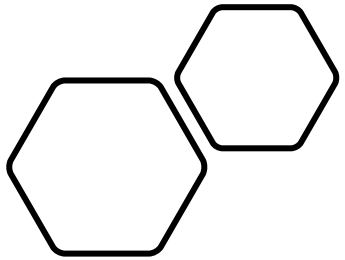
Curso 2023-2024



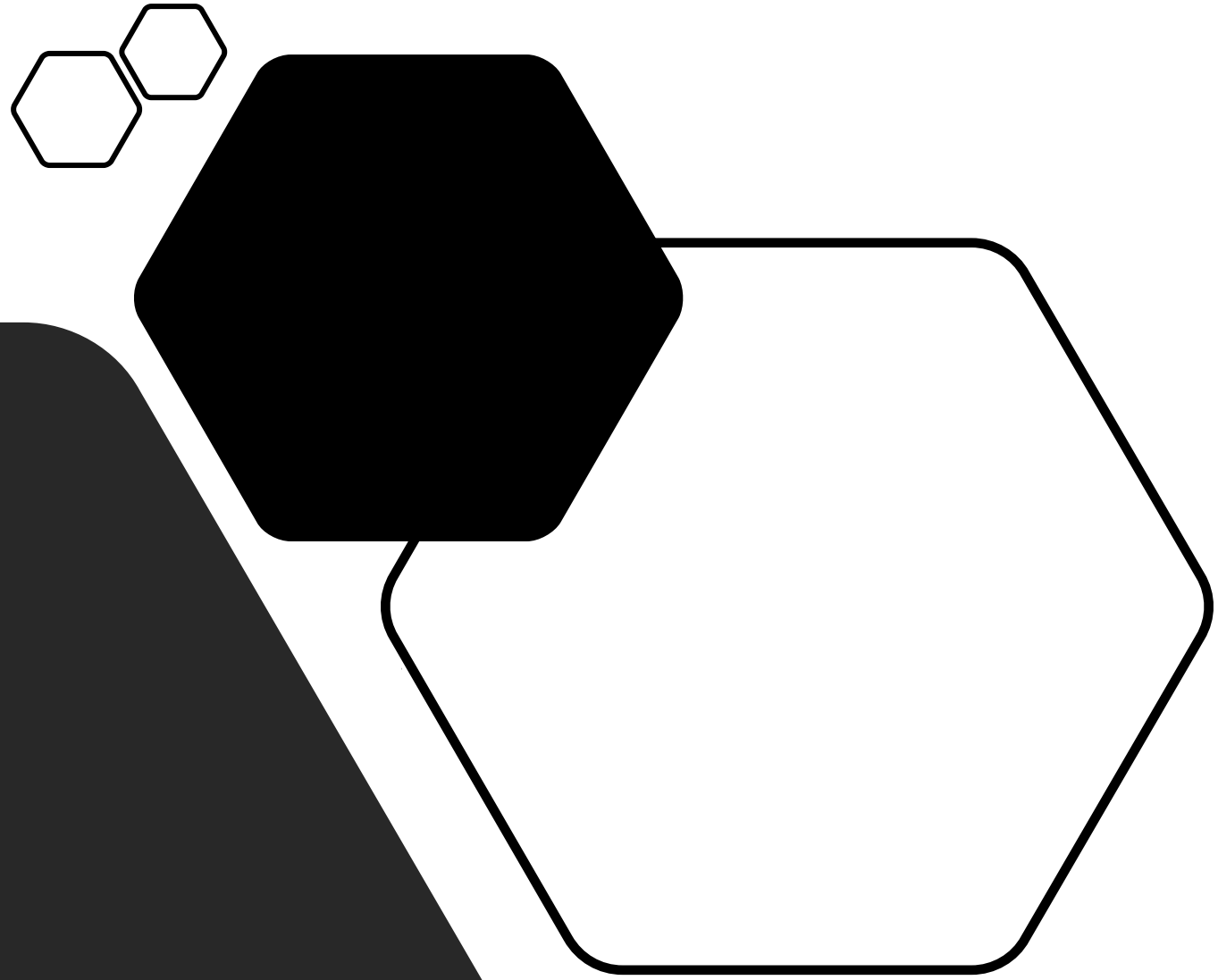
UD5: XML, conversión y adaptación

Lenguajes de marcas y sistemas de gestión de información

XML. Conversión y adaptación

- En esta unidad seguiremos profundizando en el trabajo con documentos XML.
- Estudiaremos XPath y XSLT.
- **XPath** permite extraer datos de un documento XML. En su última versión también es compatible con JSON.
- **XLT** permite generar ficheros XML o ficheros con formato, a partir del contenido de uno o varios ficheros XML.

https://xpath.playground.fontoxml.com/



XPath

XPath

- XPath es un estándar aprobado por el W3C, que permite navegar entre los **elementos** y **atributos** de un documento XML.
- Se basa en las relaciones de parentesco entre los **nodos** del documento.
- La versión actual de Xpath es la 3.1, de 2017.
- Se usa definiendo **expresiones de camino** (*paths*) para seleccionar nodos o conjuntos de nodos de un documento XML. Es parecido a las rutas que usamos en los sistemas de ficheros.

XPath

- Las expresiones que usamos para definir un camino se basan en el hecho de que el documento XML tiene una estructura interna de árbol.
- El resultado de aplicar una expresión es un conjunto de nodos.

