XSLT

Sitio: Aula Virtual do IES de Teis
Curso: Linguaxes de Marcas e Sistemas de Xestión de Información 2021-22 (DAM-A)
Libro: XSLT

Impreso por: Deivid Durán Durán
Data: Venres, 24 de Xuño de 2022, 00:19

Táboa de contidos

1. XSLT

XSLT (Xtensible Stylesheet Language Transformations: Lenguaje Extensible de Transformaciones de Hojas de Estilos) es un **lenguaje declarativo**, es decir, no utiliza una secuencia de instrucciones sino como una colección de plantillas (*template rules)*. Cada plantilla establece cómo se transforma un determinado elemento (definido mediante expresiones XPath).

- Los documentos **XSLT** se denominan también **hojas XSLT**
- Las hojas XSLT también **son documentos XML** y usan la extensión de archivo **.xsl**
- **XSLT define las reglas de transformación** que se van a aplicar al documento XML origen.

- Para realizar las transformaciones, es necesario **un procesador XSLT**: Un procesador XSLT es un programa software que toma como entrada un documento **XSLT** y otro **XML (eXtensible Markup Language, que significa Lenguaje Extensible de Marcado.)**, y crea un documento de salida aplicando las instrucciones de la hoja de estilos **XSLT** a la información del documento **XML**.

Pueden estar integrados dentro de un explorador Web, en un servidor web, puede ser un programa que se ejecuta desde la línea de comandos o puede ser una librería (conjunto de funcionalidades software) integrada con un editor.

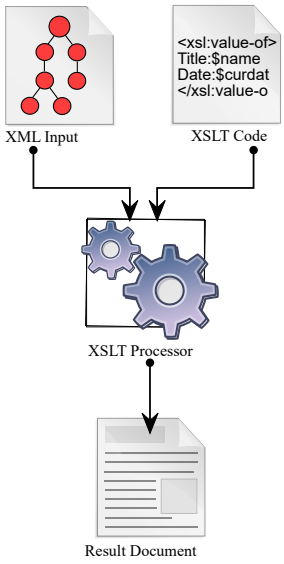


Diagram of the basic elements and processing flow of XSL Transformations, Dreftymac at English Wikipedia, bajo Creative Commons Attribution-Share Alike 3.0 Unported license

Mediante XSLT, es posible transformar un documento XML a:

- Otro documento XML
- Un documento HTML
- Un documento de texto

2. Introducción a XSLT

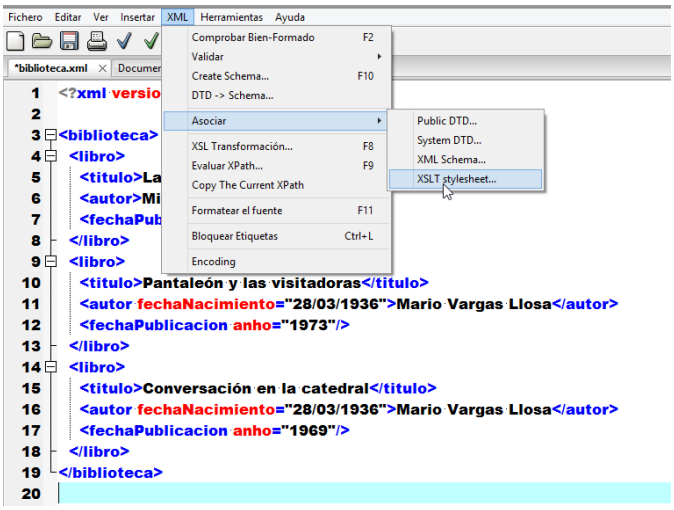
Para ver una Introducción a XSLT vamos a ver el recurso XSLT: Transformaciones XSL de McLibre (salvo el apartado de la instrucción <xsl:strip-space> que no tiene el mismo comportamiento en ninguno de los dos editores que utilizaremos: XML Copy Editor y Notepad++ con el plugin de XML Tools)

3. Herramientas de transformación

Podemos realizar una transformación XSLT con varios editores:

- XML Copy Editor

Desde el documento XML, podemos asociar un documento .xsl:

