Gestión de bibliotecas digitales
usando *Greenstone 3*

CLASE 2

Guía de la clase

Mariana Pichinini - Pablo Musa

XML

Gestión de bibliotecas digitales usando *Greenstone 3*

¿QUE ES XML?

Es un Lenguaje de Marcas eXtensible, creado para **transferir datos o almacenarlos.**

Los lenguajes de marcas son formas de codificación de documentos basados en etiquetas o marcas. El más común de estos lenguajes, es el HTML (Hyper Text Markup Language).

X Extensible se refiere a la posibilidad de generar nuestras propias marcas y hacer un modelo flexible y escalable.

M Marcado o etiquetado, se refiere a una forma o método de escribir metadatos.

L Lenguaje se refiere a implementar una sintaxis y una gramática propia.

Pero a diferencia de HTML, en XML no se utilizan marcas predefinidas, sino que las creamos los desarrolladores, teniendo en cuenta el conjunto de datos que queremos almacenar. Se dice entonces que es extensible por la flexibilidad de poder crear dichas marcas y poder escalar el modelo incorporando marcas nuevas.

XML

Gestión de bibliotecas digitales usando *Greenstone 3*

HTML

```
<html>
<head>
  <title>Bienvenidos</title>
</head>
<body>
        <h1>Bienvenidos</h1>
        <p>Clase 2 del Gestión de
        bibliotecas digitales
        usando Greenstone 3 </p>
</body
</html>
```

XML

```
<?xml version="1" encoding="utf8"?/>
<personajes>
  <personaje>
    <nombre>Martin Karadagian</nombre>
    <edad>60</edad>
    <peso>87</peso>
  </personaje>
  <personaje>
    <nombre>El Diábolo</nombre>
    <edad>34</edad>
    <peso>98</peso>
  </personaje>
</personajes>
```

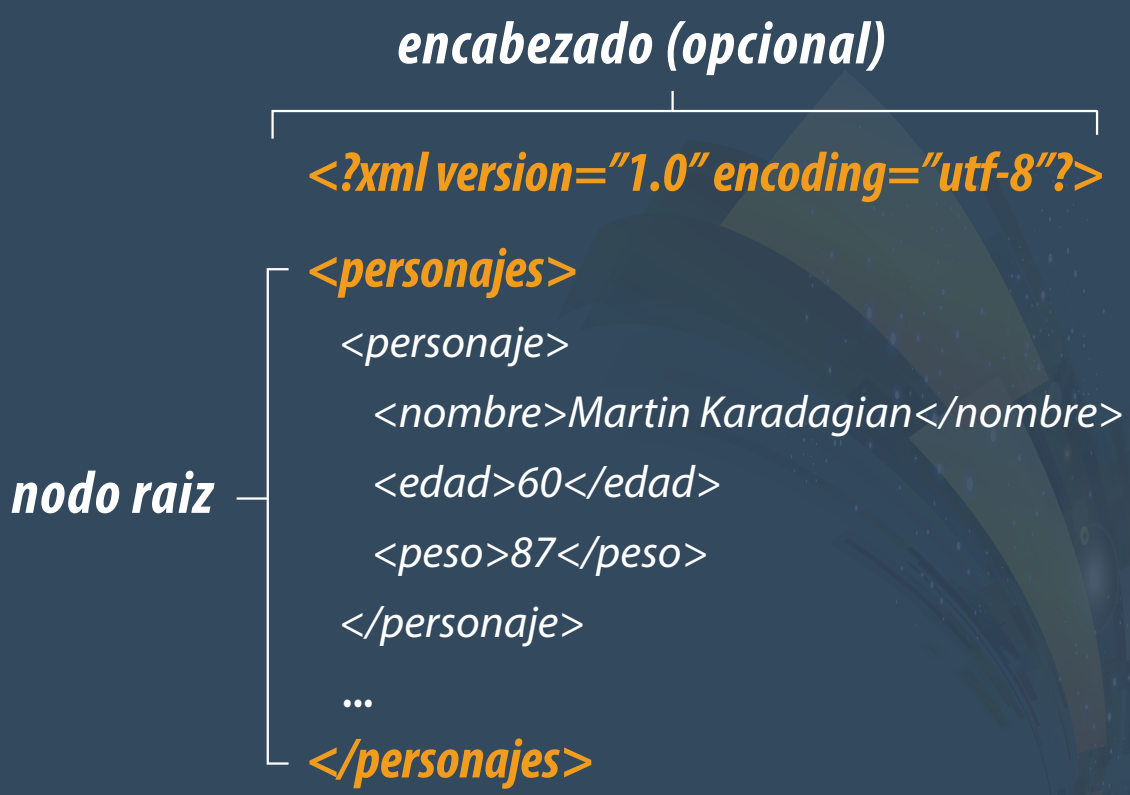
ESTRUCTURA

El XML tiene una estructura muy simple:

Primero se declara un encabezado donde se especifica la versión y la codificación de caracteres (es opcional pero se recomienda aplicarlo por el uso de los acentos y caracteres especiales).

En segundo término si existe una XSLT asociada, deberá declararse en la línea 2. Esto lo veremos un poco más adelante.

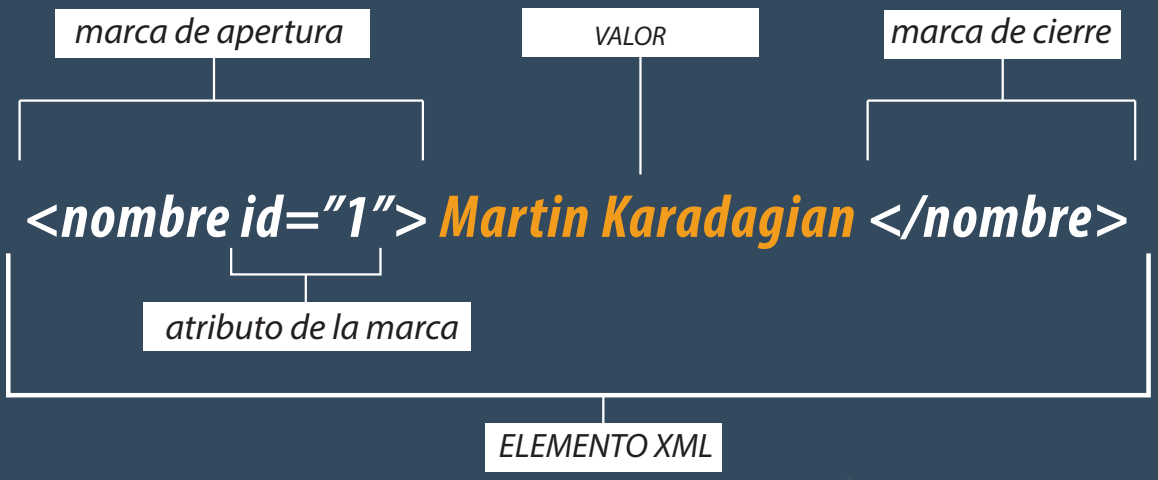
Por último, se declara una marca que encerrará todo el contenido de nuestro documento XML y que se denomina Nodo Raíz. Este nodo raíz, es OBLIGATORIO.



REGLAS Y SINTAXIS

Definimos una marca o tag como todo aquello que está entre menor y mayor "<...>".

Todas las marcas deben tener una marca de apertura y cierre.



Elemento es todo lo que hay entre una marca de apertura y cierre.Incluso otras marcas.

Las marcas son sensibles al uso de mayúsculas y minúsculas.

Ejemplo:

<A>esto no está bien

<A>esto si está bien</A>

Las marcas deben estar correctamente anidadas. Esto es, si hay dos marcas abiertas una dentro de otra, el orden de cierre es inverso al de apertura.

Ejemplo:

```
<descrip>
        Esto está muy <importante> MAL </descrip>
</importante>

<descrip>Ahora si esta muy
        <importante> BIEN</importante>
</descrip>
```

Dentro de las marcas pueden existir **atributos** declarados. Los valores de dichas declaraciones siempre debe ser **entrecomilladas, con simple o doble comillas**, dependiendo lo que se necesite.