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

<alumnos>
    <alumno genero="mujer">
        <numeroExpediente>1023</numeroExpediente>
        <nombre>María Vázquez</nombre>
        <fechaNacimiento>25/04/2004</fechaNacimiento>
        <telefono>676767676</telefono>
        <telefono>999888777</telefono>
    </alumno>
</alumnos>
```

XPath

- Una expresión de camino para el ejemplo de la página anterior puede ser /alumnos/alumno/nombre
  - `<nombre>María Vázquez</nombre>`
- Otra expresión válida es /alumnos/alumno/teléfono
  - `<telefono>676767676</telefono>`
  - `<telefono>999888777</telefono>`

Términos básicos

- Nodos
  - Un documento XML tiene estructura de árbol. Un elemento raíz contiene hijos, que a su vez contienen otros hijos.
  - A su vez, cada elemento puede tener atributos y/o contenido textual.
  - XPath trata los documentos XML como un árbol de nodos.
  - Podemos definir cuatro tipos de nodos:
    - Nodo raíz.
    - Nodo elemento.
    - Nodo atributo.
    - Nodo de texto.

Nodo raíz

- Es el nodo que encapsula al resto del documento.
- Se referencia con /
- Características:
  - No puede tener nodo padre.
  - Puede tener 0 o más nodos hijo.
  - Los nodos hijo pueden contener nodos de elemento, texto, instrucción o comentario.
  - Si el documento XML está bien formado, el nodo raíz debe tener un único nodo hijo de tipo elemento.
- Propiedades:
  - children
  - string

Nodo raíz

- En este ejemplo, el nodo raíz contiene un único hijo (alumnos).
- La propiedad children nos retorna un conjunto de nodos hijos.
- La propiedad string nos retorna todo el contenido textual de sus nodos descendientes.

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

<alumnos>
    <alumno genero="mujer">
        <numeroExpediente>1023</numeroExpediente>
        <nombre>María Vázquez</nombre>
        <fechaNacimiento>25/04/2004</fechaNacimiento>
        <telefono>676767676</telefono>
        <telefono>999888777</telefono>
    </alumno>
</alumnos>
```

Nodos elemento

- Cada nodo elemento representa cada uno de los elementos que aparecen dentro del nodo raíz.
- En el ejemplo que estamos usando, los nodos elemento son: alumnos, alumno, numeroExpediente, nombre, fechaNacimiento, telefono.
- Características:
  - Tienen siempre un nodo padre.
  - Pueden tener identificadores únicos, en forma de atributo.

Nodos atributo

- Son un tipo de nodo que representa un atributo XML.
- En el ejemplo que estamos usando, un nodo atributo es genero="mujer“
- Características:
  - Tienen siempre un nodo padre.
  - Pueden tener identificadores únicos, en forma de atributo.
- Propiedades:
  - node
  - parent
  - type
  - string

Nodos texto

- Son un tipo de nodo que encapsula contenido de caracteres XML.
- En un elemento, el nodo texto es aquel contenido entre la etiqueta de apertura y cierre.
- Características
  - El contenido puede estar vacío.
  - Solo si el padre de un nodo de texto está vacío, el contenido del nodo será una serie vacía.
- Propiedades:
  - content
  - parent

Ítems

- Los ítems pueden ser:
  - Nodos:
    - <nombre>
    - <alumno>
    - <telefono>
  - Valores atómicos:
    - 1023
    - Mujer
    - 25/04/2004

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

<alumnos>
    <alumno genero="mujer">
        <numeroExpediente>1023</numeroExpediente>
        <nombre>María Vázquez</nombre>
        <fechaNacimiento>25/04/2004</fechaNacimiento>
        <telefono>676767676</telefono>
        <telefono>999888777</telefono>
    </alumno>
</alumnos>
```

Relaciones entre nodos

- Las relaciones entre nodos pueden ser:
  - Padre-hijo.
  - Hermanos.
  - Ascendientes.
  - Descendientes

Expresiones

- XPath usa expresiones para obtener los datos requeridos de un XML.
- La **evaluación de una expresión** consiste en el análisis e interpretación de la expresión XPath, obteniendo así el resultado requerido.
- La evaluación de una expresión debe hacerse contra un documento XML, que es la fuente de donde obtendremos la información.

Expresiones. Tipos de datos

- La evaluación de expresiones XPath puede dar como resultado uno de los siguientes:
  - Un **conjunto de nodos**.
  - Una cadena de **texto**.
  - Un **número**.
  - Un **booleano**.
- Un conjunto de nodos tiene las siguientes características:
  - No está ordenado.
  - Todos los elementos de un conjunto son hermanos, independientemente de la relación original.
  - Aunque los hijos de los nodos que forman un conjunto de nodos son accesibles, los subárboles no se consideran elementos del conjunto.

Procesamiento de expresiones

- Cuando estamos procesando un documento XML, podemos hablar de los siguientes conceptos:
  - **Nodo actual**: aquel en el que se encuentra el procesador
  - **Nodo contexto**: cada expresión está formada por subexpresiones que se van evaluando antes de resolver la siguiente. El conjunto de nodos obtenido tras evaluar la expresión y que se utiliza para evaluar la siguiente es el nuevo contexto.
  - **Tamaño del contexto**: el número de nodos que se están evaluando en un momento dado en una expresión XPath.

Direccionamiento

- El direccionamiento permite localizar nodos dentro de un documento XML.
- Una ruta de localización (*location path*) es la ruta que hay que seguir en un árbol de datos para localizar ese nodo.
- Una ruta de localización siempre devuelve un conjunto de nodos, aunque puede estar vacío.
- La creación de una ruta de localización se consigue mediante la unión de varios pasos de localización.

Tipos de rutas

- Existen dos tipos de rutas:
  - **Absolutas**: comienza desde el nodo raíz y describe la ubicación exacta de un elemento en el árbol de elementos. Se especifica empleando una barra / al inicio de la ruta.
  - **Relativas**: parte del contexto actual y especifica la ruta en relación a dicho contexto.

Localización

- Localizar del nodo raíz: /
- Localizar elementos: /elemento1/elemento2…
- Localizar descendientes: //
- Localizar atributos: @
- Localizar nodos de texto: text()
- Localizar un nodo padre: ..
- Localizar el propio nodo: .

Comodines

*****	Hace referencia a cualquier nodo
@*	Hace referencia a cualquier atributo
node()	Hace referencia a cualquier nodo, independientemente del tipo de nodo

Predicados

- Un predicado es una **expresión booleana** que añade un nivel de verificación al paso de localización.
- En estas expresiones podemos incorporar **funciones XPath**.
- Usando rutas de localización podemos seleccionar varios nodos a la vez, pero usando predicados podemos seleccionar aquellos nodos que **cumplen ciertas características**.
- Sintaxis:
  - Los predicados se incluyen dentro de una ruta de localización usando **corchetes []**.
  - Podemos unir varios predicados mediante **operadores lógicos and**, **or**, **not**.

Predicados

