I.E.S. DOMINGO PÉREZ MINIK

CICLO FORMATIVO DE GRADO SUPERIOR:

"ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED (L.O.E.)"

MÓDULO:

Lenguajes de marcas y sistemas de gestión de información

(4 horas semanales)

Índice

TEMA 8.- XSLT.

1.- Aplicando estilos al XML.

Ya conocemos el lenguaje CSS (Cascade StyleSheets). Podríamos aplicar ficheros .css a nuestros documentos XML para darles una presentación adecuada en un navegador web. Sin embargo esto no se utiliza mucho porque presenta bastantes desventajas. La más importante es que nuestros ficheros XML están formados por etiquetas propias, y aunque podamos dar una representación a esas etiquetas no podremos crear estructuras complejas como tablas, listas, etc. Por este motivo se creó un lenguaje mucho más potente y adaptado a XML que puede realizar tanto **transformaciones en el documento**, como **aplicación de estilos**. Este lenguaje se llama XSL y lo veremos en este tema.

Ventajas de aplicar CSS directamente a los documentos XML:

- Fácil de aprender y utilizar. El lenguaje CSS ya lo conoces.
- No requiere la creación de una página HTML para visualizar código XML.
- Consume poca memoria y tiempo de proceso, pues no construye una representación en árbol del documento.
- Muestra el documento según se va procesando.

Desventajas de aplicar CSS directamente a los documentos XML:

- Utiliza una sintaxis diferente a la del XML.
- Sólo sirve para visualizar documentos en un navegador.
- No es muy flexible:
    - No permite realizar manipulaciones sobre el documento, tales como añadir y borrar elementos, realizar ordenaciones, etc.

- Sólo permite acceder al contenido de los elementos, no a los atributos, no permite instrucciones de proceso, etc.

Para usar una hoja de estilo CSS para presentar el contenido de un documento XML hay que añadir la siguiente línea en el prólogo:

Agregar código CSS al documento XML:
<?xml-stylesheet type="text/css" href="fichero.css" ?>

PRÁCTICA 01.- Vamos a aplicar estilos CSS al documento alumnos.xml utilizado en el tema anterior.
Crea el fichero estilos.css y engánchalo a alumnos.xml. Dentro de este fichero:
Crea la clase nombre que ponga la letra roja de tamaño 20 y el fondo rosa.
Crea la clase apellidos que ponga la letra verde de tamaño 16.
Abre el fichero alumnos.xml en un navegador web. Observa cómo se aplican los estilos a las etiquetas xml con el mismo nombre que la clase CSS.

Enganchar un estilo CSS a una etiqueta XML:
- En CSS creamos una regla con selector de clase.
- En XML la etiqueta tiene que tener el mismo nombre que la clase CSS.

PRÁCTICA 02.- Con el mismo fichero del ejercicio anterior queremos mostrar sólo los nombres y los apellidos ocultando todo lo referente a las notas y a la etiqueta repetidor.
En el fichero estilos.css agrega dos clases llamadas repetidor y notas (igual que las etiquetas que queremos ocultar). Dentro de ellas usa diplay:none.
Abre el fichero alumnos.xml en un navegador web. Observa cómo se ocultan las etiquetas pertinentes.

Como puedes ver en el resultado esto no nos resuelve mucho, puesto que no podemos agregar nuevas líneas para separar los alumnos, ni hacer tablas ni crear documentos con buena apariencia. Para eso necesitaremos **transformar el documento XML**, y ese es el principal cometido del lenguaje **XSL.**

XSL → Extensible StyleSheet Language
Es un lenguaje que recoje como entrada un documento XML y aplicando una transformación produce otro documento de salida (que podrá ser XML, PDF, HTML o cualquier otro formato).

La idea principal de este lenguaje es partir de una información (almacenada en un fichero xml) y aplicándole distintas hojas de estilo XSL producir diferentes ficheros de salida, cada uno con una transformación diferente. Así podremos producir tablas, páginas web, ficheros RDF, ficheros PDF o cualquier otro formato conocido con la misma información.

En realidad el estándar XSL está compuesto de dos subestándares:

XSL está compuesto por:

- **XSLT**: (XSL Transformations) lenguaje para transformar documentos XML en otro formato (otro XML, HTML, DHTML, texto plano, PDF, RTF, Word, etc.)
- **XSL-FO:** (XSL Formatting Objects) especificación que trata cómo deben ser los objetos de formato para convertir XML a formatos binarios (PDF, Word, imágenes, etc.). Todavía no ha alcanzado el estado de "recomendado por la W3C".

Y ambos estándares se apoyan en **XPath** para seleccionar partes de un documento XML.

Nosotros nos centraremos en XSLT, un lenguaje muy potente que nos permitirá la manipulación y formateo de los documentos XML en otros documentos. Este estándar aplica transformaciones siempre basadas en el **árbol sintáctico** de los datos, de forma que partiendo del **árbol del documento de entrada** generará otro **árbol de salida.**

Ventajas de uso de XSLT:

- La salida no tiene por qué ser HTML para visualización en un navegador, sino que puede estar en muchos formatos.
- Permite manipular de muy diversas maneras un documento XML: reordenar elementos, filtrar, añadir, borrar, etc.
- Permite acceder a todo el documento XML, no sólo al contenido de los elementos. Por ejemplo permite acceso a los atributos.
- **XSLT es un lenguaje XML**, por lo que no hay que aprender nada especial acerca de su sintaxis.

Desventajas del uso de XSLT:

- Su utilización es más compleja. (Hasta que se aprende el estándar)
- Consume cierta memoria y capacidad de proceso, pues se construye un árbol con el contenido del documento. (Poco significativa en los sistemas modernos).

Cómo usar XSLT:

- Visualizar directamente en un navegador el documento XML que tiene asociada una hoja XSLT. El navegador debe tener incorporado un procesador XSLT.
- Ejecutar un procesador XSLT independientemente. Se le pasan las entradas necesarias (fichero origen y hoja XSLT a utilizar) y genera la salida en un fichero nuevo.
- Realizar las transformaciones dentro de un programa en el servidor y enviar a los clientes sólo el resultado de la transformación. Dependiendo

del tipo de cliente podremos enviar una web, una página adaptada para un dispositivo móvil, etc.

2.- Primeros pasos con XSL.

2.1.- Creación de la hoja de estilos XSL:

Crearemos un fichero con extensión .xsl o bien .xslt. Dentro de él declararemos el espacio de nombres de este estándar asignándole el prefijo que queramos (típicamente se usa xsl como prefijo):

```
<!—Ejemplo de Fichero XSL -->

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

        …AQUÍ VA TODO EL CONTENIDO DE NUESTRO FICHERO XSLT…
        1.- ELEMENTOS DE PRIMER NIVEL
        2.- PLANTILLAS

</xsl:stylesheet>
```

El contenido de nuestra hoja de estilos se estructura en dos partes principales:

1.- Los **elementos de primer nivel**: indicarán cuestiones generales, como por ejemplo, el tipo de documento que queremos crear.

2.- Las **plantillas** (en inglés templates) que indicarán las reglas de procesamiento que deben aplicarse a las diferentes etiquetas de nuestro fichero de entrada.