Los valores de los atributos escritos en una marca, **no pueden tener contenido VACÍO**, siempre deben tener un valor.Ejemplo:

```
<mal id="">tiene el id vacio!</mal>

<bien id="nulo">ahora está correcto</bien>
```

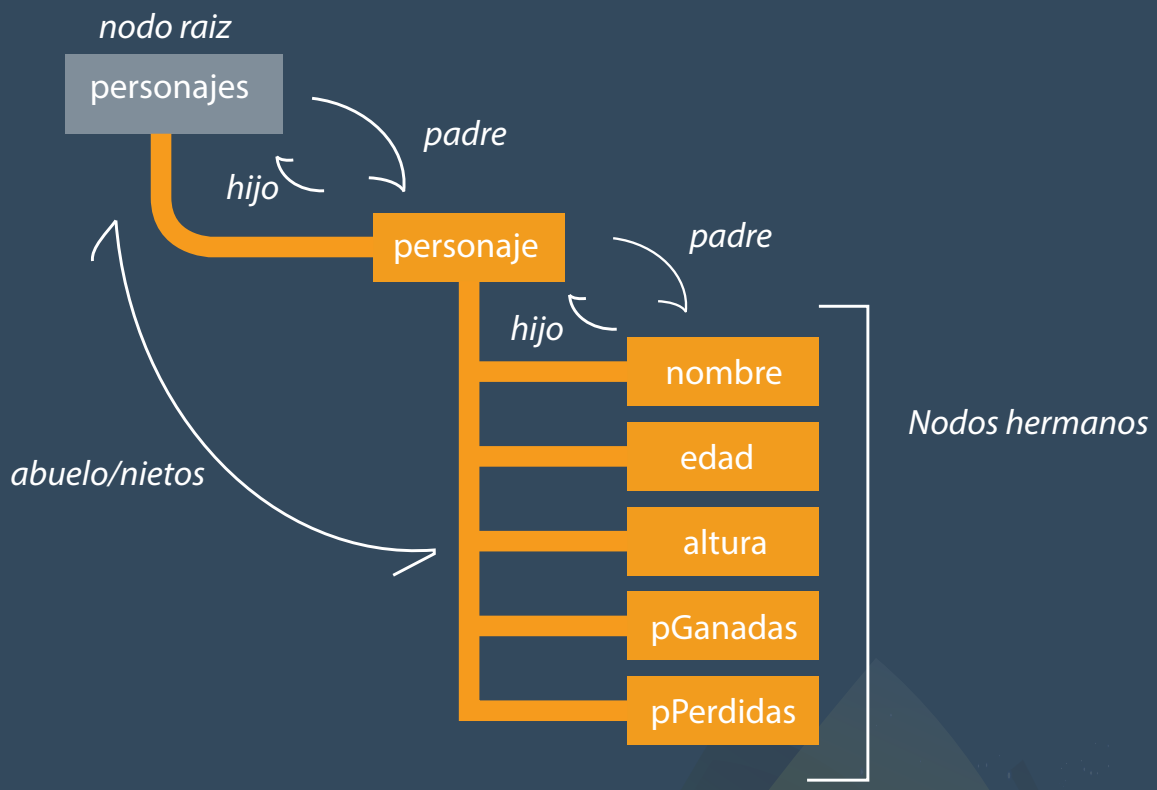
Si una marca está vacía puede abrir y cerrar en la misma marca de la forma:

```
<nombreMarca></nombreMarca>
                =
<nombreMarca  />
```

XML

Gestión de bibliotecas digitales usando *Greenstone 3*

RELACIÓN ENTRE NODOS



IMPORTANTE:
los nodos hijos son considerados ATRIBUTOS de los padres.

EL ESPACIO DE NOMBRE

El espacio de nombre está pensados para diferenciar marcas que llevan el mismo nombre. Por ejemplo, puede ser que en un xml en inglés donde se describen muebles se encuentre una marca "table" (mesa). Esta marca puede prestarse a confusión con la marca "table" del html. Para evitar la ambigüedad se utilizan los prefijos del espacio de nombres.

En la marca raiz (o en la que se va a usar) se define un prefijo que debe estar asociado a una URI (Identificador de Recursos Universal). Esto se hace de la siguiente manera:

xmlns:prefijo="http:/fahce.unlp.edu.ar/prefijo"

Coloquialmente decimos: voy a crear un XML NAME SPACE que se llama ***prefijo*** y que se encuentra especificado en esta ***URI*** .

El procesador XML no buscará ninguna información en dicha URI.