```
<biblioteca>
    <libro num_paginas="400">
        <titulo>Los miserables</titulo>
        <autor>Victor Hugo</autor>
        <anho_publicacion>1862</anho_publicacion>
    </libro>
    <libro num_paginas="50">
        <titulo>El caos reptante</titulo>
        <autor>H.P Lovecraft</autor>
        <autor>Winifred V. Jackson</autor>
        <anho_publicacion>1862</anho_publicacion>
    </libro>
</biblioteca>
```

- Si queremos seleccionar el número de páginas de un libro con un valor inferior a 300 podríamos hacerlo de la siguiente forma:

/biblioteca/libro[@num_paginas<300]/@num_paginas

Funciones

- XPath incorpora una serie de funciones, las cuales pueden ser empleadas para realizar operaciones sobre los conjuntos de elementos que devuelven los predicados.
- Algunas de las funciones nos permiten indicar o trabajar con índices. Es importante indicar que los índices en XPath comienzan en 1.

Funciones que operan sobre números

- sum(arg1, arg2,… argN): devuelve la suma de una secuencia de valores.

```
sum(/biblioteca/libro/@num_paginas)
```

```
450
```

- max(arg1, arg2,… argN): devuelve el mayor de una secuencia de valores.

```
max(/biblioteca/libro/@num_paginas)
```

```
400
```

- min(arg1, arg2,… argN):devuelve el menor de una secuencia de valores.

```
min(/biblioteca/libro/@num_paginas)
```

```
50
```

Funciones que operan sobre números

- avg(arg1, arg2,… argN): devuelve el promedio de una secuencia de valores.

```
avg(/biblioteca/libro/@num_paginas)
```

```
225
```

- ceiling(número): redondea al alza.

```
ceiling(avg(/biblioteca/libro/@num_paginas)div 2)
```

```
113
```

- floor (número): redondea a la baja.

```
floor(avg(/biblioteca/libro/@num_paginas)div 2)
```

```
112
```

Funciones que operan sobre números

- count(arg1, arg2,… argN): devuelve el número de valores de una secuencia.

```
count(/biblioteca/libro/@num_paginas)
```

```
2
```

- abs(): devuelve el valor absoluto de la cantidad indicada.

```
abs(count(/biblioteca/libro/@num_paginas)*(-1))
```

```
225
```

Funciones que operan sobre textos

- string(nodo): convierte un objeto en una cadena de texto

```
/biblioteca/libro/titulo/string()
"Los miserables"
"El caos reptante“
```

- string-legth(): indica la cantidad de caracteres que forman un string.

```
string-length("cadena de texto")
15
```

Funciones que operan sobre textos

- lower-case(cadena): convierte todas las letras mayúsculas de una cadena en letras minúsculas.

```
lower-case("AbCdEfGh")
"abcdefgh"
```

- upper-case(cadena): convierte todas las letras minúsculas de una cadena en letras mayúsculas.

```
upper-case("AbCdEfGh")
"ABCDEFGH"
```

- concat(cadena1, cadena2): concatena dos cadenas de texto.

```
concat("cadena d","e texto")
"cadena de texto"
```

Funciones que operan sobre textos

- contains(cad1,cad2): devuelve true si cad1 contiene cad2.

```
contains("Texto1","ext")
```

```
true
```

- starts-with(cad1, cad2): comprueba si cad1 comienza por cad2.

```
starts-with("Texto1","Tex")
```

```
true
```

- ends-with(arg1,arg2): comprueba si la cad1 termina con cad2.

```
ends-with("Texto1","Tex")
```

```
false
```

Funciones que operan sobre textos

- matches(cad, regExp): comprueba si la cadena cad cumple la expresión regular.

```
matches("Texto1","^T.*1$")
```

```
true
```

- replace(cad, regExp, reemplazo): permite sustituir con reemplazo las partes de cad que cumplan la expresión regular.

```
replace('Esta es la cadena original', 'es', 'era’)
```

```
"Esta era la cadena original"
```

```
replace('Esta es la cadena original', 'c.*ena', 'condena’)
```

```
"Esta es la condena original"
```

Funciones lógicas

- true(): devuelve true.
- false(): devuelve false.
- not(): devuelve el valor lógico contrario.
- empty(): devuelve true si una secuencia está vacía.

```
empty(/biblioteca/libro/titulo/string())
```

```
false
```

- exists(): devuelve true si una secuencia no está vacía.

```
exists(/biblioteca/libro/titulo/string())
```

```
true
```

Funciones de posicionamiento

- last(): devuelve el índice del último nodo del tipo sobre el que se aplica la función.

```
/biblioteca/libro/last()
2
2
```

- position(): devuelve la posición de un nodo en su contexto.

```
/biblioteca/libro/position()
1
2
```

- name(): devuelve el nombre de un nodo.

```
/biblioteca/libro/name()
"libro"
```

Funciones de secuencias

- reverse(secuencia): invierte el orden de la secuencia.

```
reverse((1,2,3,4,5))
```

```
5,4,3,2,1
```

- remove(secuencia, num): elimina el índice indicado por num en la secuencia.

```
remove(("a","b","c","d","e"),3)
```

```
"a","b","c","d","e"
```

- distinct-values(secuencia): elimina de una lista los valores duplicados.