2.2.- Asociar la hoja de estilos XSL al documento XML:

Solamente hay que añadir en el prólogo del documento XML (habitualmente la segunda línea) el enlace a la hoja XSL:

```
<?xml-stylesheet type="text/xsl" href="mifichero.xsl" ?>
```

2.3.- Un primer ejemplo:

Partiremos del fichero alumnos.xml que hemos usado en las prácticas del tema anterior. Recuerda que tenemos la etiqueta instituto, dentro de ella la etiqueta curso y dentro de ella los diferentes alumnos de cada curso. Queremos obtener una página web que muestre una lista con los nombres, apellidos y ciales de los alumnos. Observa la hoja de estilos XSLT aplicada:

```
<?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"/> <!-- elemento de primer nivel -->
```

```
<!-- plantilla para procesar el nodo raíz (/) -->
<xsl:template match="/">
<!-- Todo lo que no lleve el prefijo xsl será copiado en la salida-->
  <html>
    <head>
        <title>Alumnos del instituto</title>
    </head>
    <body>
        <h1>LISTA DE ALUMNOS:</h1>
        <ul>
<!-- Todo lo que tenga el prefijo xsl serán reglas para transformar el
documento xml de entrada mediante plantillas-->
        <xsl:apply-templates select="/instituto/curso"/>
<!-- indicamos que se apliquen las plantillas a los nodos curso y
descendientes-->
        </ul>
    </body>
</html>
</xsl:template> <!-- fin de la primera plantilla -->


<xsl:template match="alumno"><!-- En esta plantilla transformamos la
etiqueta alumno (está dentro de curso)-->
  <li><!-- cada alumno lo convertimos en  un elemento de lista-->
    Nombre: <b>
     <xsl:value-of select="./nombre"/><!-- salida del texto que está
dentro de nombre en negrita-->
    </b>
    <!-- salida del texto que está dentro de apellidos-->
    <br/>Apellidos: <xsl:value-of select="./apellidos"/>
    <!-- salida del atributo cial-->
    <br/>CIAL: <xsl:value-of select="@cial"/>
    <hr  noshade="noshade"/>
  </li>
</xsl:template> <!-- fin de la segunda plantilla-->
</xsl:stylesheet>
```

PRÁCTICA 03.- Observa detenidamente el código de la hoja XSL anterior. Después enlázala al fichero alumnos.xml. Abre alumnos.xml en un navegador compatible con XSL (Explorer, Safari, Opera, Firefox, …).

Como habrás observado el aplicar una hoja XSL consiste en crear una serie de plantillas que operan sobre el árbol sintáctico del documento XML.

Cada plantilla se define con **template** y se encarga de un conjunto de nodos definido en el atributo **match**, que contiene una expresión XPath.

Para procesar otras plantillas que operan dentro del conjunto de nodos definido en match y sus descendientes se utiliza **apply-template.**

Por último hemos usado **value-of y select** para mostrar los elementos.

PRÁCTICA 04.- Modifica la hoja XSL anterior para que por cada alumno veamos su nombre, apellidos, cial y también el curso. XPath es la clave.

PRÁCTICA 05.- Modifica la hoja XSL anterior para que sólo aparezcan en la lista los alumnos de 2º de ASIR.

3.- Elementos de primer nivel.

Los elementos de primer nivel son aquellos que sólo pueden aparecer como hijos del elemento raíz <xsl:stylesheet…>. Se encargan de cuestiones que afectarán a todo el proceso de transformación del fichero de entrada en el fichero de salida.

Típicamente los escribiremos antes de colocar las plantillas.

3.1.- <xsl:output…>:

Especifica el formato del fichero de salida. La sintaxis completa es la siguiente:

```
<xsl:output method=”xml|html|text” version=”version”
        omit-xml-declaration=”yes|no”  indent=”yes|no”
        encoding=”codificación”   doctype-system=”doctype”
        doctype-public=”doctype” />
```

- El atributo **method** puede contener xml (si queremos producir un fichero escrito en xml), html (si queremos crear una página web) o text (para producir un fichero de texto – esto se usa para el resto de los lenguajes de marcado también).

Los demás atributos son opcionales:

- El atributo **version** identifica la versión de la tecnología empleada en el fichero de salida.
- El atributo **omit-xml-declaration** tiene por defecto el valor “no”. Cuando el fichero de salida es xml por defecto se escribe la primera línea con la instrucción de procesamiento habitual <?xml version="1.0" encoding="…"?>. Con este atributo, si lo ponemos con valor “yes” omitimos la creación de esta línea.
- El atributo **indent** con valor “yes” provoca un fichero de salida con identaciones. Algunos procesadores XSLT no lo soportan.
- El atributo **encoding** permite especificar la codificación que aparece en <?xml version=”1.0” encoding=”…”> Valores típicos son UTF-8, ISO-8859-1.
- El atributo **doctype-system** permite agregar una etiqueta de tipo <!DOCTYPE> al documento de salida. Le daremos el valor de la URI que contiene el DTD de ese doctype, según el estándar que estemos produciendo. Sólo usaremos estos doctypes (los de tipo system) si son accesibles en local o en nuestra red.
- El atributo **doctype-public** es similar al anterior pero para un doctype público (no está en local sino en internet). Indicaremos el identificador público. Por ejemplo si lo que queremos es crear un documento XHTML 1.0 estricto haríamos:

```
<xsl:output method="xml" encoding="ISO-8859-1" indent="yes"
doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"/>
```

Y generaríamos un fichero de salida parecido a este (con la cabecera de XHTML):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

PRÁCTICA 06.- Las transformaciones de XSLT pueden ser muy potentes. Vamos a crear ciales.xml que tendrá la siguiente forma (para cada alumno de alumnos.xml).

Alumno en alumnos.xml (fichero de entrada):

```
<alumno cial="A92R003">
        <nombre>Luis Fernando</nombre>
        <apellidos>Jiménez Pérez</apellidos>
        <repetidor>False</repetidor>
         <notas>
            <asignatura>Seguridad Informática</asignatura>
            <nota>3</nota>
            … resto de asignaturas y notas …
         </notas>
</alumno>
```

Conversión de ese alumno en ciales.xml (fichero de salida):

```
<ciales> <!—es el nuevo elemento raíz-->
… alumnos anteriores …
  <alumno>
    <cial>A92R003</cial>
    <datos>Luis Fernando - Jiménez Pérez</datos>
    <curso>2º ASIR</curso>
  </alumno>
   …alumnos posteriores …
</ciales>
```

Observa que antes el cial era un atributo de la etiqueta alumno y ahora queremos que sea una etiqueta independiente en el nuevo fichero. Aún no conoces el estándar XSLT, pero intenta comprender el siguiente código:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml"/><!--Crearemos un fichero XML-->
  <xsl:template match="/"><!--Primera plantilla: se aplica al elemento raíz-->
      <ciales> <!--nuestro nuevo elemento raíz es la etiqueta ciales -->
        <xsl:apply-templates select="/instituto/curso/alumno"/><!—seguimos procesando-->
      </ciales>
   </xsl:template> <!-- fin de la primera plantilla -->

    <xsl:template match="alumno"><!—2ª plantilla: se aplica a la etiqueta alumno-->
       <alumno>
            <cial><xsl:value-of select="@cial"/></cial>
            <datos> <xsl:value-of select="nombre"/> <xsl:text> -   </xsl:text>
                      <xsl:value-of select="apellidos"/></datos>
            <curso><xsl:value-of select="../@nombre"/></curso>
             </alumno>
   </xsl:template> <!-- fin de la segunda plantilla-->
</xsl:stylesheet>
```

Quita toda referencia a XSLT en el fichero alumnos.xml. Copia este código en practica06.xsl. Utiliza la opción del menú "Transform a document with this XSLT" indicando que el fichero de entrada es alumnos.xml y que el de salida se llamará ciales.xml.

Abre ciales.xml para que veas el documento generado. Pruébalo también en un navegador web.

PRÁCTICA 07.- Tú que eres un experto en Xpath utilízalo para que sólo aparezcan en el fichero ciales07.xml aquellos alumnos que tengan alguno de sus apellidos "Pérez".

PRÁCTICA 08.- Deseamos generar un fichero para invitar a los alumnos Pérez a un evento. La forma del fichero será la siguiente (modifica la práctica anterior para crearlo):