Declaración de los prefijos asociados con una URI

```
<h:table
   xmlns:h="http://w3.org/TR/html4/" >
 <h:tr>
  <h:td>Peras</h:td>
  <h:td>Bananas</h:td>
 </h:tr>
</h:table>
```

```
<f:table
   xmlns:f="https://fahce.edu/muebles">
 <f:name>Mesa ratona</f:name>
 <f:width>80</f:width>
 <f:length>120</f:length>
</f:table>
```

XML

Gestión de bibliotecas digitales usando *Greenstone 3*

XML SCHEMA

Los esquemas de xml son un conjunto de reglas escritas en xml que sirven de guía y control del documento xml que se va a generar.

Estos esquemas definen cual es la estructura del XML, la relación entre los nodos y que tipo de datos pueden contener.

Los esquemas reemplazan lo que originalmente fue la Definición de TIpo de Documento (DTD).

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="nota">                          Puede existir un elemento "nota"
 <xs:complexType>                                 con multiples elementos hijos
  <xs:sequence>                                   en un orden
   <xs:element name="para" type="xs:string"/>     un elemento text "para"
   <xs:element name="de" type="xs:string"/>       luego un texto "de"
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

XML

Gestión de bibliotecas digitales usando *Greenstone 3*

TAREA

-Dado los datos del documento TAREA.TXT armar un xml bien formado y guardarlo con el nombre hitos.xml.

-Pensar una estructura de metadatos teniendo en cuenta que el resultado que pretendemos es organizar los hitos por fecha.

-Identificar cada hijo del primer nivel para poder individualizarlo luego.

SEPARACIÓN DE CONTENIDO Y DISEÑO

Desde el comienzo del desarrollo de Internet, la W3C* planteó un paradigma de trabajo. La separación entre los datos y la forma de representación. Este planteo es compartido por la mayoría de los desarrolladores de software. Para el caso del XML, existen las XSLT, que son las plantillas que definirán una visualización de los datos.

Datos *Visualización*

* La W3C (World Wide Web Cosnsortium) es un consorcio fundado en los orígenes de Internet donde participan la mayoría de las empresas de tecnología con la finalidad de acordar estándares de desarrollo.

XSLT

Gestión de bibliotecas digitales usando *Greenstone 3*

¿QUE ES XSLT?

Es otro documento XML con un lenguaje de estilos que permite la transformación de un documento XML en otro XML,o TXT, SVG o PDF. XSLT es el acrónimo de eXtensible Stylesheet Language for Transformation.

X Extensible se refiere a que, por ser un documento XML, hereda todas las características de marcado y escalabilidad.

S Hoja de Estilo (Stylesheet) es la posibilidad de aplicar elementos de diseños para mejorar la presentación y lectura.

L Con Lenguaje se refiere a implementar una sintaxis y una gramática propia.

T Y con Transformar se refiere a que luego del procesamiento de la hoja de estilo puede obtenerse otro tipo de documento.

Una hoja de estilos XSLT consta de uno o más conjuntos de reglas que se llaman plantillas (templates).
Una plantilla contiene las reglas para aplicar a un conjunto o lista de nodos seleccionados con una expresión XPATH.

DECLARACIÓN: STYLESHEET O TRANSFORM

Para identificar que el archivo que estamos escribiendo es una hoja de estilo de transformación debemos declararla en el comienzo.

Como todo en documento XML es recomendable comenzar con el encabezado <?xml version=”1.0” encoding=”utf-8”?>

Luego aplicaremos el elemento XSLT que define una hoja de estilos:

<xsl:stylesheet> o <xsl:transform>

Puede usarse cualquiera de las dos marcas.

Como vemos se aplicó un espacio de nombres con el prefijo “xsl:”, por consiguiente es necesario declararlo como lo vimos en el bloque anterior.

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

Declaración del PREFIJO XSL

```
...

</xsl:stylesheet >
```

XSLT

ENLAZAR UNA XML CON SU XSLT

Dado que este curso está orientado al uso de Greenstone 3 y la web, vamos a usar un navegador (Firefox) para procesar los documentos.

Existen varios procesadores de XSLT, el que viene con Firefox es TransforMiix (https://developer.mozilla.org/es/docs/XSLT), y solo procesa XSLT versión 1. En realidad, todos los navegadores solo cumplen con la versión 1 de XSLT aunque este año la W3C ya recomienda el uso de la 3.