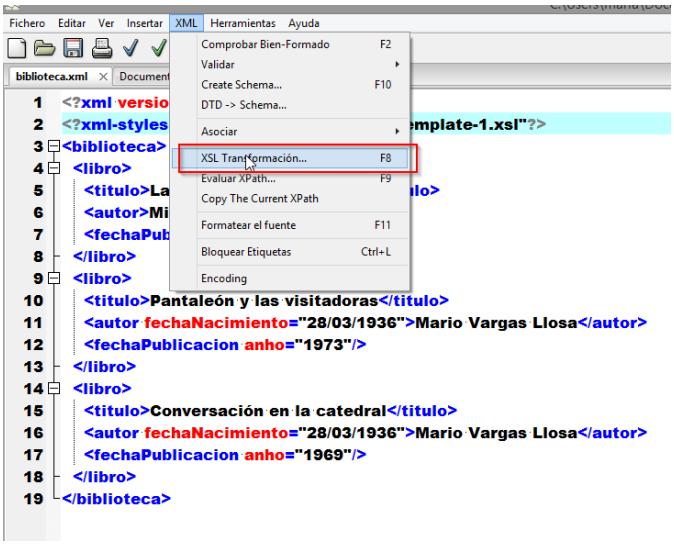


Nos añadirá una ruta absoluta al fichero .xsl que deberemos convertir a una ruta relativa desde el punto de vista del fichero XML.

<?xml-stylesheet type="text/xsl" href="file:/C:/Users/maria/Documents/Curso%2021-22/mariaferroteis/UDs/UD5/biblioteca/template-1.xsl"?>
=>
<?xml-stylesheet type="text/xsl" href="template-1.xsl"?>

Asumiendo que ambos ficheros están en la misma ubicación.

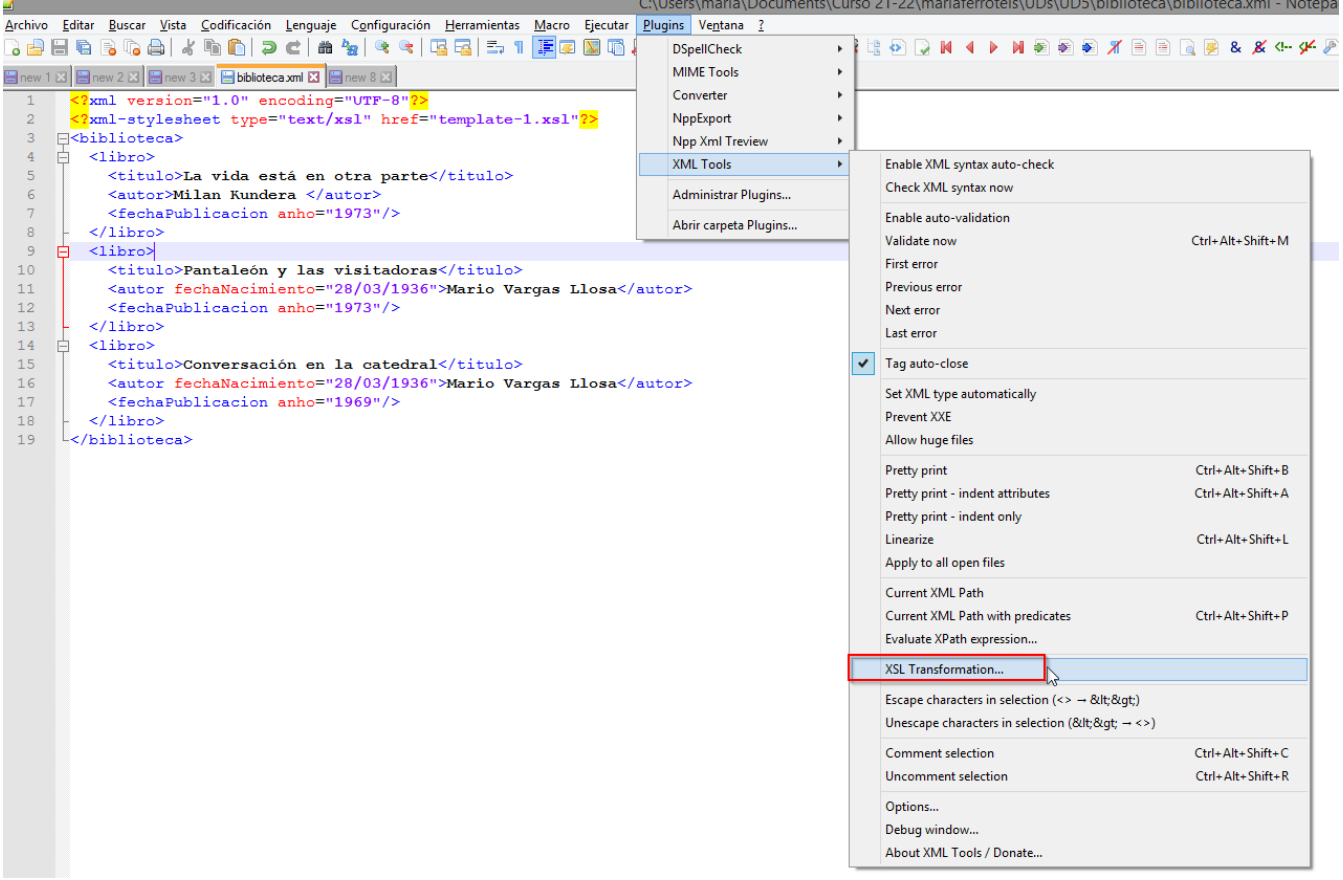
Para realizar la transformación:



- Notepad++ con XML Tools

En este caso, tendremos que cambiar la ñ por nh porque el editor no interpreta el carácter ñ como carácter válido en el nombre de un atributo.

Para realizar la transformación usaremos los menús:



A continuación, tendremos que seleccionar el archivo con el que se llevará a cabo la transformación.

4. Instrucciones básicas XSLT

Además de **<xsl:template>**, como se menciona en la introducción a XSLT de McLibre, algunas de las instrucciones básicas de XSLT que podemos destacar son:

<xsl:output>: Es un elemento de primer nivel, hijo directo de *stylesheet*. Define el formato de salida del documento: XML, text o HTML. Por defecto, será XML. Se pueden definir las versiones de XML o HTML, la codificación de caracteres, etc.

```
<?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"
encoding="UTF-8" indent="yes"/>
...

...

</xsl:stylesheet>
```

<xsl:text>: Permite incluir texto en el documento resultante. También se puede escribir el texto directamente en la plantilla, pero utilizando este elemento se pueden controlar los espacios y saltos de línea generados.

- Nueva línea:

- Tabulación: 	
- Espacio:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">

  <xsl:template match="/">
      Aquí tienes un ejemplo
     <xsl:text>Aquí &#xa; otro</xsl:text> <!--&#xa; creará una nueva línea -->
  </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <xsl:text disable-output-escaping="yes"> &lt;!DOCTYPE raiz&gt; </xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

Con el parámetro **disable-output-escaping="yes"** se evita codificar caracteres como < o > y se escribirán con sus símbolos correspondientes. El extracto anterior tendrá por resultado lo siguiente:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE raiz>
```

<xsl:element>: Crea un nuevo elemento en el documento resultante. El atributo **name** indica el nombre del nuevo elemento. Este atributo se puede componer como combinación de texto, partes del documento original, variables, valores devueltos por funciones, etc.

Por ejemplo, para crear un nuevo documento XML que cree un documento como el siguiente sobre el documento **biblioteca.xml**:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <library>
3    <book>
4      <author>Milan Kundera </author>
5    </book>
6    <book>
7      <author>Mario Vargas Llosa</author>
8    </book>
9    <book>
10     <author>Mario Vargas Llosa</author>
11   </book>
12 </library>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="3.0">
  <xsl:template match="/">
    <xsl:element name="library">
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:template>
```

```
  <xsl:template match="libro">
    <xsl:element name="book">
      <xsl:element name="author">
        <xsl:value-of select="autor"/>
      </xsl:element>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

xsl:comment: Permite introducir comentarios en el documento de salida con el formato <!-- comentario ... -->

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="3.0">
  <xsl:template match="/">
    <xsl:element name="library">
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:template>
  <xsl:template match="libro">
    <xsl:element name="book">
    <xsl:comment>Por cada autor, creamos una etiqueta &lt;author> </xsl:comment>
      <xsl:element name="author">
        <xsl:value-of select="autor"/>
      </xsl:element>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

<xsl:value-of>: Extrae el valor de un nodo o atributo **desde el contexto actual**. Tiene un atributo obligatorio **select** cuyo valor es una expresión XPath para seleccionar el nodo del que se obtendrá el valor.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="3.0">
  <xsl:template match="/">
    <xsl:element name="library">
  <xsl:attribute name="total">
  <xsl:value-of select="count(biblioteca/libro)"></xsl:value-of>
  </xsl:attribute>
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

Obtendríamos el siguiente resultado:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <library total="3"/>
```

<xsl:apply-templates>: Indica que se apliquen las plantillas que correspondan a los nodos hijos del nodo del contexto actual. Tiene un par de atributos:

- **select**: Indica los nodos a los que se les aplicará una plantilla. Por defecto, si se omite el atributo select, se buscarán y aplicarán las plantillas a todos los nodos hijo del nodo del contexto actual. Si se usa *, selecciona por completo el conjunto de nodos
- **mode**: Si existen diferentes maneras definidas de procesar el mismo nodo, se distingue entre ellas.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="3.0">
  <xsl:template match="/">
    <xsl:element name="library">
      <xsl:apply-templates select="//titulo" />
    </xsl:element>
  </xsl:template>