```
<?xml version="1.0" encoding="utf-8" ?>
- <cartas>
  - <invitacion>
      <destinatario>María - Álvarez Pérez</destinatario>
      <evento>1ª Congregación de los Pérez</evento>
    </invitacion>
  - <invitacion>
      <destinatario>Iván - Rodríguez Pérez</destinatario>
      <evento>1ª Congregación de los Pérez</evento>
    </invitacion>
  - <invitacion>
      <destinatario>Luis Fernando - Jiménez Pérez</destinatario>
      <evento>1ª Congregación de los Pérez</evento>
    </invitacion>
  </cartas>
```

3.2.- <xsl:preserve-space…>:

Se utiliza para indicar en qué elementos se deben preservar los espacios en blanco existentes:

<xsl:preserve-space elements=”lista de elementos separados por espacio”/>

El atributo elements es obligatorio. Ejemplo de uso:

<xsl:preserve-space elements=”apellidos asignatura”/>

Al procesar los elementos apellidos y asignatura se preservarán los espacios en blanco.

3.3.- <xsl:strip-space…>:

Es posible que en algunas prácticas de este tema observes cómo el resultado de una transformación no quede como lo habías deseado. Muchas veces esto se debe a que el documento de entrada se interpreta exactamente (con sus saltos de línea, espacios, etc) y se crea en el árbol de salida un conjunto de nodos que lo que contienen son espacios o caracteres no imprimibles.

La instrucción strip-space elimina los considerados espacios no significativos, que son aquellos que ocurren entre distintos elementos (NO dentro del contenido de texto de los elementos).

<xsl:strip-space elements=”lista de elementos separados por espacio”/>

El atributo elements se usa de la misma forma que en preserve-space.

<xsl:strip-space elements=”alumno”/>
Eliminará espacios en blanco entre nombre y apellidos, por ejemplo, o entre apellidos y notas.

3.4.- Otros elementos de primer nivel:

Existen otros elementos de primer nivel en el estándar XSLT. Dado que tienen funciones no básicas simplemente los citaremos y si necesitamos usarlos en alguna práctica les prestaremos más atención:

<xsl:import> → sirve para crear modularidad, es decir, para utilizar otras hojas XSLT en nuestro documento. El funcionamiento es muy similar a los include de C++ o a la etiqueta <LINK> en html: permite IMPORTAR otra hoja XSL a la nuestra.

Utilizando esta instrucción podemos importar cualquier hoja de estilos. Si en nuestro fichero se define una regla (plantilla) para un nodo llamado N y en el fichero importado también hay una regla para un nodo N entonces la que tendrá prioridad es la de nuestra hoja de estilos. Esto puede modificarse con la instrucción <apply-templates>.

Uso: <**xsl:import href**=”./estilos/fichero02.xsl”/>

<xsl:key> → sirve para crear un índice de información en el fichero de entrada, de forma que el procesador XSLT tenga un acceso más eficiente al documento, en lugar de buscar en todo el fichero. Sólo se usa con ficheros de entrada extremadamente grandes. La mayoría de los procesadores actuales de XSLT no lo soportan todavía.

Imagina nuestro fichero de alumnos. Constantemente estamos haciendo transformaciones que implican buscar un alumno concreto. Podríamos usar su atributo cial como clave para crear un índice.

<**xsl:key name=**”indiceCial” **match**=”/instituto/curso/alumno” **use**=”@cial”/>

Con este ejemplo se crearía un índice llamado índiceCial que guardará el cial de cada alumno. A partir de ese momento podríamos usar la función **key()** para localizar a los alumnos.

<xsl:decimal-format> → sirve para dos cosas:
Si utilizamos el atributo name crearemos un formato de números propio, que luego podremos usarlo en la función de XPath format-number().
Si no utilizamos el atributo name servirá para crear un formato de números que será el que se aplique a todos los números del fichero de salid.
Tiene otros atributos que se usan para hojas de estilos en lenguajes distintos al inglés (especialmente alfabetos distintos).

4.- Las plantillas.

Las plantillas (en inglés **templates**) serán las reglas de transformación que debemos aplicar sobre el árbol sintáctico del documento de entrada. Deben especificar por una parte **el conjunto de nodos sobre el que operan** y por otra parte **el formato de salida para cada nodo de ese conjunto.**

Funcionan de forma recursiva, es decir, se aplica una primera plantilla sobre una expresión XPath (**expresión que debe retornar un conjunto de nodos**), y sobre este conjunto de nodos se irá analizando si hay definida alguna otra plantilla (que a su vez podrá provocar otros conjuntos de nodos), etc.

4.1.- <xsl:template…>:

Crea una nueva plantilla.

Hay dos tipos básicos de plantillas: la plantillas generales (usan el atributo **match**) y las plantillas con nombre (usan el atributo **name**). Uno de estos dos atributos es obligatorio.

```
<xsl:template match=”expresión XPath” name=”nombre”
 mode=”modo”   priority=”prioridad” xml:space=”default|preserve”/>
```

- Atributo **match**: contiene una expresión XPath que debe retornar o bien un conjunto de nodos o bien un nodo individual. La transformación se realizará sobre cada uno de los nodos que devuelve esta expresión.

Ejemplo: <xsl:template match=”/instituto/curso”> → aplicará una transformación a cada elemento “curso” del fichero xml de entrada.

- Atributo **name**: sirve para dar un nombre a una plantilla que será invocada con la instrucción <xsl:call-template name=”nombre”/>. Este atributo es alternativo a match. Este tipo de plantillas soporta el paso de información (parámetros) en la llamada utilizando la instrucción <xsl:with-param>.

El resto de los atributos son opcionales, y los explicaremos cuando los necesitemos.

4.2.- <xsl:apply-templates…>:

Instrucción para indicar al procesador XSLT que continúe procesando plantillas para los elementos seleccionados. Se coloca dentro de un <xsl-template> para analizar su funcionamiento mira el siguiente código:

```
<xsl:template match="/instituto/curso">
  <ol>
    <xsl:apply-templates select="alumno"/>
  </ol>
</xsl:template> <!-- fin de la plantilla aplicada a curso-->

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

```
  <li>
    <xsl:value-of select="@cial"/>
  </li>
</xsl:template> <!-- fin de la plantilla aplicada a curso-->
```

En este ejemplo tenemos una primera plantilla que se ejecutará para cada curso. El atributo match define que se estén seleccionando los subárboles de cada curso en el árbol sintáctico del fichero de entrada. La primera plantilla crea para cada subárbol una lista ordenada con <ol> y .

Sin embargo dentro de <ol> tenemos apply-templates indicando que deben procesarse las plantillas definidas para los nodos alumno hijos del nodo contexto. Es justo nuestra segunda plantilla la que estamos invocando. Por lo tanto dentro de la lista <ol> de cada curso ejecutaremos la segunda plantilla, que creará una lista de todos los ciales de los alumnos de ese curso.

Recursividad:
Es la idea de funcionamiento de XSLT. Se procesa un conjunto de nodos y por cada elemento del mismo se genera una transformación.

Podríamos llamar a <xsl:apply-templates> sin usar el atributo select, y en este caso se invocarían las plantillas definidas para todos los hijos del nodo contexto de forma automática. Observa el ejemplo:

```
<xsl:template match="/instituto/curso/alumno">
  <p align=”center”>
    <xsl:apply-templates>
  </p>
</xsl:template>
```

En este caso nos encontramos con una plantilla que se aplica a cada nodo alumno. Dentro se crea un párrafo y se llama a apply-templates. Como no hay nada definido en esta llamada se buscaría en el fichero si hay plantillas creadas para todos los hijos de alumno, esto es, para nombre, para apellidos y para notas.

Luego la sintaxis es:

<xsl:apply-templates select=”expresión XPath” **mode**=”modo” />

El atributo mode sirve para enlazar con el atributo del mismo nombre en <xsl:template>. Tiene que ver con la necesidad de reutilizar contenido en distintas partes del documento. Los vemos en el siguiente apartado.