Pero bien, para nuestro trabajo vamos a enlazar el XML con la XSLT para poder ver el "fruto de nuestro aprendizaje"

Usaremos el archivo que está en el rar del campus y que se llama cartas.xml

Y en su segunda línea agregaremos:

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

Listo, ahora el XML tiene asociado una xslt que será su hoja de estilos. Para procesarla debemos arrastrar el archivo cartas.xml y soltarlo sobre el navegador.

El resultado será que todos los valores de todos los nodos se mostrarán bajo una sola marca <transformiix:result>. EL porqué de este resultado lo veremos a continuación.

XSLT

Gestión de bibliotecas digitales usando *Greenstone 3*

PROCESAMIENTO

Todo la tarea de procesamiento de un documentos XSLT consiste en la generación e iteración sobre listado de nodos. Siempre tendremos un listado de nodos actual sobre el que estaremos procesando un nodo particular de esa lista.

XSLT

Gestión de bibliotecas digitales usando *Greenstone 3*

Existen 2 elementos XSLT que iteran sobre un Listado de Nodos. Dichas marcas son:
xsl:apply-templates y xsl:for-each:

xsl: apply-templates

xsl: for-each

APPLY-TEMPLATES

La marca xsl:apply-templates genera un listado de nodos y recorre el documento xml buscando cual es el xsl:template que mejor responde a ese conjunto de nodos.

<xsl:apply-templates select="/personaje" >

atributo con la expresión de búsqueda en XPATH

Buscá un template que pueda formatear a los nodos que resulten de la siguiente búsqueda:

`dame los elementos personaje`

Si no declaramos nada en la xslt, el procesador aplicará por default una marca:

<xsl:apply-templates />, que supone la creación de una lista de nodos desde la raíz del documento a procesar y la aplicación de la mejor plantilla que encuentre declarada. Si no existe una plantilla declarada, por defecto aplicará una plantilla que el procesador tenga ya predefinida. O sea, siempre devolverá un resultado.

Es por esto que en nuestro ejemplo, nos devolvió un tag Transformiix, porque internamente puede procesar los nodos del XML.

XSLT

Gestión de bibliotecas digitales usando *Greenstone 3*

REGLA DE PLANTILLA

Template

Como inicio de nuestro archivo de transformación vamos a declarar un elemento de plantilla con la marca XSL:TEMPLATE, que definirá un estilo para aquellos nodos que se ajusten al criterio expresado en su atributo MATCH, veamos un ejemplo.

El ejemplo de la imagen procesa los nodos del XML que tengan el nombre de tag personaje.

XSLT

Gestión de bibliotecas digitales usando *Greenstone 3*

XPATH PRIMERA PARTE.

Como ven el criterio de búsqueda es una expresión XPATH. Este lenguaje es parte del conjunto de tecnologías XML y se utiliza para navegar un documento xml. Veremos las expresiones XPATH un poco más adelante.
Por ahora diremos que es parecido a escribir rutas en el DOS o en una terminal linux.

La barra	/	especifica nivel.
Dos barras	//	significan en cualquier nivel desde la raiz.
El punto	.	especifica el lugar en el que me encuentro.
Dos puntos	..	es el nivel superior, o el nodo padre.
Arroba	@	especifica el nombre del atributo.
Mayor y menor	> o <	si un valor es mayor o menor.

Por ejemplo, con el siguiente template especificamos que podemos procesar la raíz del documento xml.

```
<xsl:template match="/">
  Hola Mudo!!!
</xsl:template>
```

Escribamos este código en nuestro documento titanes.xsl, dentro de la marca stylesheet.
Esta regla será nuestro disparador para comenzar a procesar el documento XSLT.

Coloquialmente dicho, el proceso hasta acá sería:

En la línea 1 declaramos que lo que sigue es un xml.

En la línea 2, que además es una hoja de estilos de transformación.

En la línea 3, que hay un template que sabe cómo procesar los elementos del nodo raíz.

Por no estar declarada tácitamente, el procesador utiliza un apply-template que genera un listado de nodos desde la raíz y busca cual es el template que mejor responda a dicho conjunt. Sin dudas, el de la línea 3 es el que mejor se ajusta.

SALIDA HTML

Lo que vemos en la pantalla del Firefox es una representación (renderizado) de lo que dice el código. Para ver el código de salida que está mostrando el navegador apretaremos F12, y se abrirá la ventana de desarrollo web.

Allí veremos el código procesado de la xslt con sus marcas.

Lo que pretendemos es que nuestro resultado sea un HTML!