  <xsl:template match="libro">
    <xsl:element name="book">
      <xsl:element name="author">
        <xsl:value-of select="autor"/>
      </xsl:element>
    </xsl:element>
  </xsl:template>


   <xsl:template match="titulo">
    Título: <xsl:value-of select="."></xsl:value-of>
  </xsl:template>

</xsl:stylesheet>
```

Procesará en el nodo raíz, solo las plantillas que se puedan aplicar a los descendientes de cualquier nivel **titulo**, aplicando la plantilla de titulo y no la de libro:

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
    Título: La vida está en otra parte
    Título: Pantaleón y las visitadoras
    Título: Conversación en la catedral</library>

```

Para ver un ejemplo de como aplicar dos plantillas diferentes al mismo nodo, usamos el atributo **mode** para indicar con qué plantilla se transformará el nodo. Para ello debemos usar **mode** tanto en la definición de la plantilla como en <xsl:apply-templates>:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="3.0">
  <xsl:template match="/">
    <xsl:element name="library">
      <xsl:apply-templates mode="en" />
      <xsl:apply-templates mode="fr" />
    </xsl:element>
  </xsl:template>

  <xsl:template match="libro" mode="en">
    <xsl:element name="book">
      <xsl:element name="author">
        <xsl:value-of select="autor"/>
      </xsl:element>
    </xsl:element>
  </xsl:template>

  <xsl:template match="libro" mode="fr">
    <xsl:element name="livre">
      <xsl:element name="auteur">
        <xsl:value-of select="autor"/>
      </xsl:element>
    </xsl:element>
  </xsl:template>
```

Obteniendo el siguiente resultado:

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
    <book><author>Milan Kundera </author></book>
    <book><author>Mario Vargas Llosa</author></book>
    <book><author>Mario Vargas Llosa</author></book>

    <livre><auteur>Milan Kundera </auteur></livre>
    <livre><auteur>Mario Vargas Llosa</auteur></livre>
    <livre><auteur>Mario Vargas Llosa</auteur></livre>
</library>
```

Para consultar más elementos y más información y ejemplos de cada uno de los elementos que podemos usar en XSLT, se puede consultar la referencia de elementos XSLT 1.0 en w3schools que también tenéis en la sección de Recursos.

4.1. Attribute value templates (AVT)

En el recurso de introducción de McLibre de XSLT se introducía cómo crear un atributo en el resultado final con la instrucción **<xsl:attribute>**.

Sin embargo, es posible utilizar una sintaxis más sencilla usando *Attribute value templates o AVT.*

En lugar de crear un enlace de este modo:

```
<a>
 <xsl:attribute name="href">
     <xsl:value-of select="../link"/>
</xsl:attribute>
     Texto aquí
</a>
```

Podemos crearlo usando simplemente:

```
 <a href="{../link}">
          <xsl:value-of select="../title"/>
Texto aquí
        </a>
```

Al parecer, esta sintaxis recibe el nombre de Attribute value templates y permite evaluar el contenido entre llaves como una expresión XPath.

Tenéis un enlace a la documentación de la version=3.0 aquí, aunque en ella se menciona que también está presente en la versión 1.0. Además incluye el siguiente ejemplo:

```
<xsl:variable name="image-dir" select="'/images'"/>

<xsl:template match="photograph">
  <img src="{$image-dir}/{href}" width="{size/@width}"/>
</xsl:template>
```

With this source

```
<photograph>
  <href>headquarters.jpg</href>
  <size width="300"/>
</photograph>
```

the result would be

```
<img src="/images/headquarters.jpg" width="300"/>
```

Para el valor del atributo ***src*** se usa:

- El valor de la variable **$image-dir** que se envuelve entre llaves para que se evalúe y se reemplace por su contenido (la cadena '/images')
- El carácter literal "/"
- Una AVT con la expresión XPath **href** entre llaves, que ha de evaluarse en el contexto actual generado por el atributo match="photograph", por lo tanto, hará referencia al subelemento ***href*** de ***photograph*** y las llaves forzarán la evaluación del subelemento ***href*** de la misma forma que lo haría **<xsl:value-of select="href">**

5. Instrucciones avanzadas

<xsl:variable>: Podemos declarar una variable, como en cualquier lenguaje de programación. Puede ser:

- **global:** Si se declara como hija directa de stylesheet
- **local:** Si está dentro de una plantilla

Debe tener el a**tributo obligatorio *name*** para darle un nombre a la variable. Se le puede asignar un valor mediante:
- El atributo opcional ***select*** que puede ser un valor literar o una expresión XPath
- El contenido de la propia etiqueta <xsl:variable>, por ejemplo, una plantilla.