PRÁCTICA 09.- Crea practica09.xsl que utilizando el fichero de entrada msn_data.xml cree el fichero practica09.xml que contenga la siguiente información:

```
<?xml version="1.0" encoding="utf-8" ?>
- <noticias>
    <titulo>Vuelo 447 Rio-Paris, imágenes de los restos del Airbus en el fondo del mar</titulo>
    <titulo>Gestiona todo tu correo a través de Hotmail - Artículo - Canal Windows Live - MSN</titulo>
    <titulo>El vanguardismo llega a los nuevos museos - Listas - MSN Viajes</titulo>
    <titulo>La inyección de una espuma plástica no frena el escape de agua radiactiva de Fukushima-1 - MSN Noticias</titulo>
    <titulo>LAS MIL CARAS DE BELEN ESTEBAN ..NO TE PIERDAS ESTE POST ¿ CUANDO APRENDIO A GESTICULAR ASI?</titulo>
    <titulo>PASOS A SEGUIR PARA UNA ALIMENTACION SANA Y ERRORES QUE SOLEMOS COMETER AL HACER DIETA</titulo>
    <titulo>EL NIÑO THAILANDÉS Y LA SERPIENTE GIGANTE. UNA FASCINANTE AMISTAD. VÍDEO E IMÁGENES</titulo>
    <titulo>Satélite de la NASA medirá la salinidad de la superficie de los océanos</titulo>
    <titulo>PASOS A SEGUIR PARA UNA ALIMENTACION SANA Y ERRORES QUE SOLEMOS COMETER AL HACER DIETA</titulo>
    <titulo>Cuando Me Enamoro Juan Luis Guerra y Enrique Igleseias ...preciosa cancion</titulo>
  </noticias>
```

PRÁCTICA 10.- Ahora crea practica10.xsl que utilizando el fichero de entrada msn_data.xml cree el fichero practica10.html que contenga la siguiente información:

Vuelo 447 Rio-Paris, imágenes de los restos del Airbus en el fondo del mar
Gestiona todo tu correo a través de Hotmail - Artículo - Canal Windows Live - MSN
El vanguardismo llega a los nuevos museos - Listas - MSN Viajes
La inyección de una espuma plástica no frena el escape de agua radiactiva de Fukushima-1 - MSN Noticias
LAS MIL CARAS DE BELEN ESTEBAN ..NO TE PIERDAS ESTE POST ¿ CUANDO APRENDIO A GESTICULAR ASI?
PASOS A SEGUIR PARA UNA ALIMENTACION SANA Y ERRORES QUE SOLEMOS COMETER AL HACER DIETA
EL NIÑO THAILANDÉS Y LA SERPIENTE GIGANTE. UNA FASCINANTE AMISTAD. VÍDEO E IMÁGENES
Satélite de la NASA medirá la salinidad de la superficie de los océanos
PASOS A SEGUIR PARA UNA ALIMENTACION SANA Y ERRORES QUE SOLEMOS COMETER AL HACER DIETA
Cuando Me Enamoro Juan Luis Guerra y Enrique Igleseias ...preciosa cancion

Nota: cada título está dentro de una tabla con width=”70%” y border=”1”. Entre tabla y tabla hay un <br/>.

4.3.- Uso del atributo mode:

Ya hemos visto cómo crear plantillas con <xsl:template> y hemos visto cómo indicarle al procesador que continúe ejecutando plantillas con <xsl:apply-templates>.

Imagina que tenemos <xsl:template match=”alumno”> y definimos en ella que estamos creando una tabla de nombres. Imagina ahora que más adelante necesitamos volver a utilizar otra plantilla que trabaja con alumnos, para por ejemplo producir una lista de ciales. No podríamos volver a escribir la instrucción <xsl:template match=”alumno”> porque confundiríamos al procesador.

Para esta necesidad de utilizar un mismo conjunto de nodos y producir distintos formatos de salida existen los modos. Observa el ejemplo:

```
<xsl:template match="/instituto/curso/ ">
  <h1>Nombres y apellidos</h1>
    <xsl:apply-templates select=”alumno”>
  <h1>Ciales</h1>
    <xsl:apply-templates select=”alumno” mode=”imprimeCial”>
</xsl:template>
```

Tenemos una plantilla que se ejecuta sobre cada curso. En ella se muestra un mensaje “Nombre y apellidos” y se llama a una plantilla que opera sobre los alumnos. Después de esto se muestra el mensaje “Ciales” e invocamos también a una plantilla que opera sobre alumnos pero le hemos puesto mode=”imprimeCial” para diferenciarla de la primera.

Para crear las dos plantillas sólo tendríamos que hacer lo siguiente:

```
<xsl:template match="alumno">
  … contenido de la plantilla que mostrará los nombres y los apellidos
</xsl:template>
<xsl:template match="alumno" mode=”imprimeCial”>
  … contenido de la plantilla que mostrará los ciales
</xsl:template>
```

Como ves el nombre que utilicemos en mode debe coincidir tanto en <xsl:template> como en <xsl:apply-templates>.

PRÁCTICA 11.- Crea practica11.xsl que genera una página web con la siguiente información:

1º.- Imprime el texto: “Lista de nombres y apellidos”.

2º.- Por cada alumno crea una tabla con sólo dos celdas: la primera con el nombre y la segunda con los apellidos.

3º.- Imprime el texto: “Lista de ciales y curso”.

4º.- Por cada alumno crea una tabla con sólo dos celdas: la primera con el cial y la segunda con el nombre del curso.

4.4.- <xsl:value-of …>:

Esta instrucción, que ya has utilizado en las prácticas anteriores, sirve para producir como salida el resultado de una expresión XPath. Esta expresión no debería mostrar un conjunto de nodos sino un valor (un texto, un atributo o una variable).

<xsl:value-of select=”expresión XPath” **disable-output-escaping**=”yes|no” />

- Atributo **select**: es obligatorio y contiene la expresión XPath que se volcará al fichero de salida.
- Atributo **disable-output-escaping**: es opcional y su valor por defecto es “no”. Es habitual que al producir ficheros xml o html ciertos caracteres como el “<” se muestren codificados como entidades (< para el ejemplo). Si ponemos este atributo con valor “yes” esto no se produce y los caracteres se escribirán en el fichero de salida tal y como están en el fichero de entrada.

4.5.- <xsl:text …>:

Esta instrucción sirve para mostrar en el fichero de salida un texto. Observa el ejemplo: <xsl:text> : </text> → escribirá en el fichero de salida un espacio en blanco, un carácter de dos puntos y otro espacio en blanco.

<xsl:text disable-output-escaping=”yes|no”> contenido </**xsl:text**>

El atributo **disable-output-escaping** funciona igual que en <xsl:value-of..>

PRÁCTICA 12.- Crea practica12.xsl que genere una página web como la siguiente (atento a las cuestiones de estilo):

Alumnos por curso:

1º ASIR		
CIAL:	Nombre:	Apellidos:
A18X111	Javier	Márquez Fernández
A27M242	Maria	Álvarez Pérez
A90R112	Iván	Rodríguez Pérez

2º ASIR		
CIAL:	Nombre:	Apellidos:
A92T861	Marcos	Domínguez
A92R003	Luis Fernando	Jiménez Pérez
A87R122	Elena	Ramírez

1º DAI		
CIAL:	Nombre:	Apellidos:
A95M600	Irina	Lugo Lugo
A92R543	Genaro	Hernández Hernández
A92R590	Miguel	Rodríguez

4.6.- <xsl:element …>:

Esta instrucción permite crear elementos en el fichero de salida, de forma que el nombre del elemento se toma como resultado de una expresión XPath.

<xsl:element name=”nombre” namespace=”espaciodenombres”
use-attribute-sets=”atributos que toma el elemento separados por un espacio en blanco” />

- El atributo **name** es obligatorio. Será el nombre de la etiqueta generada.
- El atributo **namespace** se usa sólo si queremos que la etiqueta pertenezca a un espacio de nombres completo.
- El atributo **use-attribute-sets** sirve para definir los atributos que tendrá nuestra nueva etiqueta. Si vamos a usar varios atributos los separamos por un espacio en blanco.