Para esto, vamos a utilizar otra marca de XSLT, que es xsl:output

```
<xsl:output method="html" version="4.0" encoding="UTF-8" indent="yes"/>
```

Nótese que esta marca se cierra en sí misma, y va como la primera hija de xsl:stylesheet. Esto especifica que el resultado de la transformación será un html.

Ahora deberíamos escribir una estructura html que será la salida de diseño.

```
<html>
    <head>
      <title>Adoro XSLT</title>
    </head>
```

```
<body>
    <div id=”envoltorio”>
    <h1>Hola Mundo</h1>
    <p>Nuestro primer código convertido desde un xml. Peeeero… acá no hay ningún dato del xml original!!! <span class="destacado">Esto es una estafa!</span></p>
    </div>
</body>
</html>
```

Perfecto pero, ¿a dónde escribimos el html?

Dentro del elemento xsl:template!

Muy bien, por último vamos a agregarle una hoja de estilos CSS al html para poder customizar un poco más la visualización.

Copiemos de los documentos del curso el archivo que dice “cartas.css” y pongamoslo en la carpeta donde se encuentra el xsl y el xml.

Ahora, en el código que creamos recién en la xslt, agregaremos la marca <link> que enlaza la css, más un código que agrega un tipo de fuente desde google font.

ITERACIÓN

For-each
Como explicamos antes, existen 2 formas de iterar sobre nodos, la primera es buscar un template (apply-templates) y la segunda es el elemento xslt xsl:for-each, que generá una lista de nodos y por cada uno aplicará lo que definamos dentro de la marca.

atributo con la expresión de búsqueda en XPATH

```
<xsl:for-each select="/personaje" >
    ...
</xsl:for-each>
```

"Por cada elemento que resulte de la siguiente búsqueda:
dame los nodos "personaje"
aplicale el formato que está entre la apertura y cierre de esta marca.

Aplicaremos esta marca dentro del body del html que creamos antes y en su interior aplicaremos un elemento xslt xsl:value-of, que nos imprime el valor de la marca en la que estamos.

El código es el siguiente:

```
<xsl:for-each select="./personajes/personaje">
  <div class="personaje">
    <xsl:value-of select="."/>
  </div>
</xsl:for-each>
```

Coloquialmente dice:

Por cada elemento (for-each) de la lista que generes con todos los tags personaje, que dependen del elemento personajes, que a su vez dependa del nodo en el que me encuentro en este momento (.)… o sea, de la raiz… aplicá una marca div de html y dentro de esta, imprimí el valor del elemento (value-of) que estés procesando actualmente. Uff…

Ahora parece rebuscado, pero cuando comprendemos la lógica todo se ilumina. Jeje

Aplicar y ver el resultado en el navegador.

La marca value-of, tiene el atributo select donde escribimos la expresión XPATH del elemento que deseamos imprimir su valor. Ahora imprime todo lo que hay en personaje… ¿y si quiero solo el nombre del personaje?

```
<xsl:value-of select="nombre"/>
```

XSLT

Gestión de bibliotecas digitales usando *Greenstone 3*

CONDICIONALES

Los condicionales son elementos que nos permiten filtrar contenidos por algún criterio. En XSLT existen 2 elementos que nos permiten condicionar la aplicación de un código.

xsl: if

*Esta sentencia condiciona la aplicación de una regla **si se cumple con un criterio**.*

xsl: choose
xsl:when
wsl:otherwise

*La sentencia CHOOSE se emplea para aplicar **múltiples condicionales.***

En el caso del condicional Si (if), por lo general lo ubicamos dentro de un bucle, pero se puede aplicar en cualquier otro caso.

```
<xsl:for-each match="/personaje" >
    <xsl: if  test="pGanadas &gt; 100">
        ...
    </xsl:if>
</xsl:for-each>
```

***Si (IF)** el nodo actual tiene un nodo **"pGanadas"** cuyo valor es **mayor que 100,** entonces, aplico la regla.*

XSLT

Gestión de bibliotecas digitales usando *Greenstone 3*

CONDICIONALES

Los condicionales son elementos que nos permiten filtrar contenidos por algún criterio. En XSLT existen 2 elementos que nos permiten condicionar la aplicación de un código.