```
distinct-values((1,2,3,4,4,2))
```

```
1,2,3,4
```

Funciones de secuencias

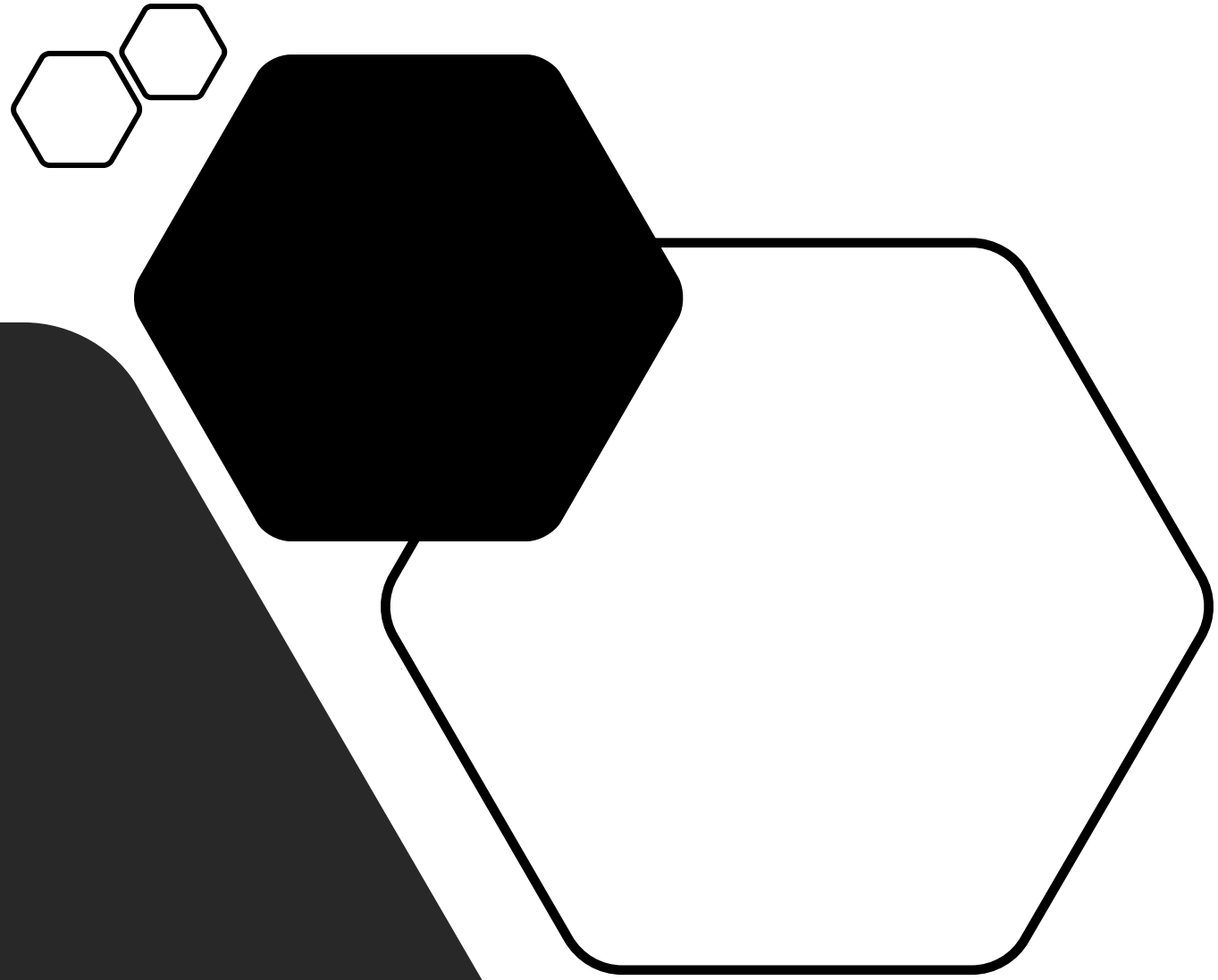
- index-of(secuencia, arg): devuelve el índice de arg en la secuencia.

```
index-of(/biblioteca/libro/titulo,"El caos reptante")
2
index-of(/biblioteca/libro/@num_paginas,"400")
1
```

Operadores aritméticos

and	or	mod	div
*	|	/	//
+	-	=	!=
<	<=	>	>=

https://www.freeformatter.com/xsl-transformer.html#before-output



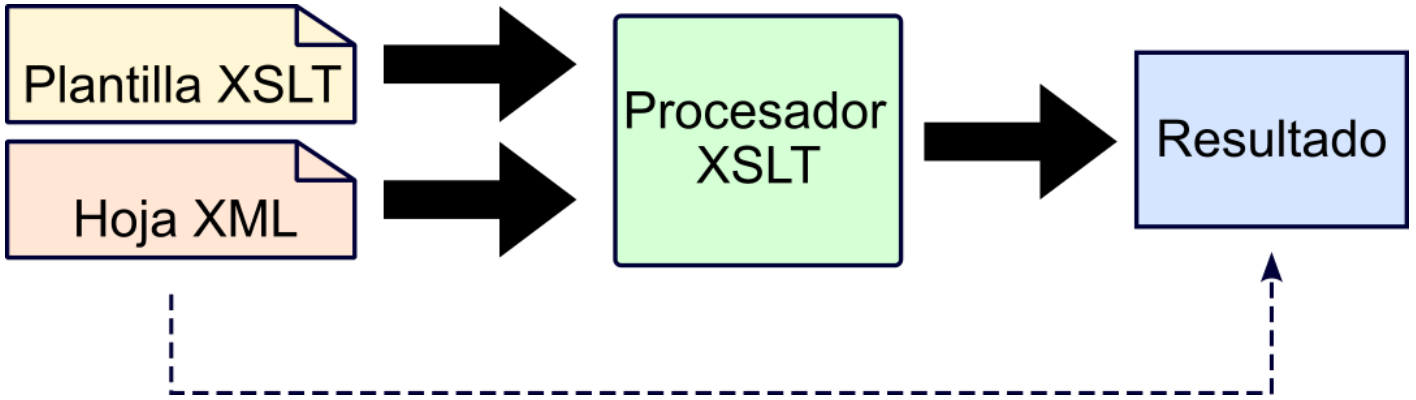
XSLT

XSLT

- XSLT (*eXtensible Stylesheet Language Transformations*) es un lenguaje estandarizado por el W3C, utilizado para transformar documentos XML a otros formatos, como HTML, o CSV.
- El estándar XSLT cuenta con múltiples versiones, siendo la más reciente la 3.0.
- Para realizar las transformaciones, se utilizan unas reglas definidas previamente que especifican cómo se deben extraer, procesar y organizar los datos del documento XML (origen) en el nuevo documento (destino).

Transformación de documentos

- El proceso de **transformación XSLT** consiste en la conversión de un documento XML a otro formato diferente. Esta conversión es llevada a cabo por un **procesador XSLT**.
- Para realizar parte del proceso, es necesario usar XPath para localizar las partes del documento XML sobre la que aplicaremos cada regla de transformación.



Hojas XSLT

- Los documentos que recogen las reglas de transformación XSLT se denominan hojas XSL.
- Están escritas en testo plano.
- Su extensión es .xsl.
- XSLT es también un documento XML (al igual que sucede con XSD).

Vinculación de hojas XSL

- Para realizar una transformación, es necesario vincular una hoja XSLT a un documento XML. Esto se consigue añadiendo la siguiente línea:

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

- Esta línea tiene que ir en el documento XML justo después del prólogo (y del DOCTYPE, en caso de usar DTD).

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE famosos SYSTEM "famosos.dtd">
<?xml-stylesheet type="text/xsl" href="agenda.xsl"?>
```

Elementos XSLT

- Dentro de una hoja XSLT podemos distinguir tres tipos de elementos:
  - **Elementos XSLT:** son los que usamos para definir las reglas de transformación XSLT. Se preceden del prefijo xsl:
  - **Elementos LRE (*Literal Result Element*)**: un elemento de resultado literal no pertenece a XSLT, sino que es repetido literalmente en la salida.
  - **Elementos de extensión**: no pertenecen al espacio de nombres xsl y son gestionados por implementaciones concretas del procesador. Estos normalmente no son usados.

Elementos XSLT

- **Elemento XSLT raíz**
  - Una hoja XSL puede tener como elemento raíz stylesheet o transform. Siendo ambos equivalentes. Lógicamente no pueden aparecer de forma simultánea, ya que en un documento XML no es posible tener más de un elemento raíz.
- Los hijos directos del elemento raíz son los **elementos de nivel superior**.
- Las **instrucciones** están contenidas dentro de los elementos de nivel superior.

stylesheet / transform

- El elemento stylesheet es el elemento raíz de una hoja XSL.
- La estructura es la siguiente:

```
<xsl:stylesheet version=“3.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

- Si un elemento stylesheet no tiene contenido, la transformación consistirá en la extracción de todos los datos de los elementos (los atributos no se extraen).
- Si el contenido de stylesheet está vacío dará como resultado un documento XML no válido, puesto que falta elemento raíz.

output

- El elemento output permite definir el formato de salida del documento.
- Es un elemento de nivel superior (hijo directo del elemento raíz). Todos sus atributos son opcionales.

```
<xsl:output />
```

method	version	encoding	omit-xml-declaration	standalone
doctype-public	doctype-system	cdata-section-elements	indent	media-type

output

- `method="xml|html|text"`
  - Define el formato de salida. Por defecto es xml.
- `version="string"`
  - Define la versión del formato de salida. Solo se aplica si method es html o xml.
- `encoding="string"`
  - Establece el sistema de codificación del fichero de salida.
- `omit-xml-declaration="yes|no"`
  - Indica si la declaración XML puede ser omitida o debe ser incluida. El valor por defecto es no.
- `standalone="yes|no"`
  - Indica si la declaración del standalone debe aparecer o ser optativa. El valor por defecto es no.

output

- `doctype-public="string"`
  - Permite establecer el valor del atributo PUBLIC de la declaración del DOCTYPE del fichero de salida.
- `doctype-system="string"`
  - Permite establecer el valor del atributo SYSTEM de la declaración del DOCTYPE del fichero de salida.
- `cdata-section-elements="namelist"`
  - Permite separar, mediante espacios en blanco, los elementos de una lista cuyo contenido debería estar escrito como secciones CDATA.
- `indent="yes|no"`
  - Indica si la salida debe estar indentada.
- `media-type="string"`
  - Permite establecer el tipo MIME del fichero de salida. Por defecto es text/xml.

template

- El elemento template es el bloque fundamental de una hoja XSL, el cual permite definir una plantilla que permite generar una salida conforme a lo definido en ella.

```
<xsl:template match="expresion-xpath" name="nombre">
</xsl:template>
```

- match es una expresión XPath que nos permite indicar el elemento a extraer.
- name define el nombre de la plantilla, que más tarde podremos invocar mediante call-template.

```xml
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="famosos.xsl"?>
<famosos>
    <famoso>
        <nombre>Michael Jackson</nombre>
        <apodo>El rey del pop</apodo>
        <profesion>cantante</profesion>
    </famoso>
    <famoso>
        <nombre>Frank Sinatra</nombre>
        <apodo>La voz</apodo>
        <fechaNacimiento>
            <anho>1915</anho>
            <mes>12</mes>
            <dia>12</dia>
        </fechaNacimiento>
        <profesion>cantante</profesion>
        <profesion>actor</profesion>
    </famoso>
</famosos>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text"/>
    <xsl:template match="/"></xsl:template>
</xsl:stylesheet>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text"/>
    <xsl:template match="//nombre"></xsl:template>
</xsl:stylesheet>
```

```
        El rey del pop
        cantante
        La voz
            1915
            12
            12
        cantante
        actor
```

call-template

- El elemento call-template permite invocar a un template definido previamente.

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

- name indica el nombre de una plantilla definida en la hoja XSL.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text"/>
    <xsl:template match="/">
        <xsl:call-template name="nombre-plantilla"/>
    </xsl:template>
    <xsl:template name="nombre-plantilla">
        Contenido de la plantilla
    </xsl:template>
</xsl:stylesheet>
```

```
Contenido de la plantilla
```

apply-templates

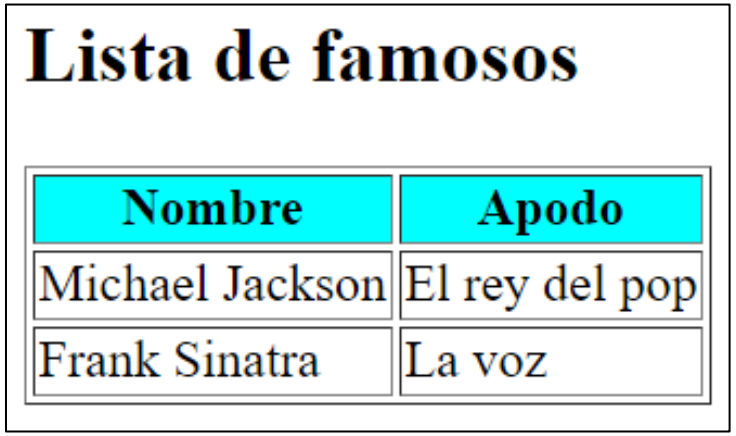
- El elemento apply-templates selecciona un conjunto de nodos del documento XML y le aplica un template.

```
<xsl:apply-templates select="expresion-xpath"/>
```

- name indica el nombre de una plantilla definida en la hoja XSL.
- apply-templates es útil porque si aplicamos dos elementos template que seleccionen nodos, y uno de ellos es un subconjunto del otro, solo se aplicará al que incluye más nodos, pero si usamos apply-templates podemos evitar esta limitación.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html><head><meta charset="UTF-8"/><title>Lista de famosos</title></head><body>
                <h2>Lista de famosos</h2>
                <table border="1">
                    <tr bgcolor="cyan">
                        <th>Nombre</th>
                        <th>Apodo</th>
                    </tr>
                    <xsl:apply-templates select="famosos/famoso"/>
                </table></body></html>
    </xsl:template>
    <xsl:template match="famoso">
        <tr>
            <td><xsl:value-of select="nombre"/></td>
            <td><xsl:value-of select="apodo"/></td>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

```html
<html>
   <head>
      <meta charset="UTF-8"/>
      <title>Lista de famosos</title>
   </head>
   <body>
      <h2>Lista de famosos</h2>
      <table border="1">
         <tr bgcolor="cyan">
            <th>Nombre</th>
            <th>Apodo</th>
         </tr>
         <tr>
            <td>Michael Jackson</td>
            <td>El rey del pop</td>
         </tr>
         <tr>
            <td>Frank Sinatra</td>
            <td>La voz</td>
         </tr>
      </table>
   </body>
</html>
```



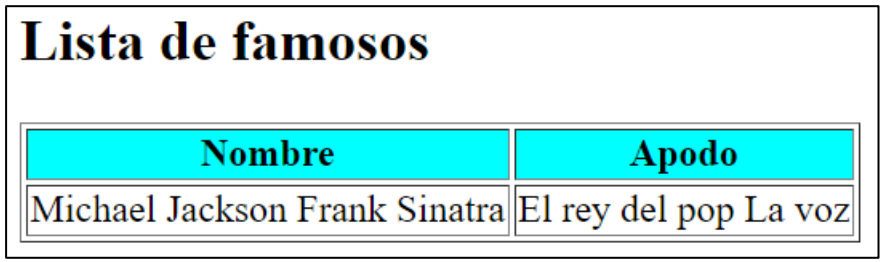
value-of

- El elemento value-of es utilizado para extraer el valor de un elemento XML y añadirlo como salida a la hora de realizar la transformación.

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

```xml
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html><head><meta charset="UTF-8"/><title>Lista de famosos</title></head><body>
                <h2>Lista de famosos</h2>
                <table border="1">
                    <tr bgcolor="cyan">
                        <th>Nombre</th>
                        <th>Apodo</th>
                    </tr>
                    <tr>
                        <td><xsl:value-of select="/famosos/famoso/nombre"/></td>
                        <td><xsl:value-of select="/famosos/famoso/apodo"/></td>
                    </tr>
                </table></body></html>
    </xsl:template>
</xsl:stylesheet>
```

```html
<html>
   <head>
      <meta charset="UTF-8"/>
      <title>Lista de famosos</title>
   </head>
   <body>
      <h2>Lista de famosos</h2>
      <table border="1">
         <tr bgcolor="cyan">
            <th>Nombre</th>
            <th>Apodo</th>
         </tr>
         <tr>
            <td>Michael Jackson Frank Sinatra</td>
            <td>El rey del pop La voz</td>
         </tr>
      </table>
   </body>
</html>
```



variable

- El elemento variable permite definir y declarar variables.

```
<xsl:variable name="nombre" select="expresion">
</xsl:variable>
```

- La ventaja de las variables es que podemos reutilizarlas en diferentes partes de la transformación. Se pueden almacenar valores estáticos o dinámicos (usando expresiones XPath).
- Para hacer uso del contenido de una variable la referenciamos con $:

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

copy-of

- El elemento copy-of permite crear una copia completa de un nodo.

```
<xsl:copy-of select="expresion"/>
```

- select es un atributo obligatorio, que indica qué va a ser copiado.

attribute

- El elemento attribute permite añadir atributos a los elementos. Este elemento permite reemplazar atributos ya existentes con nombres equivalentes.

```
<xsl:attribute name="nombre">
  <!-- template -->
</xsl:attribute>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml"/>
    <xsl:template match="/">
        <nombresFamosos>
            <xsl:apply-templates select="famosos/famoso"/>
        </nombresFamosos>
    </xsl:template>
    <xsl:template match="famoso">
        <nombreFamoso>
            <xsl:attribute name="apodo">
                <xsl:value-of select="apodo"/>
            </xsl:attribute>
            <xsl:value-of select="nombre"/>
        </nombreFamoso>
    </xsl:template>
</xsl:stylesheet>
```

```
<nombresFamosos>
   <nombreFamoso apodo="El rey del pop">Michael Jackson</nombreFamoso>
   <nombreFamoso apodo="La voz">Frank Sinatra</nombreFamoso>
</nombresFamosos>
```

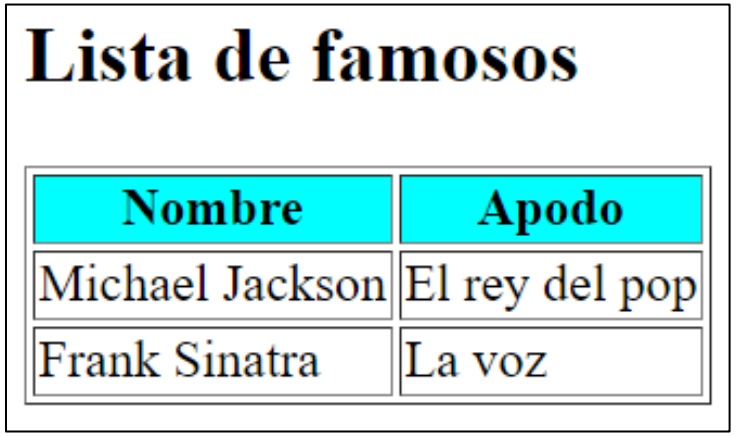
for-each

- El elemento for-each es un elemento iterativo que permite crear bucles dentro de archivos XSLT. Este elemento permite recorrer una serie de nodos seleccionados para obtener la información que estos contienen.

```
<xsl:for-each select="expresion-xpath">
  <!-- <xsl:value-of select="expresion"> -->
</xsl:for-each>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html><head><meta charset="UTF-8"/><title>Lista de famosos</title></head><body>
                <h2>Lista de famosos</h2>
                <table border="1">
                    <tr bgcolor="cyan">
                        <th>Nombre</th>
                        <th>Apodo</th>
                    </tr>
                    <xsl:for-each select="//famoso">
                        <tr>
                            <td><xsl:value-of select="nombre"/></td>
                            <td><xsl:value-of select="apodo"/></td>
                        </tr>
                    </xsl:for-each>
                </table></body></html>
    </xsl:template>
</xsl:stylesheet>
```

```html
<html>
   <head>
      <meta charset="UTF-8"/>
      <title>Lista de famosos</title>
   </head>
   <body>
      <h2>Lista de famosos</h2>
      <table border="1">
         <tr bgcolor="cyan">
            <th>Nombre</th>
            <th>Apodo</th>
         </tr>
         <tr>
            <td>Michael Jackson</td>
            <td>El rey del pop</td>
         </tr>
         <tr>
            <td>Frank Sinatra</td>
            <td>La voz</td>
         </tr>
      </table>
   </body>
</html>
```



if

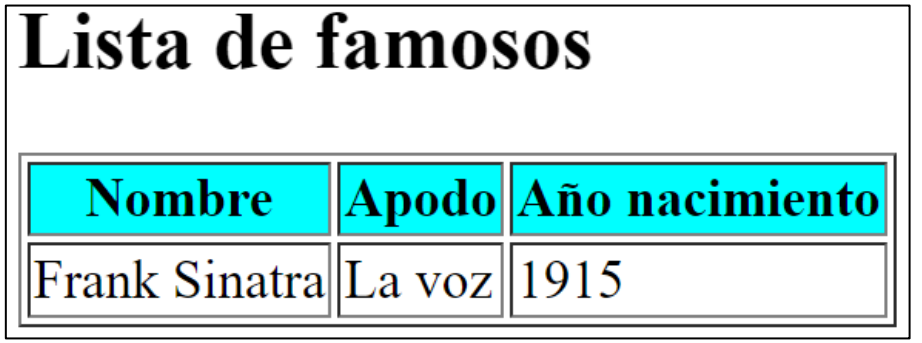
- El elemento if es un elemento condicional que permite modificar el XML de salida en función de si los datos de entrada cumplen una determinada condición.

```
<xsl:if test="expresion">
  <!-- Salida si se cumple la expresión -->
</xsl:if>
```

- test es un atributo que permite especificar la condición que se debe cumplir.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html>
            <head><meta charset="UTF-8"/><title>Lista de famosos</title></head>
            <body>
                <h2>Lista de famosos</h2>
                <table border="1">
                    <tr bgcolor="cyan">
                        <th>Nombre</th>
                        <th>Apodo</th>
                        <th>Año nacimiento</th>
                    </tr>
                    <xsl:for-each select="//famoso">
                        <xsl:if test="fechaNacimiento">
                            <tr>
                                <td><xsl:value-of select="nombre"/></td>
                                <td><xsl:value-of select="apodo"/></td>
                                <td><xsl:value-of select="fechaNacimiento/anho"/></td>
                            </tr>
                        </xsl:if>
                    </xsl:for-each>
                </table></body></html>
    </xsl:template>
</xsl:stylesheet>
```

```html
<html>
   <head>
      <meta charset="UTF-8"/>
      <title>Lista de famosos</title>
   </head>
   <body>
      <h2>Lista de famosos</h2>
      <table border="1">
         <tr bgcolor="cyan">
            <th>Nombre</th>
            <th>Apodo</th>
            <th>Año nacimiento</th>
         </tr>
         <tr>
            <td>Frank Sinatra</td>
            <td>La voz</td>
            <td>1915</td>
         </tr>
      </table>
   </body>
</html>
```



choose

- El elemento choose es un elemento que se suele utilizar en conjunción con when y otherwise para expresar múltiples condiciones. El elemento otherwise es opcional.

```
<xsl:choose>
   <xsl:when test="condicion1"><!-- … --> </xsl:when>
   <xsl:when test="condicion2"><!-- … --> </xsl:when>
   <!-- … -->
   <xsl:when test="condicionN"><!-- … --> </xsl:when>
   <xsl:otherwise><!-- … --> </xsl:otherwise>
</xsl:choose>
```

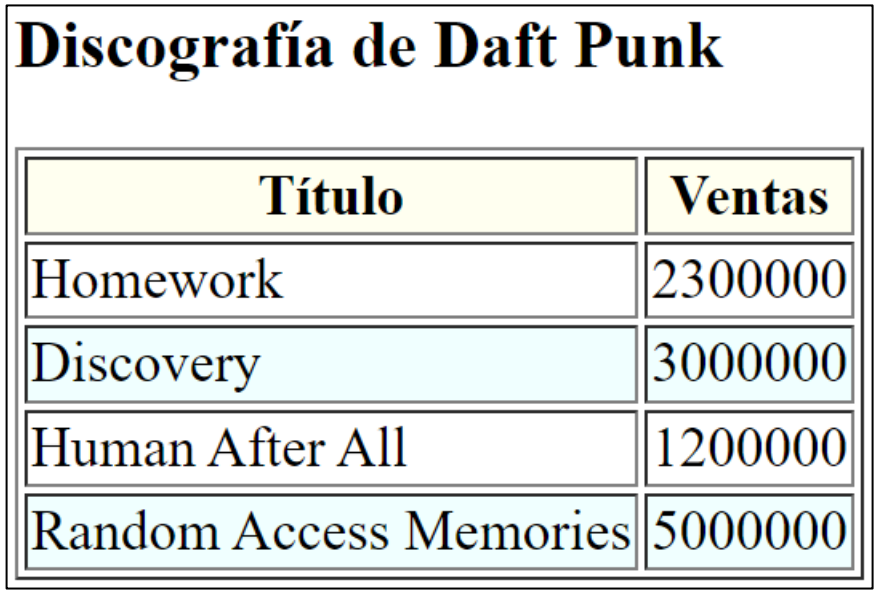
choose

- Si el valor evaluado por el elemento when es verdadero, se añadirá a la salida el código que éste contiene.
- Si el valor es falso, y está presente otherwise, se añadirá a la salida el contenido de éste.
- Si ningún elemento when es verdadero y no está presente el elemento otherwise, no se añadirá nada a la salida.

```xml
<discos>
    <disco grupo="Daft Punk">
        <titulo>Homework</titulo>
        <ventas>2300000</ventas>
        <lanzamiento>
            <anho>1997</anho>
            <mes>1</mes>
            <dia>20</dia>
        </lanzamiento>
    </disco>
    <disco grupo="Daft Punk">
        <titulo>Discovery</titulo>
        <ventas>3000000</ventas>
        <lanzamiento>
            <anho>2001</anho>
            <mes>3</mes>
            <dia>13</dia>
        </lanzamiento>
    </disco>
    <disco grupo="Daft Punk">
        <titulo>Human After All</titulo>
        <ventas>1200000</ventas>
        <lanzamiento>
            <anho>2005</anho>
            <mes>03</mes>
            <dia>14</dia>
        </lanzamiento>
    </disco>
    <disco grupo="Daft Punk">
        <titulo>Random Access Memories</titulo>
        <ventas>5000000</ventas>
        <lanzamiento>
            <anho>2013</anho>
            <mes>05</mes>
            <dia>17</dia>
        </lanzamiento>
    </disco>
</discos>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" _doctype-public="HTML"/>
    <xsl:template match="/">
        <html><head><meta charset="UTF-8"/><title>Lista de discos de Daft Punk</title>
            </head><body>
                <h3>Discografía de Daft Punk</h3>
                <table border="1" border-collapse="collapse">
                    <tr><th bgcolor="ivory">Título</th>
                        <th bgcolor="ivory">Ventas</th></tr>
                    <xsl:for-each select="//disco">
                        <tr>
                        <xsl:choose>
                            <xsl:when test="ventas >= 3000000">
                                <td bgcolor="azure"><xsl:value-of select="titulo"/></td>
                                <td bgcolor="azure"><xsl:value-of select="ventas"/></td>
                            </xsl:when>
                            <xsl:otherwise>
                                <td><xsl:value-of select="titulo"/></td>
                                <td><xsl:value-of select="ventas"/></td>
                            </xsl:otherwise>
                        </xsl:choose>
                        </tr>
                    </xsl:for-each>
                </table></body></html>
    </xsl:template>
</xsl:stylesheet>
```

```html
<html>
   <head>
      <meta charset="UTF-8"/>
      <title>Lista de discos de Daft Punk</title>
   </head>
   <body>
      <h3>Discografía de Daft Punk</h3>
      <table border="1" border-collapse="collapse">
         <tr>
            <th bgcolor="ivory">Título</th>
            <th bgcolor="ivory">Ventas</th>
         </tr>
         <tr>
            <td>Homework</td>
            <td>2300000</td>
         </tr>
         <tr>
            <td bgcolor="azure">Discovery</td>
            <td bgcolor="azure">3000000</td>
         </tr>
         <tr>
            <td>Human After All</td>
            <td>1200000</td>
         </tr>
         <tr>
            <td bgcolor="azure">Random Access Memories</td>
            <td bgcolor="azure">5000000</td>
         </tr>
      </table>
   </body>
</html>
```



Discografía de Daft Punk

Título	Ventas
Homework	2300000
Discovery	3000000
Human After All	1200000
Random Access Memories	5000000

sort

- El elemento sort permite ordenar los elementos que van a formar el XML de salida. Este elemento suele ir dentro de un elemento for-each.

```
<xsl:sort
   select="expression-xpath"
   order="ascending|descending"
   case-order="upper-first|lower-first"
   data-type="string|number"
/>
```

sort

- select indica el elemento XML por el que vamos a ordenar.
- order indica si se ordena en orden ascendente (ascending) o descendente (descending). El valor por defecto es ascending.
- case-order indica si se ordena antes por mayúsculas (upper-first) o minúsculas (lower-first). El valor por defecto es upper-first.
- data-type indica el tipo de dato a ordenar. Puede ser string o number, y el valor por defecto es string.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" _doctype-public="HTML"/>
    <xsl:template match="/">
        <html><head><meta charset="UTF-8"/><title>Lista de discos de Daft Punk</title>
            </head><body>
                <h3>Discografía de Daft Punk</h3>
                <table border="1" border-collapse="collapse">
                    <tr><th>Título</th>
                        <th>Ventas</th></tr>
                    <xsl:for-each select="//disco">
                        <xsl:sort select="ventas" data-type="number" order="descending"/>
                        <tr>
                            <td><xsl:value-of select="titulo"/></td>
                            <td><xsl:value-of select="ventas"/></td>
                        </tr>
                    </xsl:for-each>
                </table></body></html>
    </xsl:template>
</xsl:stylesheet>
```

```html
<html>
   <head>
      <meta charset="UTF-8"/>
      <title>Lista de discos de Daft Punk</title>
   </head>
   <body>
      <h3>Discografía de Daft Punk</h3>
      <table border="1" border-collapse="collapse">
         <tr>
            <th>Título</th>
            <th>Ventas</th>
         </tr>
         <tr>
            <td>Random Access Memories</td>
            <td>5000000</td>
         </tr>
         <tr>
            <td>Discovery</td>
            <td>3000000</td>
         </tr>
         <tr>
            <td>Homework</td>
            <td>2300000</td>
         </tr>
         <tr>
            <td>Human After All</td>
            <td>1200000</td>
         </tr>
      </table>
   </body>
</html>
```

Discografía de Daft Punk

Título	Ventas
Random Access Memories	5000000
Discovery	3000000
Homework	2300000
Human After All	1200000