Observa (y prueba el resultado sobre el fichero alumnos.xml) el siguiente código:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"/>
<xsl:template match="/">
   <alumnos>
     <xsl:apply-templates select="/instituto/curso/alumno"/>
   </alumnos>
</xsl:template>
```

```
<xsl:template match="alumno">
  <xsl:element name="identificador" >
    <xsl:value-of select="@cial"/>
  </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

Como puedes comprobar generamos un fichero xml cuyo elemento raíz se llama <alumnos>. Dentro de éste elemento existirá, por cada alumno, el elemento <identificador> cuyo contenido es un texto con el cial del alumno. Hemos creado, por tanto, un **nuevo elemento**.

PRÁCTICA 13.- Crea practica13.xsl que transforme msn_data.xml en un fichero como el siguiente:

```
<?xml version="1.0" encoding="utf-8" ?>
- <reporteros>
  - <reportero>
      <nombre>chupidos</nombre>
      <noticia>Vuelo 447 Rio-Paris, imágenes de los restos del Airbus en el fondo del mar</noticia>
      <fecha>Mon, 04 Apr 2011 22:36:00 GMT</fecha>
    </reportero>
  - <reportero>
      <nombre>yuly</nombre>
      <noticia>Gestiona todo tu correo a través de Hotmail - Artículo - Canal Windows Live - MSN</noticia>
      <fecha>Mon, 04 Apr 2011 21:40:00 GMT</fecha>
    </reportero>
  - <reportero>
      <nombre>HOY</nombre>
      <noticia>El vanguardismo llega a los nuevos museos - Listas - MSN Viajes</noticia>
      <fecha>Mon, 04 Apr 2011 21:00:00 GMT</fecha>
    </reportero>
  - <reportero>
      <nombre>PAsturias</nombre>
      <noticia>La inyección de una espuma plástica no frena el escape de agua radiactiva de Fukushima-1 - MSN Noticias</noticia>
      <fecha>Mon, 04 Apr 2011 19:03:00 GMT</fecha>
    </reportero>
  - <reportero>
      <nombre>DUNALUNA</nombre>
      <noticia>LAS MIL CARAS DE BELEN ESTEBAN ..NO TE PIERDAS ESTE POST ¿ CUANDO APRENDIO A GESTICULAR ASI?</noticia>
      <fecha>Mon, 04 Apr 2011 14:46:00 GMT</fecha>
    </reportero>
```

Utiliza <xsl:element> para crear los elementos reportero, nombre, noticia y fecha.

4.7.- <xsl:attribute …>:

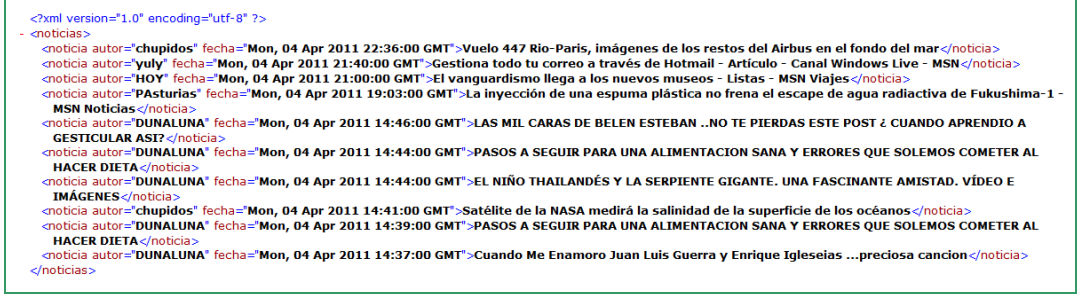
Esta instrucción permite crear asociar atributos a los elementos del fichero de salida, de forma que se pueda rellenar el atributo como resultado del proceso de los nodos.

<xsl:attribute name=”nombre o expression XPath” **namespace**=”espaciodenombres”/>

- El atributo **name** será directamente una cadena o bien el resultado de una expresión Xpath y se corresponde con el nombre del atributo creado.
- El atributo **namespace** sirve para asignar un espacio de nombres al atributo, esto es, agrega en el fichero de salid el prefijo especificado.

Esta declaración de atributos debe colocarse dentro del elemento al que deseas asignárselo.

PRÁCTICA 14.- Crea practica14.xsl que transforme msn_data.xml en un fichero como el siguiente:



```
<?xml version="1.0" encoding="utf-8" ?>
- <noticias>
    <noticia autor="chupidos" fecha="Mon, 04 Apr 2011 22:36:00 GMT">Vuelo 447 Rio-Paris, imágenes de los restos del Airbus en el fondo del mar</noticia>
    <noticia autor="yuly" fecha="Mon, 04 Apr 2011 21:40:00 GMT">Gestiona todo tu correo a través de Hotmail - Artículo - Canal Windows Live - MSN</noticia>
    <noticia autor="HOY" fecha="Mon, 04 Apr 2011 21:00:00 GMT">El vanguardismo llega a los nuevos museos - Listas - MSN Viajes</noticia>
    <noticia autor="PAsturias" fecha="Mon, 04 Apr 2011 19:03:00 GMT">La inyección de una espuma plástica no frena el escape de agua radiactiva de Fukushima-1 -
      MSN Noticias</noticia>
    <noticia autor="DUNALUNA" fecha="Mon, 04 Apr 2011 14:46:00 GMT">LAS MIL CARAS DE BELEN ESTEBAN ..NO TE PIERDAS ESTE POST ¿ CUANDO APRENDIO A
      GESTICULAR ASI?</noticia>
    <noticia autor="DUNALUNA" fecha="Mon, 04 Apr 2011 14:44:00 GMT">PASOS A SEGUIR PARA UNA ALIMENTACION SANA Y ERRORES QUE SOLEMOS COMETER AL
      HACER DIETA</noticia>
    <noticia autor="DUNALUNA" fecha="Mon, 04 Apr 2011 14:44:00 GMT">EL NIÑO THAILANDÉS Y LA SERPIENTE GIGANTE. UNA FASCINANTE AMISTAD. VÍDEO E
      IMÁGENES</noticia>
    <noticia autor="chupidos" fecha="Mon, 04 Apr 2011 14:41:00 GMT">Satélite de la NASA medirá la salinidad de la superficie de los océanos</noticia>
    <noticia autor="DUNALUNA" fecha="Mon, 04 Apr 2011 14:39:00 GMT">PASOS A SEGUIR PARA UNA ALIMENTACION SANA Y ERRORES QUE SOLEMOS COMETER AL
      HACER DIETA</noticia>
    <noticia autor="DUNALUNA" fecha="Mon, 04 Apr 2011 14:37:00 GMT">Cuando Me Enamoro Juan Luis Guerra y Enrique Igleseias ...preciosa cancion</noticia>
  </noticias>
```

4.8.- <xsl:number …>:

Esta instrucción sirve para crear numeraciones en el fichero de salida. Por ejemplo si tenemos información de un libro compuesto por capítulos nos puede interesar que antes del comienzo de cada capítulo aparezca una numeración del tipo 1.1, 1.2, 1.3, etc.

También existe otro uso de esta instrucción y es dar formato a un número generado por una expresión XPath.

En el primer uso (las enumeraciones) el número asignado a cada nodo dependerá de su posición en el conjunto de nodos devuelto por XPath (comenzando, como sabes, en 1).

Tiene muchos atributos con valores posibles para controlar la numeración creada. Veamos la sintaxis de los más importantes:

```
<xsl:number value=”valor” count=”nodos” level=”single|multiple|any”
   format=”1|A|a|I|i“ />
```

- El atributo **value**: se usa para formatear un valor contreto devuelto por una expresión XPath que debe devolver un único valor. No lo usaremos si estamos creando enumeraciones ya que sirve para formatear números concretos.

  Ejemplo: <xsl:number value=”sum(nota)” …/> → da un formato al número resultante de sumar las notas de un alumno.

- El atributo **count**: al contrario que el atributo value éste sirve para crear enumeraciones. Contendrá una expresión XPath que devuelve un conjunto de nodos y devolverá el número dentro de este conjunto para cada nodo.