xsl: if

*Esta sentencia condiciona la aplicación de una regla **si se cumple con un criterio**.*

xsl: choose
xsl:when
wsl:otherwise

*La sentencia CHOOSE se emplea para aplicar **múltiples condicionales.***

En el caso del condicional **Si** (if), el criterio de filtrado se expresa en el atributo **test**.

```
<xsl:for-each match="/personaje" >
    <xsl: if  test="pGanadas &gt; 100">
        ...
    </xsl:if>
</xsl:for-each>
```

***Si (IF)** el nodo actual tiene un nodo **"pGanadas"** cuyo valor es **mayor que 100,** entonces, aplico la regla.*

XSLT

Gestión de bibliotecas digitales usando *Greenstone 3*

El otro elemento XSLT que puede usarse para discriminar contenidos **según uno o varios criterios** es la marca *xsl:choose*

```
<xsl: choose>
<xsl:when test="tipo='arbitro'"></xsl:when>
<xsl:otherwise></xsl:otherwise>
</xsl: choose>
```

Coloquialmente podríamos leer:
Elija ***(choose)*** *cuando* ***(when)*** *el valor de marca* ***"tipo"*** *sea igual a "arbitro". Si no* ***(otherwise)****, aplique otra cosa.*

Utilizando estos condicionales vamos a integrar otra forma en el uso de las plantillas. Las plantillas con nombre.

LLAMADA A UN TEMPLATE POR SU NOMBRE

El tag xsl:template, tiene otra forma de uso que puede ordenar el código. Podemos utilizar el atributo **name**, que define un nombre que luego podrá ser invocado mediante la marca xsl:-call-template.

```
<xsl:call-template name="arbitro"/>
```

*Esto es como un atajo o llamada al template que tiene como atributo **name** el valor **"arbitro".***

Aquí definimos un template que lleva el nombre "arbitro".

```
<xsl:template name="arbitro">
    Hola soy el árbitro de la pelea, mi nombre es:
    <xsl:value-of select="nombre"/>
</xsl:template>
```

Hasta aquí tenemos una base XSLT. Ahora sería bueno profundizar más en el lenguaje XPATH que nos permite crear listas de nodos.

XSLT

Gestión de bibliotecas digitales usando *Greenstone 3*

XPATH SEGUNDA PARTE

EJES

Los ejes establecen un conjunto de nodos relativos al nodo actual.

ancestor	Selecciona todos los niveles superiores, padres abuelos etc.
ancestor-or-self	Selecciona todos los niveles superiores, incluso el nodo actual
attribute	Selecciona todos los atributos del nodo actual
child	Selecciona todos los hijos del nodo actual
descendant	Selecciona todos los niveles inferiores del nodo actual, hijos, nietos etc.
descendant-or-self	Lo mismo que el anterior incluyendo el nodo actual
following	Selecciona todo lo que está encerrado en el elemento actual
following-sibling	Selecciona todos los hermanos que siguen al nodo actual
namespace	Todos los nodos con espacio de nombre del nodo actual
parent	Selecciona el padre del nodo actual
preceding	Selecciona todos los elemento que estan antes que el nodo actual, excepto
	los que son directos antecesores, padre, abuelo
preceding-sibling	Seleccionea todos los hermanos que estan antes del nodo actual
self	Selecciona el nodo actual