Para utilizar la variable, usaremos $ seguido del valor del atributo **name: $color.** En los casos en los que se pueda confundir con su valor literal como cadena de texto, necesitará envolverse entre llaves {}. Por ejemplo: **{$color}**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="3.0">
  <xsl:output method="html"  encoding="utf-8" indent="yes"/>
  <!--Usamos una variable global llamada color y le asignamos como valor la cadena de texto 'red' -->
  <xsl:variable name="color" select="'red'"/>
     <xsl:template match="/">
        <xsl:text disable-output-escaping='yes'>&lt;!DOCTYPE html></xsl:text>
        <html>
            <head>
             <meta charset="UTF-8"/>
             <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
             <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
              <title>Document</title>
         </head>
            <body>
               <h1 style="background-color:{$color}">Biblioteca</h1>
            </body>
          </html>
    </xsl:template>
</xsl:stylesheet>
```

En este segundo ejemplo, se usa una variable cuyo valor se establece a través de una expresión XPath:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="3.0">
  <xsl:output method="html"  encoding="utf-8" indent="yes"/>
  <!--Usamos una variable global-->
  <xsl:variable name="contador" select="count(//libro)"/>
     <xsl:template match="/">
        <xsl:text disable-output-escaping='yes'>&lt;!DOCTYPE html></xsl:text>
        <html>
            <head>
             <meta charset="UTF-8"/>
      <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
      <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
      <title>Document</title>
            </head>
            <body>
    <h1>Biblioteca</h1>
    <p>Hay <xsl:value-of select="$contador" ></xsl:value-of> libros</p>

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

.

5.1. Instrucciones de control

- **<xsl:for-each select="xpath_expression">:** Permite iterar por cada nodo de un conjunto de nodos. Esta instrucción marca un nuevo contexto actual con la expresión XPath del atributo *select*, de la misma forma que el atributo *match* de template marca el contexto actual.
- **<xsl:if test="expression">:** Permite evaluar la expresión booleana. Si se evalúa a true, el contenido de la expresión if, formará parte del resultado final. En esta instrucción, no existe la opción de evaluar el caso contrario, si la expresión se evalúa a false.
- **<xsl:choose>:** Permite evaluar múltiples condiciones a través de diferentes instrucciones **<xsl:when test="xpath_expression">.** También permite añadir **<xsl:otherwise>** en caso de que ninguna de las instrucciones **<xsl:when test="xpath_expression">.** se hayan evaluado a true. Sería similar a

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

         <xsl:when test="price >10">
        ....
        </xsl:when>

        <xsl:otherwise>
        ....
        </xsl:otherwise>
 </xsl:choose>
```

Vamos a ver un ejemplo que aúna este tipo de instrucciones para crear un documento HTML: **ejemploUD5-2022.zip**:

La hoja XSLT procesa el documento **ejemploUD5.xml, que contiene las notas de varios alumnos** y genera un documento HTML5 que contiene una tabla como muestra la imagen:

Ejemplo UD05

Convocatoria de Junio

Datos		Notas		
Nombre y apellidos	**Tareas**	**Cuestionarios**	**Examen**	**Final**
Jose Juncal Jarrón	4.5	3.0	4.0	INSUFICIENTE
Ana Antas Aparicio	9.0	8.0	9.0	NOTABLE
Esther Egea Écija	9.8	9.0	9.3	SOBRESALIENTE
Manuel Muñoz Miño	3.0	2.0	2.0	BIEN
Media numérica de la nota final de alumnos de Junio				7.06

En la transformación, se lleva a cabo lo siguiente:

1. Se utiliza una variable para mostrar el comienzo del código HTML5.
2. Se muestra en una tabla HTML la lista de los alumnos **solo de la convocatoria de junio**, en este caso, mediante un for-each.
3. El nombre y apellidos aparecen en la primera celda
4. La nota final se transforma a texto de la siguiente forma:
    - Si es menor que 5: INSUFICIENTE
    - Si final >=5 y final <6: SUFICIENTE
    - Si final >=7 y final <9: NOTABLE
    - Si final >=9: SOBRESALIENTE
    - En otro caso: ERROR

5. Se aplica el estilo 'suspenso' a las celdas (<td>) de la nota final de cada alumno si el alumno tiene una nota <5. En caso contrario, se aplica el estilo 'aprobado'. Esto se hace por medio de una variable.
6. En la última fila, se muestra la media numérica de la nota final de los alumnos cuya convocatoria ha sido Junio. Se ha de tener en cuenta que, si no existen alumnos en la convocatoria de junio, se debe mostrar un 0. Para ello se ha utilizado la función **format-number(number, format)** que convierte un número a string mediante la cadena de texto: format que puede contener los siguientes caracteres con un significado especial:

- 0 (Cualquier dígito)
- # (Cualquier dígito, pero si es cero no se mostrará)
- . (Para mostrar la posición del punto decimal: ###.##)
- , ((Para mostrar la posición de los millares: ###,###.##)
- % Para mostrar un número como porcentaje: ##%)
- ; (Separador de patrones. El primero se asignará a números positivos y el segundo a los números negativos)

NOTA: XML Copy Editor añade el prólogo de XML al documento de salida, aún siendo un documento HTML5. Habría que eliminarlo del documento de salida.

5.2. Instrucciones para trabajar con más de una hoja de estilos

<xsl:include href="URI">: Permite incluir el contenido de una hoja XSLT dentro de otra mediante su URI o ubicación. Debe aparecer como un elemento de primer nivel: como hija directa de stylesheet. La hoja incluida tiene la misma prioridad que la hoja que la incluye.

<xsl:import href="URI"/>: Permite importar los contenidos de una hoja de estilos XSLT en otra hoja de estilos XSLT.
Debe aparecer como un elemento de primer nivel: como hija directa de stylesheet y al inicio.
Se diferencia de **<xsl:include href="URI">** en que los contenidos importados tinen menor prioridad que los de la hoja XSLT que los importa.

Por ejemplo, para crear documentos HTML5, podemos crear una hoja de transformación XSLT **html5.xsl** con el marcado de inicio y de fin de un documento HTML5 e incluirlo en todas las hojas XSLT que tengan por salida HTML5:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:include href="html5.xsl"/>
  <xsl:template match="/">
    <xsl:value-of select="$html5_start" disable-output-escaping="yes"/>
    <h1>Ejemplo UD05</h1>
    <xsl:value-of select="$html5_end" disable-output-escaping="yes"/>
  </xsl:template>
</xsl:stylesheet>
```

Las variables **$html5_start** y **$html5_end** están definidas en **html5.xsl**

5.3. XSLT y espacios de nombres

Para transformar mediante XSLT documentos XML que cuentan con espacios de nombres, tendremos que:

1. Añadir el espacio de nombres al elemento raíz de la hoja XSLT: *stylesheet*.
2. Utilizar el prefijo en las expresiones XPath que le correspondan en la hoja XSLT
3. Añadir el atributo ***exclude-result-prefixes*** al elemento raíz de la hoja XSLT: *stylesheet* con el valor de los prefijos que no queremos que se añadan en el documento de salida.

Por ejemplo, para el documento **empleados-ns-b.xml** usaremos la hoja de estilos **empleados-html.xsl** y **html5.xsl**

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3   xmlns:e="http://www.teleco-galicia.com/empleados"
4   xmlns:p="http://www.teleco-galicia.com/puestos" version="1.0"
5  exclude-result-prefixes="e p">
6    <xsl:include href="html5.xsl"/>
7    <xsl:template match="/">
8     <xsl:value-of select="$html5_start" disable-output-escaping="yes"/>
9     <h1>Empleados</h1>
10    <!-- vamos a suponer que existen empleados -->
11    <ul>
12     <xsl:apply-templates select="//e:nombre"/>
13    </ul>
14    <xsl:value-of select="$html5_end" disable-output-escaping="yes"/>
15   </xsl:template>
16
17   <xsl:template match="e:nombre">
18    <li>
19     <xsl:value-of select="."/>
20     <xsl:if test="count(./..//p:nombre)&gt;0">
21      <ol>
22       <xsl:apply-templates select="./..//p:nombre"/>
23      </ol>
24     </xsl:if>
25    </li>
26   </xsl:template>
27
28   <xsl:template match="p:nombre">
29    <li>
```

En este caso estamos creando una lista desordenada con los nombres de los empleados y, si hay algún puesto para cada nombre de empleado, creamos anidada una lista ordenada para el nombre de los puestos de cada empleado:

```
<body>
  <h1>Empleados</h1>
  <ul>
    <li>Juan López Martínez
      <ol>
        <li>Administrador de sistemas</li>
        <li>Soporte de primer nivel</li>
      </ol>
    </li>
    <li>Ana Casal Brea
      <ol>
        <li>Project manager</li>
        <li>Consultora funcional</li>
      </ol>
    </li>
  </ul>
  </body>
</html>
```

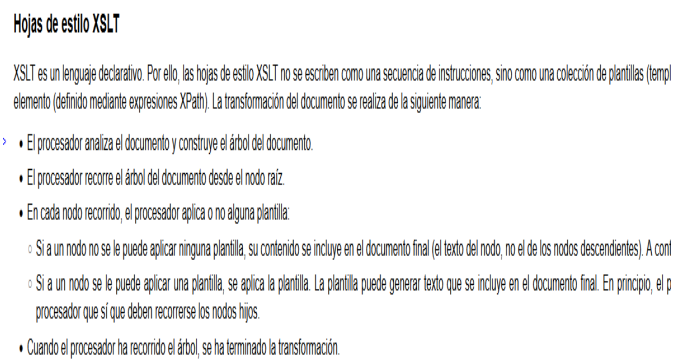
Si omitimos el atributo ***exclude-result-prefixes,*** el resultado que obtendríamos sería este otro:

```
    <body>

<h1 xmlns:e="http://www.teleco-galicia.com/empleados" xmlns:p="http://www.teleco-galicia.com
Empleados
</h1>
<ul xmlns:e="http://www.teleco-galicia.com/empleados" xmlns:p="http://www.teleco-galicia.com
  <li>Juan López Martínez
    <ol>
      <li>Administrador de sistemas</li>
      <li>Soporte de primer nivel</li>
    </ol>
  </li>
  <li>Ana Casal Brea
    <ol>
      <li>Project manager</li>
      <li>Consultora funcional</li>
    </ol>
  </li>
</ul>

</body>
</html>
```

6. Recapitulación de XSLT

Para trabajar con XSLT, en este módulo, es necesario:

1. Conocer cómo crear expresiones XPath. Tenéis un ejercicio de refuerzo en Actividad 5.2.2: XPath-Refuerzo Páxina
2. Conocer cómo funciona el procesador XSLT por defecto descrito en el recurso de McLibre de introducción a XSLT:



Hojas de estilo XSLT

XSLT es un lenguaje declarativo. Por ello, las hojas de estilo XSLT no se escriben como una secuencia de instrucciones, sino como una colección de plantillas (templ elemento (definido mediante expresiones XPath). La transformación del documento se realiza de la siguiente manera:

- El procesador analiza el documento y construye el árbol del documento.
- El procesador recorre el árbol del documento desde el nodo raíz.
- En cada nodo recorrido, el procesador aplica o no alguna plantilla:
  - Si a un nodo no se le puede aplicar ninguna plantilla, su contenido se incluye en el documento final (el texto del nodo, no el de los nodos descendientes). A cont
  - Si a un nodo se le puede aplicar una plantilla, se aplica la plantilla. La plantilla puede generar texto que se incluye en el documento final. En principio, el p procesador que sí que deben recorrerse los nodos hijos.
- Cuando el procesador ha recorrido el árbol, se ha terminado la transformación.

3. **Uso de templates**. Al menos necesitaremos una plantilla para el nodo raíz **<xsl:template match="/"> </xsl:template>** con el contenido que queremos generar una única vez. El elemento <**xsl:template>** debe ser hijo directo del elemento raíz **stylesheet**. Típicamente ahí crearemos:
   - El marcado HTML5 de inicio y fin
   - El elemento raíz de un documento XML.

   En el medio del marcado de HTML5 y fin o entre las etiquetas de apertura y cierre del elemento raíz de XML habrá que usar **<xsl:apply-templates select="" />**para indicarle al procesador de XSLT que aplique otras plantillas a los nodos del árbol del documento XML seleccionados por el **select**. El atributo **select** contendrá una expresión XPath que **seleccione los nodos a los que les queremos aplicar la (**primera**) transformación**. Dicha expresión XPath debe interpretarse **en el contexto actual** marcado por el match="/" de la plantilla.

   Por ejemplo:
   - A los alumnos de la convocatoria de junio: <xsl:apply-templates select="notas/alumno[@convocatoria='Junio']">
   - A las categorías padre: <xsl:apply-templates select="/hipermercado/categorias/categoria[not(@padreCategoriaID)]">
   - A las categorías hija: <xsl:apply-templates select="/hipermercado/categorias/categoria[@padreCategoriaID]">

   Deberemos por lo menos crear **una segunda plantilla**, para esos nodos que requieren una transformación, indicando con el atributo match la expresión XPath a la que se aplicará. La expresión XPath podría ser :
   - simplemente el nombre del nodo: <xsl:template match="alumno">
   - podrían añadirse la ruta desde la raíz: <xsl:template match="/notas/alumno">
   - o un nombre de nodo con un predicado: <xsl:template match="alumno[@convocatoria='Junio']">

   Dentro de esa segunda plantilla usaremos el marcado HTML o XML que deseamos generar en el documento final. Por ejemplo:
   - la creación de filas en una tabla: <tr>...</tr>
   - la creación de items de listas html: <li> ...
   - la creación de etiquetas video, etc.: <video>... </video>

   Podrían añadirse más elementos `<xsl:template>` siguiendo la misma mecánica:

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

 <xsl:template match="/">
  <html>
   <head>
    <title>Título</title>
   </head>
   <body>
    <xsl:apply-templates select="expresion_XPath_que_devuelve_nodos_a_los que se aplicará_template2"/>
   </body>
  </html>
 </xsl:template>

 <xsl:template match="expresion_XPath_template_2">
  <p>
   <xsl:apply-templates select="expresion_XPath_que_devuelve_nodos_a_los que se aplicará_template_n"/>
  </p>
 </xsl:template>

 <xsl:template match="expresion_XPath_template_n">
  <xsl:value-of select="expresion_XPath_desde_contexto_actual"/>
 </xsl:template>

</xsl:stylesheet>
```

4. **Importancia de conocer el contexto actual.** Para obtener los datos del documento XML original usaremos **<xsl:value-of select="" >** donde ***select*** espera una expresión XPath que se evaluará desde **el contexto actual.**

   - El contexto actual dentro de una plantilla (**template**), se define por el atributo ***match:*** **<xsl:template match="alumno">**Las rutas XPath parten ya del elemento alumno.
   - El contexto actual dentro de un **for-each**, se define por el atributo ***select:*** ***<xsl:for-each select="//categoria[@padreCategoriaID=$id]">.*** Las rutas XPath parten de los nodos categoria que cumplan la condición incluida en el predicado.
   - El contexto actual dentro de un **predicado** lo define el último elemento que hay justo antes de comenzar el corchete. **(<xsl:value-of select="count(//producto[categoria=$subCatID])"/>)** Dentro del predicado (dentro de los corchetes), las rutas XPath parten del elemento **producto**

5. **Instrucciones de control**: De forma paralela a casi cualquier lenguaje de programación os podéis encontrar con las instrucciones de control <xsl:if>, <xsl:for-each> y <xsl:choose>. Gracias a ellas podemos personalizar el contenido de la transformación en función de una determinada condición o condiciones y/o para un determinado conjunto de nodos.

6. Para **ordenar** los elementos usamos **<xsl:sort select="expresión_XPath">.** Debe situarse o bien dentro de **<xsl:apply-templates>** o bien dentro del **<xsl:for-each> y aparecer antes** que cualquier otra instrucción o marcado. El atributo **select** indica la ruta XPath del elemento o atributo por el que se ordena que se debe interpretar dependiendo del **contexto actual** marcado por el ***match*** **de apply-templates o el *select* del for-each.**

7. ¿<xsl:template> o <xsl:for-each>?
La mayor parte de las veces, el uso de <xsl:template> y <xsl:for-each> es intercambiable. Si es así, se prefiere el uso de <xsl:template> porque encapsula la transformación, es reutilizable, más mantenible y el diseño es más modular. La imagen anterior podría cambiarse por:

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

    <xsl:for-each select="expresion_XPath_que_devuelve_nodos_a_los que se aplicaría_template2">
     <p>


     <xsl:for-each select="expresion_XPath_que_devuelve_nodos_a_los que se aplicaría_template_n">
      <xsl:value-of select="expresion_XPath_desde_contexto_actual"/>
     </xsl:for-each>


     </p>
    </xsl:for-each>


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

A veces, como en el caso del hipermercado.xml y las categorías, necesitaremos comparar valores de diferentes contextos (el identificador de categoría padre con el atributo @padreCategoriaID de una categoría hija) y esto no es posible solo mediante expresiones XPath porque desde un contexto interior no tenemos acceso al otro contexto. En esos casos, nos ayudamos de las instrucciones <xsl:variable> para almacenar los valores del contexto exterior y usar el valor almacendado en el contexto interior. **Además, con lo que hemos visto en el módulo, esto solo sería posible utilizando la instrucción <xsl:for-each> al menos en el contexto más interior.**

Las variables también nos pueden ayudar a crear un marcado de salida, como las variables $html5_start y $html5_end**.**