  Ejemplo: <xsl:number count=”asignatura|nota” …/> → asignará un número entero a cada conjunto de asignatura y nota, contando juntos la ocurrencia de estos dos elementos.

- El atributo **level**: controla si estamos creando enumeraciones de un único nivel (1,2,3…) o de varios niveles (1.1, 1.2, 1.3).
  - Si a level le damos el valor "**single"** sólo se contarán los nodos que estén en el mismo nivel del subárbol especificado en el atributo count. La secuencia será sencilla: 1,2,3…
  - Si a level le damos el valor "**multiple**" se entenderá que estamos enumerando los nodos que están al mismo nivel del árbol y también los antecesores, creando múltiples niveles de numeración. Imagina que la expresión que usamos en count devuelve un subárbol de dos niveles se usará 1.1, 1.2, 1.3, 2.1, 2.2, … Si en count hay un subárbol de tres niveles se usarán números de tres niveles 1.1.1, 1.1.2, …
  - Si a level le damos el valor "**any**" se creará una secuencia simple de 1,2,3, … pero que incluirá no sólo los nodos del mismo nivel sino cualquier antecesor que esté en el subárbol especificado en count.
- El atributo **format**: funciona igual que las listas ordenadas en html. 1 para números, A y a para letras (mayúsculas y minúsculas respectivamente), I e i para números romanos (mayúsculas y minúsculas respectivamente). La única novedad es que cuando usas números decimales puedes poner patrones como "001", que añadirá dos ceros antes del número pertinente. También puedes agregar modelos de enumeración: 1) (agrega un paréntesis tras el número), 1.- (agrega un punto y un guión tras el número), …

Para comprender mejor los diferentes atributos de esta instrucción hagamos algunas prácticas con nuestro fichero alumnos.xml. Modifica tus códigos hasta conseguir los resultados solicitados:

PRÁCTICA 15.- Crea practica15.xsl que transforme alumnos.xml en una página web como la siguiente:

Lista de cursos disponibles:

a) 1º ASIR
b) 2º ASIR
c) 1º DAI

PRÁCTICA 16.- Crea practica16.xsl que transforme alumnos.xml en una página web como la siguiente:

Lista de alumnos matriculados:

a) 1º ASIR
1) Javier
2) Maria
3) Iván
b) 2º ASIR
1) Marcos
2) Luis Fernando
3) Elena
c) 1º DAI
1) Irina
2) Genaro
3) Miguel

PRÁCTICA 17.- Crea practica17.xsl que transforme alumnos.xml en una página web como la siguiente:

Lista de alumnos con alguna materia suspensa:

1º ASIR
i Javier tiene alguna materia suspensa.
ii Iván tiene alguna materia suspensa.
2º ASIR
i Luis Fernando tiene alguna materia suspensa.
1º DAI
i Irina tiene alguna materia suspensa.
ii Miguel tiene alguna materia suspensa.

5.- Bucles y condicionales.

En XSLT existen una serie de instrucciones que sirven para implementar bucles y saltos condicionales en un sentido parecido a los existentes en los lenguajes de programación.

5.1.- <xsl:for-each …>:

La traducción al español sería "para cada…". Si se lo aplicamos a un conjunto de nodos se producirá un bucle que va recorriendo cada uno de los nodos de este conjunto, de forma que podemos aplicarle una transformación a cada uno de ellos.

Solamente dispone del atributo **select** que contiene la expresión XPath que devuelve el conjunto de nodos sobre el que vamos a realizar el recorrido.

Veamos la sintaxis:

```
<xsl:for-each  select="XPath-conjunto de nodos" >
        ... procesamiento aplicado a cada nodo del conjunto ...
</xsl:for-each>
```

Esta instrucción se utiliza con bastante frecuencia. Es común verla en hojas que crean tablas. Anidando varias sentencias for-each lograremos crear tablas complejas. También en ocasiones observarás que el uso de for-each evita la creación de una plantilla nueva. Este es el caso de la siguiente práctica:

PRÁCTICA 18.- Crea practica18.xsl que transforme alumnos.xml en una página web como la siguiente:

Lista de cursos ofertados:

Código	Nombre
1	1º ASIR
2	2º ASIR
3	1º DAI

Utiliza for-each y una única plantilla.

PRÁCTICA 19.- Crea practica19.xsl que transforme alumnos.xml en una página web como la siguiente:

Lista de cursos alumnos:

Número	Curso	Cial	Nombre	Apellidos	Repetidor
1	1º ASIR	A18X111	Javier	Márquez Fernández	False
2	1º ASIR	A27M242	Maria	Álvarez Pérez	True
3	1º ASIR	A90R112	Iván	Rodriguez Pérez	False
4	2º ASIR	A92T861	Marcos	Dominguez	True
5	2º ASIR	A92R003	Luis Fernando	Jiménez Pérez	False
6	2º ASIR	A87R122	Elena	Ramirez	False
7	1º DAI	A95M600	Irina	Lugo Lugo	False
8	1º DAI	A92R543	Genaro	Hernández Hernández	True
9	1º DAI	A92R590	Miguel	Rodriguez	False

PRÁCTICA 20.- Crea practica20.xsl que agregue a la tabla anterior una columna con la suma de las notas del alumno:

Lista de cursos alumnos:

Número	Curso	Cial	Nombre	Apellidos	Repetidor	Total de puntos
1	1º ASIR	A18X111	Javier	Márquez Fernández	False	24
2	1º ASIR	A27M242	Maria	Álvarez Pérez	True	23
3	1º ASIR	A90R112	Iván	Rodriguez Pérez	False	25
4	2º ASIR	A92T861	Marcos	Dominguez	True	NaN
5	2º ASIR	A92R003	Luis Fernando	Jiménez Pérez	False	NaN
6	2º ASIR	A87R122	Elena	Ramirez	False	NaN
7	1º DAI	A95M600	Irina	Lugo Lugo	False	24
8	1º DAI	A92R543	Genaro	Hernández Hernández	True	23
9	1º DAI	A92R590	Miguel	Rodriguez	False	7

¿Por qué motivo crees que hay tres alumnos que obtienen como total de puntos el valor NaN? Resolveremos este problema más adelante.

PRÁCTICA 21.- Crea practica21.xsl que agregue a la tabla anterior una columna con la nota media del alumno:

Lista de cursos alumnos:

Número	Curso	Cial	Nombre	Apellidos	Repetidor	Total de puntos	Nota Media
1	1º ASIR	A18X111	Javier	Márquez Fernández	False	24	6
2	1º ASIR	A27M242	Maria	Álvarez Pérez	True	23	5.75
3	1º ASIR	A90R112	Iván	Rodriguez Pérez	False	25	6.25
4	2º ASIR	A92T861	Marcos	Dominguez	True	NaN	NaN
5	2º ASIR	A92R003	Luis Fernando	Jiménez Pérez	False	NaN	NaN
6	2º ASIR	A87R122	Elena	Ramirez	False	NaN	NaN
7	1º DAI	A95M600	Irina	Lugo Lugo	False	24	6
8	1º DAI	A92R543	Genaro	Hernández Hernández	True	23	5.75
9	1º DAI	A92R590	Miguel	Rodriguez	False	7	3.5

Para la siguiente práctica tendrás que utilizar varias sentencias for-each. Observa la tabla resultante para que ajustes bien el código html de la tabla.

PRÁCTICA 22.- Crea practica22.xsl que presente una tabla como la siguiente para todos los alumnos:

Lista de cursos alumnos:

Número	Curso	Cial	Nombre	Apellidos	Repetidor	Total de puntos	Nota Media
1	1º ASIR A18X111	Javier	Márquez Fernández		False	24	6
	a) Lenguajes de marcas					8	
	b) Redes					7	
	c) Fundamentos de hardware					3	
	d) Sistemas Operativos					6	
2	1º ASIR A27M242	María	Álvarez Pérez		True	23	5.75
	a) Lenguajes de marcas					5	
	b) Redes					5	
	c) Fundamentos de hardware					7	
	d) Sistemas Operativos					6	
3	1º ASIR A90R112	Iván	Rodríguez Pérez		False	25	6.25
	a) Lenguajes de marcas					4	
	b) Redes					9	
	c) Fundamentos de hardware					9	
	d) Sistemas Operativos					3	

5.2.- <xsl:if …>:

Sirve para realizar una ejecución condicional. Si una condición es cierta entonces se ejecutará el código escrito dentro de <xsl:if…>. Si la condición resulta ser falsa entonces el procesimiento continuará tras la sentencia <xsl:if>

```
<xsl:if  test=”expression booleana en XPath” >
        ... procesamiento aplicado a cada nodo del conjunto ...
</xsl:if>
```

- El atributo **test** es obligatorio y contendrá la condición a evaluar.

Ejemplo para imprimir un texto sólo para los alumnos con edad > 17 años:

```
<xsl:if test=”alumno/edad &gt; 17”>
    <p>El alumno es mayor de edad</p>
</xsl:if>
```

5.3.- <xsl:choose …>:

Es una sentencia if con múltiples opciones.

```
<xsl:choose>
    <xsl:when test=”condición1”>
       ... procesamiento aplicado si condición1 es cierta...
    </xsl:when>
    <xsl:when test=”condición2”>
       ... procesamiento aplicado si condición2 es cierta...
    </xsl:when>
            ..................
    <xsl:when test=”condiciónN”>
       ... procesamiento aplicado si condiciónN es cierta...
    </xsl:when>
    <xsl:otherwise>
       ... procesamiento aplicado si NINGUNA condición es cierta...
    </xsl:otherwise>
</xsl:choose>
```

- El atributo **test** es obligatorio en las cláusulas when. Las cláusulas when deben ser excluyentes entre sí. Observa el ejemplo:

```
Ejemplo para imprimir un texto sólo para los alumnos con edad > 17 años:
    <xsl:choose>
      <xsl:when test="temperatura &lt; 12">Clima frío </xsl:when>
      <xsl:when test="(temperatura &gt; 12) and (temperatura &lt; 20)">
            Clima templado </xsl:when>
      <xsl:when test="(temperatura &gt; 20)">
            Clima cálido</xsl:when>
    </xsl:choose>
```

PRÁCTICA 23.- Crea practica23.xsl que introduzca, en la práctica anterior las siguientes modificaciones:

En la columna Repetidor aparece el texto "Sí" sólo para aquellos alumnos que repiten. Si el alumno no repite la columna queda vacía. (Usa if).

Las notas aparecerán con su calificación en texto (Suspenso, Suficiente, Bien, Notable y Sobresaliente). Si la nota no fuera numérica copiaremos su valor (se aplica en el caso de APTO/No APTO. (Usa choose).

Lista de cursos alumnos:

Número	Curso	Cial	Nombre	Apellidos	Repetidor	Total de puntos	Nota Media
1	1º ASIR A18X111	Javier		Márquez Fernández		24	6
	a) Lenguajes de marcas					Notable	
	b) Redes					Notable	
	c) Fundamentos de hardware					Suspenso	
	d) Sistemas Operativos					Bien	
2	1º ASIR A27M242	Maria		Álvarez Pérez	Si	23	5.75
	a) Lenguajes de marcas					Suficiente	
	b) Redes					Suficiente	
	c) Fundamentos de hardware					Notable	
	d) Sistemas Operativos					Bien	
3	1º ASIR A90R112	Iván		Rodríguez Pérez		25	6.25
	a) Lenguajes de marcas					Suspenso	
	b) Redes					Sobresaliente	
	c) Fundamentos de hardware					Sobresaliente	
	d) Sistemas Operativos					Suspenso	
4	2º ASIR A92T861	Marcos		Dominguez	Si	NaN	NaN
	a) Seguridad Informática					Suficiente	
	b) Planificación de redes					Notable	
	c) Servicios web					Notable	
	d) Prácticas de empresa					APTO	

5.4.- <xsl:sort …>:

El término inglés "sort" podemos traducirlo como "ordenar". Esta sentencia se usa dentro (y justo al comienzo) de una sentencia <xsl:apply-templates> o <xsl:for-each> y se encargará de ordenar los nodos del conjunto de nodos al que se aplicará el apply-templates o el for-each.

Podríamos combinar varios <xsl:sort> para ordenar por varios criterios. Por ejemplo si tuviérmos dos <xsl:sort>, se ordenarían los resultados por el primero de ellos, y para los nodos con idéntico valor se aplicaría el segundo criterio. Si ocurriera que dos nodos tienen los mismos valores en todos los criterios entonces se respetará el orden en el documento xml de entrada.

La sintaxis es la siguiente:

```
<xsl:sort select=”Expresión Xpath – conjunto de nodos”
          data-type=”text|number” order=”asc|desc”
          case-order=”upper-first|lower-first” lang=”código lenguaje” />
```

- El atributo **select** contiene la expresión XPath correspondiente al conjunto de nodos que se ordenarán. Si no la usamos se aplica por defecto sobre el conjunto de nodos de la sentencia que contiene al <xsl:sort> .
- El atributo **data-type**: contendrá “text” para ordenar cadenas de texto o bien “number” para hacer ordenación numérica.
- El atributo **order**: será “ascending” para orden ascendente (de menor a mayor) o bien “descending” para ordenación descendente (de mayor a menor). Si no se especifica el valor por defecto es “ascending”.
- El atributo **case-order**: se usa en la comparación de cadenas para establecer cuándo se ordenarán antes las mayúsculas (“upper-first”) o las minúsculas (“lower-first”). El valor por defecto depende del lenguaje definido en el atributo **lang**, que contendrá el código del lenguaje (consultar una guía de referencia para ver los códigos).

PRÁCTICA 24.- Crea practica24.xsl que modifique la práctica 23 de forma que los alumnos aparezcan ordenados por apellido.

PRÁCTICA 25.- Crea practica25.xsl que modifique la práctica 23 de forma que los alumnos aparezcan ordenados primero por curso y después de mayor a menor nota media.

6.- Otras sentencias XSLT.

6.1.- <xsl:variable …>:

Una variable es una porción de memoria donde podremos guardar un dato. Generalmente una variable es el hecho de asociar esa porción de memoria con un nombre concreto. Este concepto es la clave de la programación. Veamos un ejemplo en el lenguaje de programación C:

```
int minumero = 8;  // declaramos una variable llamada minumero y le damos valor 8
minumero = 3*5;  // ahora la variable minumero contiene un 15.
minumero = minumero + 5;  // ahora la variable minumero contiene 15+5, un 20.
```

En XPath también se pueden usar variables, siempre anteponiéndoles el símbolo **$**. Sin embargo las variables tanto en XPath como en XSLT **no pueden cambiar de valor**. Es decir, se les puede asignar un valor en el momento de crearlas y utilizarlas tantas veces como queramos, pero no podremos modificarlas. Luego aunque se llaman variables en realidad corresponden a lo que en los lenguajes de programación se llaman **constantes.**

Tenemos dos **tipos de variables**:

- **Las variables globales:** se pueden utilizar en todo el fichero .xsl. Para hacer que una variable sea global basta con colocar la sentencia <xsl:variable…> como hijo directo del elemento raíz <xsl:stylesheet..>, es decir, como si fuera un elemento de primer nivel.
- **Las variables locales:** sólo se pueden utilizar dentro de la sentencia donde son declaradas. Imagina que colocamos un <xsl:variable…> dentro de un <xsl:for-ecah>. En este caso la variable creada sólo existirá mientras se esté ejecutando el bucle for-each. Fuera de este bucle no podemos acceder a la variable.

En cuanto al valor que le daremos a la variable también tenemos **dos formas** de indicarlo:

- Usando la variable en un atributo **select** de XSLT. La variable tomará el valor de la expresión.
- Colocamos directamente el valor en el momento de crear la variable con <xsl:variable…>

La sintaxis es la siguiente:

<xsl:variable name=”nombre“ **select=**”Expresión Xpath con valor”/>

Ejemplo:

```
<xsl:variable name=”alturaminima” select=”150”/>
<!—también podríamos declarar la variable de esta forma -->
<xsl:variable name=”alturaminima”>150</xsl:variable>
<xsl:for-each select=”//alumno[@altura > $alturaminima]”>
    <p>El alumno <value-of select=”nombre”/> cumple requisitos de altura.</p>
</xsl:for-each>
```

Nota: Observa cómo una vez declarada la variable se puede utilizar en XPath anteponiéndole el símbolo $.

PRÁCTICA 26.- Esta es una práctica avanzada. Crea practica26.xsl que muestre las materias y notas de los alumnos de 1º de ASIR. La dificultad se encuentra en que cuando estés analizando una asignatura deberás extraer la siguiente etiqueta nota del fichero (uso de ejes necesarios). Observa el resultado:

NOTAS:

Número	Alumno	Materias:
1	María Álvarez Pérez	Lenguajes de marcas: 5 Redes: 5 Fundamentos de hardware: 7 Sistemas Operativos: 6
2	Javier Márquez Fernández	Lenguajes de marcas: 8 Redes: 7 Fundamentos de hardware: 3 Sistemas Operativos: 6
3	Iván Rodríguez Pérez	Lenguajes de marcas: 4 Redes: 9 Fundamentos de hardware: 9 Sistemas Operativos: 3

PRÁCTICA 27.- Crea practica27.xsl que modifique la práctica anterior para que aparezcan únicamente las materias suspensas por cada alumno. Para esto define una variable global llamada aprobado con valor 5 y úsala para filtrar:

SUSPENSOS:

Número	Alumno	Materias Suspensas:
1	Maria Álvarez Pérez	
2	Javier Márquez Fernández	Fundamentos de hardware: 3
3	Iván Rodriguez Pérez	Lenguajes de marcas: 4 Sistemas Operativos: 3

Sugerencia: puedes utilizar una variable local llamada posición donde guardarás la posición de aquella nota que sea menor que aprobado. Utilizando esta posición podrás acceder al nodo asignatura correspondiente.

6.2.- <xsl:param …>:

En informática llamamos **parámetro** a una información que puede pasarse como argumento a una función. La idea es crear trozos de código (llamados funciones) que pueden aceptar distintos valores de entrada. Para tratar de entender esto veamos una función declarada en el lenguaje de programación C que calcula el máximo de dos cantidades:

```
void calculaMaximo (int parametro1; int parametro2) {
   if (parametro1 > parametro2)
        printf("\Máximo=%d", parametro1);
    else
        printf("\Máximo=%d", parametro2);
}
```

Una vez realizada esta función podemos usarla con distintos valores en sus parámetros de entrada. Veamos ejemplos:

calculaMaximo(37, 32) → imprime el texto "Máximo=37"
calculaMaximo(15, 87) → imprime el texto "Máximo=87"
calculaMaximo(15*3, 25-5) → imprime el texto "Máximo=45"

Pero no estamos dando clase de C, sino de XSLT. En XSLT la forma básica de transformar elementos es el uso de las plantillas. La idea será crear plantillas que puedan aceptar parámetros de entrada. Para hacerlo cada parámetro tendrá que tener un nombre único (establecido en el atributo **name**).

Cuando se llame a la plantilla lo primero que se hace es **determinar qué valor tiene cada uno de sus parámetros**:

1.- Si la plantilla que hace la llamada usa **<xsl:with-param>** entonces ése será el valor asignado al parámetro.

2.- Si la plantilla que hace la llamada no usa <xsl:with-param> entonces se usará el valor por defecto del parámetro, que se escribe en <xsl:param> usando el atributo **select** o bien dentro de <xsl:param>, igual que con las variables:

Ejemplo: <xsl:param name="edad" select="18"/> o bien
<xsl:param name="edad">18</xsl:param>

3.- Si <xsl:param> no establece un valor por defecto se asigna una cadena vacía como valor.

Nota importante: esto que contamos ocurre también con las variables. Si el valor lo ponemos dentro de select y es una cadena hay que usar dobles nivel de comillas (dobles fuera y sencillas dentro). Ejemplo:
<xsl:param name=”color” select=”’verde’”/>

Antes de ver un ejemplo de cómo usar los parámetros analicemos la sintaxis:

<xsl:param name=”nombre“ **select=**”valor por defecto”/>

6.3.- <xsl:with-param …>:

Este elemento sirve, como aclaramos en el apartado anterior, para establecer cambios en los valores de los parámetros. Es importante resaltar que sólo puede utilizarse dentro de **<xsl:call-template>** o dentro de <**xsl:apply-templates**>, y además debe referirse a un nombre ya declarado dentro de <xsl:param>.

<xsl:with-param name=”nombre“ **select=**”valor deseado”/>

También ocurre como en el caso de las variables y los parámetros, podemos omitir el atributo select si el valor lo especificamos dentro de la etiqueta. Son equivalentes:

```
<xsl:with-param name="color" select="'verde'"/>
<xsl:with-param name="color">verde</xsl:with-param>
```

6.4.- <xsl:call-template …>:

Es el último punto para necesario para entender el funcionamiento de los parámetros. Call-template permite invocar una plantilla a la que hemos dado nombre. El único atributo es **name** y su valor debe coincidir con el nombre asignado en una plantilla <xsl:template name=”…”>. Dentro de call-template usa un <xsl:with-param> por cada parámetro al que quieras dar un valor.

EJEMPLO: mostrar una tabla de alumnos de forma que el fondo es blanco y la letra normal pero si el alumno es repetidor el fondo sea rojo y la letra subrayada.

PASO 1: crearemos una plantilla llamada estudiante, que se encarga de escribir una celda de una tabla.

```
<xsl:template name="estudiante" match="alumno">
  <xsl:param name="fondo">white</xsl:param> <!—valores por defecto -->
  <xsl:param name="estilo">none</xsl:param>
  <td>
    <xsl:attribute  name="style"> <!—agregamos atributo style a td -->
      background-color:<xsl:value-of select="$fondo"/>;
      text-decoration:<xsl:value-of select="$estilo"/>
    </xsl:attribute>
    <xsl:value-of select="nombre"/><xsl:text> </xsl:text>
    <xsl:value-of select="apellidos"/>
  </td>
</xsl:template>
```

Como puedes observar la plantilla crea una celda de una tabla:

```
 <td style=”background-color:White;text-decoration:none”>
      Nombre y apellidos del alumno
</td>
```

PASO 2: creemos la plantilla principal. Cuando el alumno sea repetidor tendremos que llamar a la plantilla “estudiante” cambiando el parámetro fondo para poner valor “red” y el parámetro estilo para poner el valor “underline”.

```
<xsl:template match="/">
   <html><head><title>Alumnos:</title></head>
   <body>
    <h3>ALUMNOS:</h3>
    <table>
    <tr  style="background-color:pink;color:white">
      <td>Alumno</td></tr>
    <xsl:for-each select="/instituto/curso/alumno">
      <tr>
      <xsl:choose>
        <xsl:when  test="repetidor='True'">
          <xsl:call-template name="estudiante">
            <xsl:with-param name="fondo">red</xsl:with-param>
            <xsl:with-param name="estilo">underline</xsl:with-param>
          </xsl:call-template>
        </xsl:when>
        <xsl:when  test="repetidor='False'">
            <xsl:call-template name="estudiante"/>
        </xsl:when>
      </xsl:choose>
      </tr>
    </xsl:for-each>
    </table>
    </body>
    </html>
</xsl:template>
```

Observa que si el alumno no está repitiendo curso simplemente llamamos a la plantilla sin usar with-param, ya que se utilizarán los valores por defecto de los parámetros.

I.E.S. DOMINGO PÉREZ MINIK

CICLO FORMATIVO DE GRADO SUPERIOR:

"ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED (L.O.E.)"

MÓDULO:

Lenguajes de marcas y sistemas de gestión de información

(4 horas semanales)