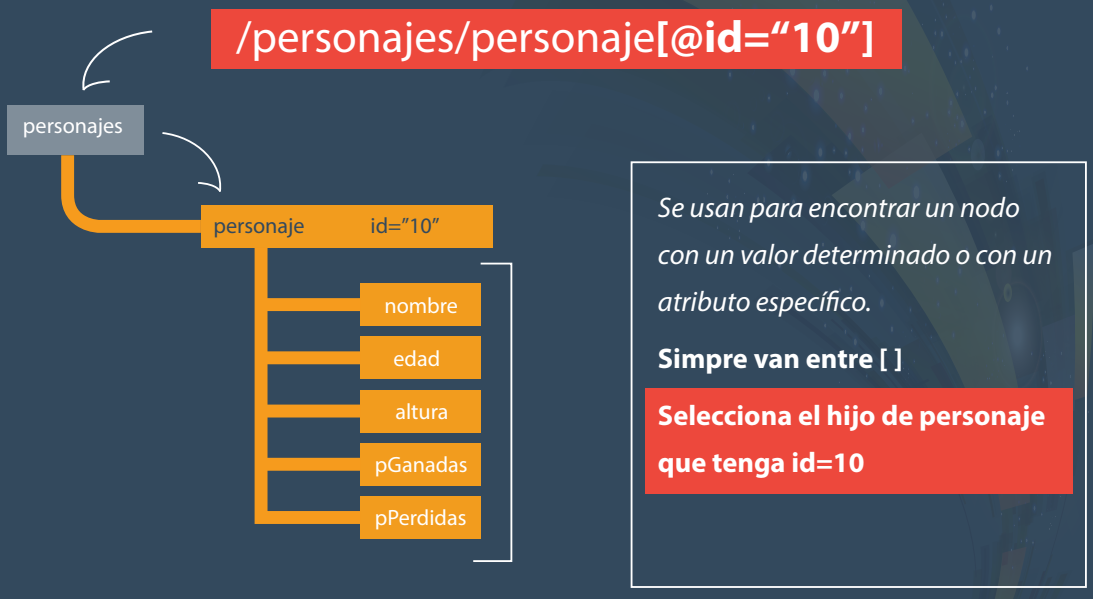
XSLT

Gestión de bibliotecas digitales usando *Greenstone 3*

XPATH SEGUNDA PARTE

PREDICADOS

Se usan para encontrar un elemento con un valor determinado o con un atributo específico. Siempre van entre [ ]



XSLT

Gestión de bibliotecas digitales usando *Greenstone 3*

XPATH SEGUNDA PARTE

OPERADORES

Los operadores sirven para hacer cálculos matemáticos simples, o aplicar expresiones booleanas.

|	Computa 2 conjuntos de nodos	//libros | //cd
+	Suma	6 + 4
-	Resta	6 - 4
*	Multiplica	6 * 4
div	Divide	8 div 4
=	Igual	precio=9.80
!=	Distinto	precio!=9.80
<	Menor que	precio<9.80
<=	Menor o igual que	<=9.80
>	Mayor a	precio>9.80
>=	Mayor o igual a	precio>=9.80
or	O	precio=9.80 or precio=9.70
and	Y	precio>9.00 and precio<9.90
mod	Módulo (resto en la división)	5 mod 2

Para finalizar agrego los enlaces con el listado completo de funciones y elementos que completan lo que vimos

Funciones XPATH

https://www.w3schools.com/xml/xsl_functions.asp

Listado de Elemento XSLT

https://www.w3schools.com/xml/xsl_elementref.asp

SORT

Vamos a ir finalizando la clase explicando 2 marcas más de XSLT. La marca xsl:sort que se utiliza para ordenar la salida o resultado de un bucle por uno de los nodo de dicha lista. Por defecto se ordena en forma ascendente.

```
<xsl:for-each select="/personaje" >
    <sort select="pGanadas"/>
     <h3>
          <xsl:value-of select="nombre"/>
     </h3>
</xsl:for-each>
```

Por cada elemento que resulte de la siguiente búsqueda:
`dame los nodos "personaje"`
y ordenalos por su la marca `pGanadas`.

Esta marca nos muestra otra lógica utilizada en XSLT, que es la de especificar con una marca el seteo del padre. Nótese que xsl:sort se utiliza como marca sin valor, y que sus atributos determinan como debe ser la salida de la marca padre xsl:for-each. Otro caso como este es la marca xsl:attribute, que veremos a continuación.

ATTTIBUTE

Este elemento define un valor y nombre de atributo para el nodo padre.

En el siguiente ejemplo vamos a construir un enlace con una marca "anchor" de HTML y vamos a agregar el valor del atributo href, mediante el uso de esta marca.

```
<a alt="">
  <xsl:attribute name="href">
   <xsl:value-of select="audio"/>
  </xsl:attribute>
</a>
```

Con el uso de esta marca cerramos la clase.

Desde luego que hay más elementos para investigar que son más avanzados, tanto en XSLT como en XPATH. Pero esto es la base que necesitamos para a encarar las XSLT de greenstone y comprender que sucede en su desarrollo.

XSLT

Gestión de bibliotecas digitales usando *Greenstone 3*

TAREA

- Crear una hoja de estilos XSLT y enlazarla al documento hitos.xml.

- Transformar el xml en html.

- Crear una visualización de línea de tiempo de los 10 hitos, que se horizontal y que nos permita utilizar el scroll para movernos.

La línea de tiempo debe tener un formato parecido al siguiente: