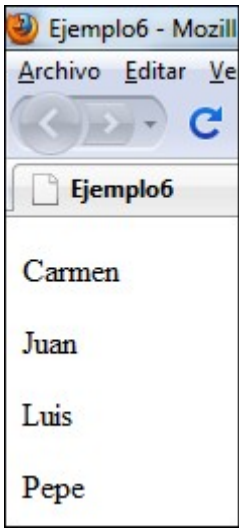
```
<xsl:sort select=”expresión XPath” lang=”lang-code” data-type=”text|number”
order=”ascending|descending” "/>
```

Aunque solo suele usarse el primer atributo, siendo optativos los demás.

Así, añadiendo a nuestro ejemplo anterior este elemento tenemos:

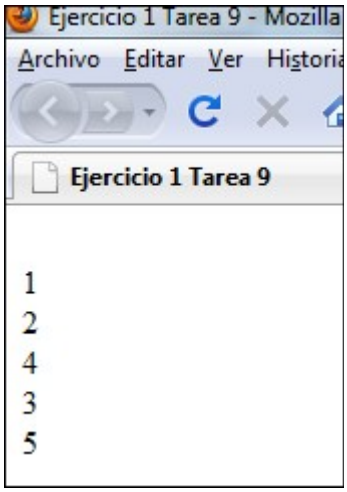
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet                                                    version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match='/'>
      <html>
      <head>
          <title>Ejemplo6 </title></head>
       <body>
           <xsl:for-each select="grupo/nombre">
           <xsl:sort select="."/>
                  <p> <xsl:value-of select="." /> </p>
           </xsl:for-each>
      </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

Y en este caso el resultado contendría una salida de los nombres ordenados alfabéticamente en orden ascendente:

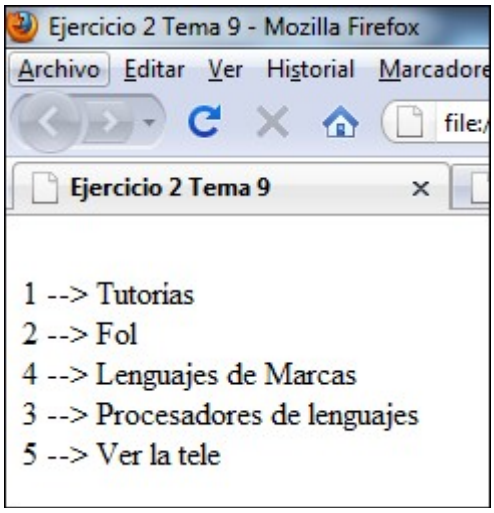


Ejercicios básicos primera parte del tema 9

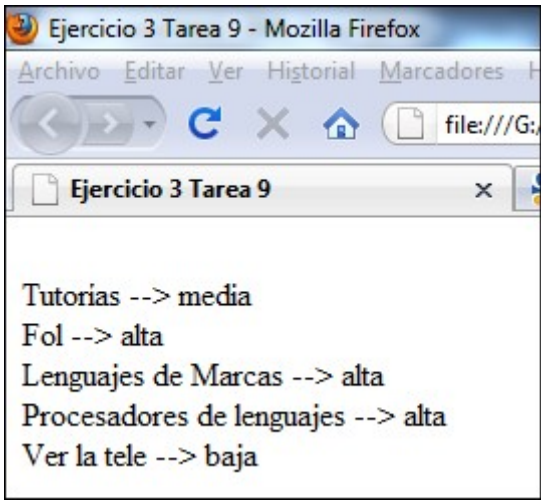
Ejercicio 1- Con el fichero XML original, realiza la transformación necesaria para obtener una página html como la siguiente que visualiza el contenido de la etiqueta *numdia*

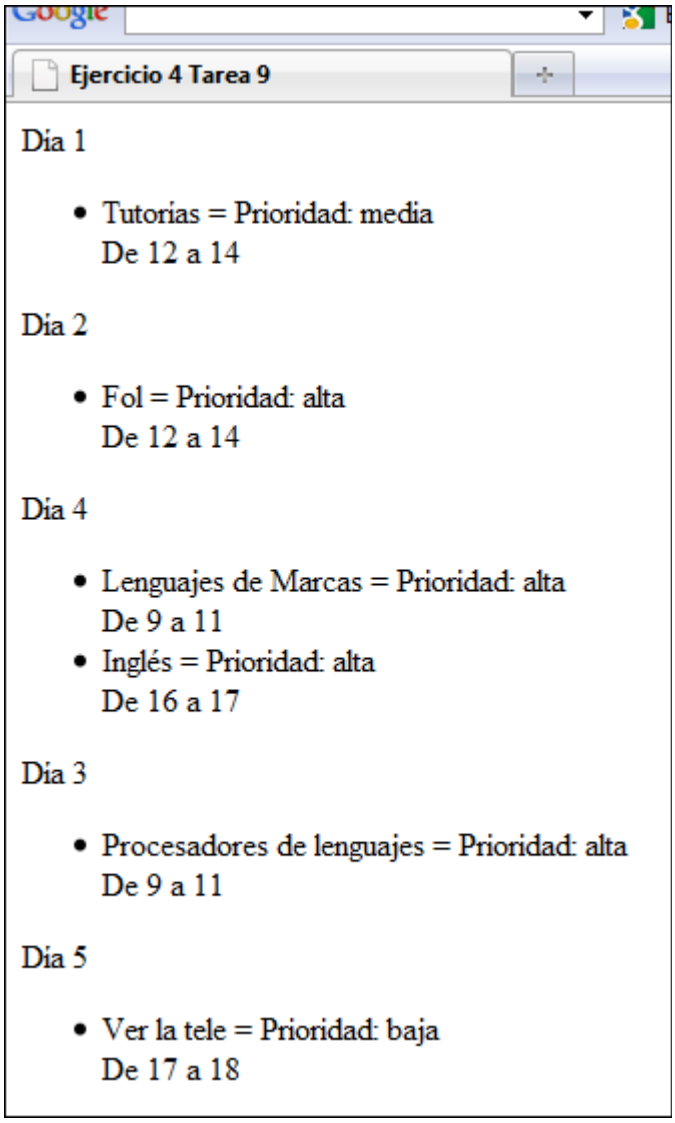


Ejercicio 2 – Con el fichero XML original realiza la transformación necesaria para obtener una página html como la siguiente:



Ejercicio3 - Con el fichero XML original realiza la transformación necesaria para obtener una página html como la siguiente:

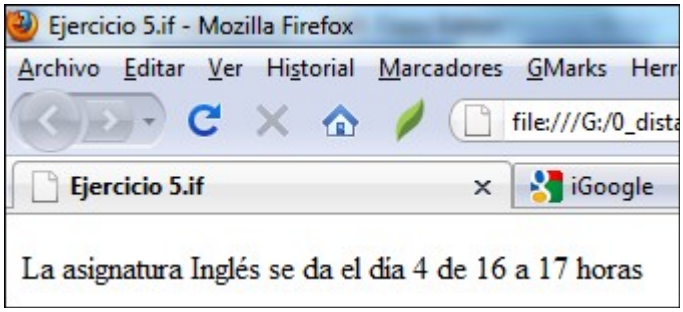
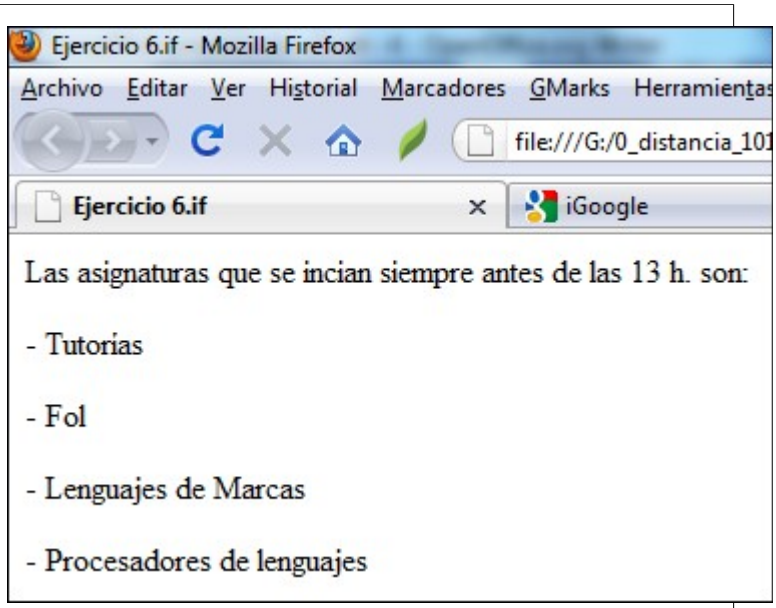


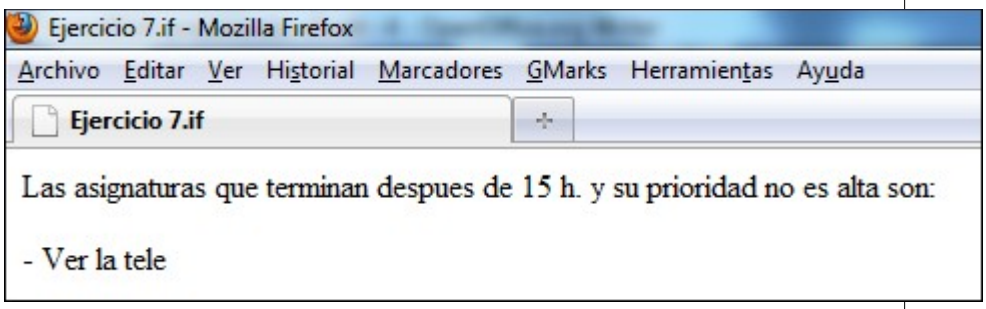
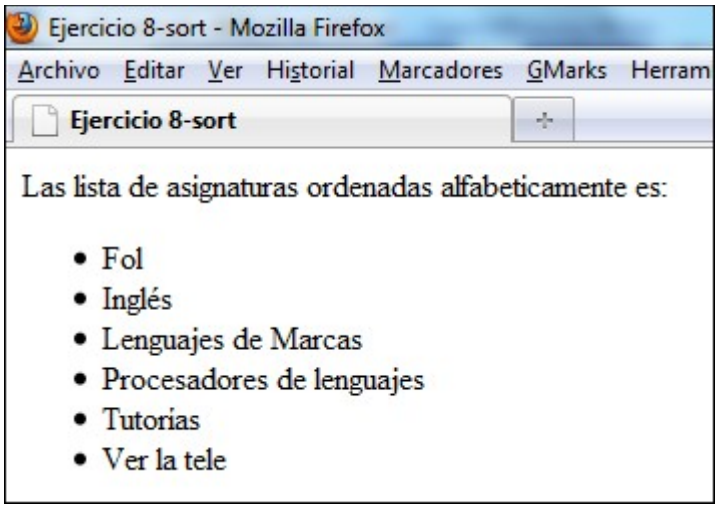
Ejercicio4 - Con el fichero XML original realiza la transformación necesaria para obtener una página html como la siguiente:	

ejercicios 2ª parte. Intenta utilizar sentencias de control (for, if, chose,sort)

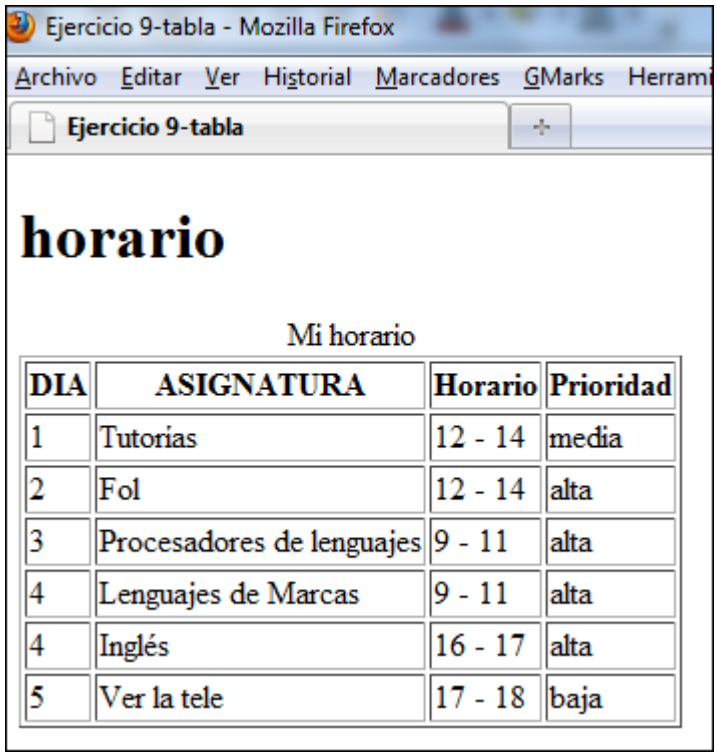
Recuerda algunas cosas importantes:

- Si tienes que usar comillas anidadas dentro de la misma expresión, tendrás que alternar las comillas simples (') con las dobles("), como ya se comento en el ejemplo del apartado 9.8.2 (ejmplo *if test="."='Carmen' "*)
- Cuidado al usar en las expresiones símbolos especiales, como los de > (mayor) o < (menor). Para evitar errores en algunas ocasiones tendrás que sustituirlos por la refencia a entidad correspondiente, como ***>*** o ***<***
- Ten en cuenta la sintaxis de especificación del *camino relativo*.

Ejercicio5 - Con el fichero XML original realiza la transformación necesaria para obtener una página html como la siguiente:	
Ejercicio6 - Con el fichero XML original realiza la transformación necesaria para obtener una página html como la siguiente:	

Ejercicio7 - Con el fichero XML original realiza la transformación necesaria para obtener una página html como la siguiente:	
Ejercicio8 - Con el fichero XML original realiza la transformación necesaria para obtener una página html como la siguiente:	

Ejercicio9 - Con el fichero XML original realiza la transformación necesaria para obtener una página html como la siguiente, en forma de tabla:



horario

Mi horario

DIA	ASIGNATURA	Horario	Prioridad
1	Tutorias	12 - 14	media
2	Fol	12 - 14	alta
3	Procesadores de lenguajes	9 - 11	alta
4	Lenguajes de Marcas	9 - 11	alta
4	Inglés	16 - 17	alta
5	Ver la tele	17 - 18	baja

Ejercicio 10- Modifica el ejercicio anterior para visualizar el día según el nombre del día de la semana a que corresponde y cambiar el horario por el turno a que corresponde, así como la interpretación de la prioridad por la importancia que corresponde, según se muestra:

horario

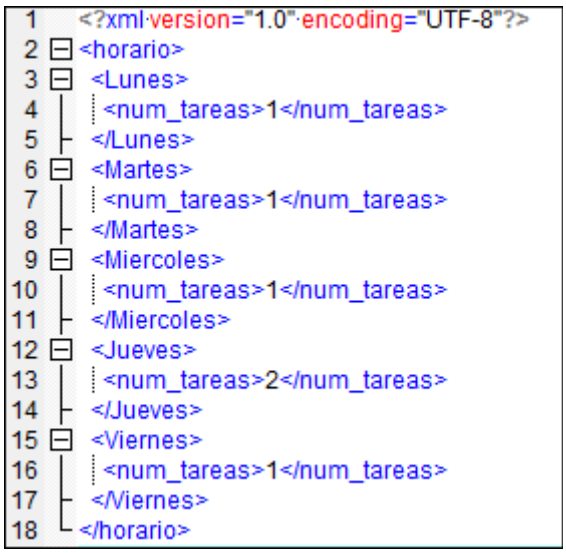
Mi horario

DIA	ASIGNATURA	Turno	Importancia
Lunes	Tutorias	Mañanas	Normal
Martes	Fol	Mañanas	Muy importante
Miércoles	Procesadores de lenguajes	Mañanas	Muy importante
Jueves	Lenguajes de Marcas	Mañanas	Muy importante
Jueves	Inglés	Tardes	Muy importante
Viernes	Ver la tele	Tardes	Poco importante

Ejercicio 11 – Transforma el fichero XML original en otro XML con el siguiente contenido

```
<?xml version="1.0" encoding="UTF-8"?>
<horario>
<Lunes>
<tarea prioridad="media">
<hora-ini>12</hora-ini>
<hora-fin>14</hora-fin>
<asignatura>Tutorías</asignatura>
</tarea>
</Lunes>
<Martes>
<tarea prioridad="alta">
<hora-ini>12</hora-ini>
<hora-fin>14</hora-fin>
<asignatura>Fol</asignatura>
</tarea>
</Martes>
```

	y resto...

Ejercicio 12 - Transforma el fichero XML original en otro XML con el siguiente contenido:	

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <horario>
3    <Lunes>
4      <num_tareas>1</num_tareas>
5    </Lunes>
6    <Martes>
7      <num_tareas>1</num_tareas>
8    </Martes>
9    <Miercoles>
10     <num_tareas>1</num_tareas>
11   </Miercoles>
12   <Jueves>
13     <num_tareas>2</num_tareas>
14   </Jueves>
15   <Viernes>
16     <num_tareas>1</num_tareas>
17   </Viernes>
18 </horario>
```