El lenguaje XML

1. ¿Qué es XML?

XML son las siglas de *eXtended Markup Language*, o *lenguaje de marcado extendido*. Se concibió inicialmente como una extensión o ampliación del lenguaje originario SGML, y se emplea para estructurar la información utilizando un conjunto de etiquetas personalizable por el usuario.

Podríamos concebir XML también como un *metalenguaje*, es decir, un lenguaje que permite definir otros lenguajes, ya que a través de su sintaxis y sus normas, podemos definir el vocabulario y el conjunto de reglas sintácticas que necesitemos para nuestros usos particulares.

XML se propone como un estándar para intercambiar información estructurada entre distintas plataformas o sistemas. De este modo, podemos generar un fichero XML a partir de una aplicación de edición de imágenes, para luego importarlo en una página web, por ejemplo.

2. Estructura de un documento XML

La estructura básica de un documento XML comienza con una etiqueta que indica que, efectivamente, estamos utilizando dicho lenguaje, junto con su versión (típicamente la 1.0), y el formato de codificación del archivo, como por ejemplo, UTF-8.

Después suele haber una etiqueta principal que engloba todo el contenido del documento, y que se suele denominar etiqueta *raíz*. Dentro se tiene el resto de etiquetas, atributos e información que queramos almacenar. Aquí vemos un posible ejemplo de documento XML, basado en un fragmento de ejemplo previo.

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

<biblioteca>
    <libro>
        <titulo>El juego de Ender</titulo>
        <autor>Orson Scott Card</autor>
        <paginas>325</paginas>
        <fechaPublicacion anyo="1985" />
    </libro>
    <libro>
        <titulo>La tabla de Flandes</titulo>
        <autor nacimiento="1951">Arturo Pérez Reverte</autor>
        <paginas>384</paginas>
        <fechaPublicacion anyo="1990" />
    </libro>
</biblioteca>
```

En este caso, la etiqueta `biblioteca` es la etiqueta raíz que contiene al resto de elementos. Internamente, la información se divide en distintos bloques `libro`, con información relativa a diferentes libros. Notar que puede haber etiquetas de apertura y cierre (caso de `libro`, `titulo`, etc.) y también etiquetas sin contenido, y por tanto, sin etiqueta de cierre, como es el caso de `fechaPublicacion`. En este último caso, la información asociada a esa etiqueta únicamente puede establecerse por los atributos de la misma (atributo `anyo` en este caso).

> **Ejercicio 1**
>
> Define un archivo llamado `DAM.xml`. Copia dentro el siguiente contenido, que contiene información sobre los módulos del ciclo formativo de Desarrollo de Aplicaciones Multiplataforma.

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

<ciclo codigo="DAM">
    <modulo curso="1">
        <nombre>Programacion</nombre>
        <horas valor="256" />
    </modulo>
    <modulo curso="1">
        <nombre>Bases de Datos</nombre>
        <horas valor="160" />
    </modulo>
    <modulo curso="1">
        <nombre>Lenguajes de Marcas</nombre>
        <horas valor="96" />
    </modulo>
    <modulo curso="2">
        <nombre>Acceso a Datos</nombre>
        <horas valor="120" />
    </modulo>
</ciclo>
```

2.1. Los espacios de nombres o *namespaces*

En ocasiones nos puede interesar utilizar un conjunto de etiquetas externo, creado por terceras partes. Normalmente este conjunto de etiquetas conforma lo que se denomina un **espacio de nombres**, es decir, un grupo de nombres válidos, y debemos indicar en el propio documento qué espacio de nombres vamos a utilizar, y dónde encontrar su especificación.

Por ejemplo, más adelante utilizaremos un formato de XML específico llamado XSLT, para hacer transformaciones de documentos XML. Para poder utilizar las etiquetas específicas de XSLT, debemos indicar en el documento que vamos a utilizar ese espacio de nombres, de este modo:

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

    <xsl:template match="/">
        <xsl:apply-templates/>
    </xsl:template>
    ...
```

Como vemos, para poder utilizar un espacio de nombres determinado debemos indicar un prefijo (en este caso se indica el prefijo `xsl`) asociado al espacio de nombres deseado (`xmlns`), dado por una URL. En concreto, vamos a utilizar las etiquetas del espacio de nombres *http://www.w3.org/1999/XSL/Transform*. Esto

nos habilitará para después poder utilizar etiquetas como `xsl:template` o `xsl:apply-templates`, entre otras, correspondientes a ese espacio de nombres.

3. Validación de documentos XML

Una de las principales ventajas que aporta el uso del lenguaje XML para estructurar el contenido de nuestros documentos es que podemos verificar que la estructura del documento es sintácticamente correcta, y no hemos omitido ningún elemento necesario, ni ubicado ninguna etiqueta en un lugar que no le corresponde.

Existen distintos mecanismos que nos permiten verificar si un documento XML es válido. Principalmente, podemos hacer uso de las DTD (*Document Type Definition*, definición de tipo de documento) o a través de esquemas (*schemas*). Trataremos la primera de las opciones, por ser más simple.

3.1. Sintaxis de las DTD

Las DTD definen una serie de reglas que debe cumplir un documento XML para considerarse válido. Básicamente, mediante estas reglas comprobamos que las etiquetas o elementos que contiene el documento son correctas, están en un orden adecuado y almacenan información apropiada (a través de reglas `ELEMENT`) y que los atributos de las etiquetas también son correctos (a través de reglas `ATTLIST`).

Reglas de elementos (ELEMENT)

Comienzan con la palabra `ELEMENT` seguida del nombre de la etiqueta a la que hacemos referencia. Después, añadimos la siguiente información:

- Si contiene subetiquetas internas, se especifican entre paréntesis y separadas por comas (si pueden coexistir todas) o por barras verticales `|` (si son excluyentes). Para cada subetiqueta que especifiquemos, podemos emplear los siguientes comodines:
  - `*` para indicar que la subetiqueta puede aparecer 0 o más veces
  - `+` para indicar que la subetiqueta puede aparecer 1 o más veces
  - `?` para indicar que la subetiqueta puede aparecer 0 o 1 veces
- Si no contiene subetiquetas, sino directamente información, lo indicamos con `#PCDATA`.
- Si es una etiqueta sin contenido, lo indicamos con `EMPTY`.

Por ejemplo, estas serían las reglas `ELEMENT` para el ejemplo de biblioteca anterior:

```
<!ELEMENT biblioteca (libro*)>
<!ELEMENT libro (titulo, autor, paginas, fechaPublicacion)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT paginas (#PCDATA)>
<!ELEMENT fechaPublicacion EMPTY>
```

Reglas de atributos (ATTLIST)

Las reglas de atributos normalmente se colocan a continuación de las reglas de elemento (aunque el orden puede ser otro), y definen las características que deben cumplir los atributos de las etiquetas que definimos. Para cada regla `ATTLIST`, indicaremos primero el nombre de la etiqueta a la que hacemos referencia, seguido del nombre del atributo que queremos definir, y el tipo de dato que admite. Este último elemento puede ser:

- `CDATA` si admite texto en general (el valor más habitual)
- Un conjunto de valores limitado, separados por barras verticales `|` y entre paréntesis
- `ID` si el valor del atributo es único (no repetido en otros elementos del documento para el mismo atributo).
- `IDREF` si el valor del atributo hace referencia a otro atributo de tipo ID
- `IDREFS` si el valor del atributo es un conjunto de valores que hacen referencia a otro atributo de tipo ID. En este caso, el conjunto de valores se dan separados por espacios.

Además, la regla `ATTLIST` puede tener una serie de indicadores al final que dan información adicional sobre el atributo:

- `#REQUIRED` indica que es obligatorio indicar el atributo y darle un valor.
- `#IMPLIED` indica que no es obligatorio indicar el atributo
- `#FIXED` para indicar un valor prefijado por defecto (y sin posibilidad de cambiarlo).
- También podemos darle un valor por defecto en caso de que no se le asigne ninguno.

Por ejemplo, para el atributo `anyo` de la etiqueta `fechaPublicacion` en el ejemplo anterior, podemos indicar que es obligatorio indicarlo de este modo:

```
<!ATTLIST fechaPublicacion anyo CDATA #REQUIRED>
```

El atributo `nacimiento` de la etiqueta `autor` es opcional, ya que algunos autores no lo tienen. Podemos indicarlo así, y además, podemos darle un valor por defecto en el caso de que no se especifique

```
<!ATTLIST autor nacimiento CDATA "No especificado">
```

3.2. Ubicación de las reglas DTD

Podemos definir este conjunto de reglas de la DTD tanto embebidas en el propio documento XML, como en un fichero aparte referenciado desde el XML. Si optamos por la primera opción, basta con que añadamos las reglas dentro de una etiqueta `DOCTYPE`, antes del contenido del documento. Por ejemplo:

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

<!DOCTYPE biblioteca[
    <!ELEMENT biblioteca (libro*)>
    <!ELEMENT libro (titulo, autor, paginas, fechaPublicacion)>
    <!ELEMENT titulo (#PCDATA)>
    <!ELEMENT autor (#PCDATA)>
    <!ELEMENT paginas (#PCDATA)>
    <!ELEMENT fechaPublicacion EMPTY>
    <!ATTLIST fechaPublicacion anyo CDATA #REQUIRED>
    <!ATTLIST autor nacimiento CDATA "No especificado">
]>

<biblioteca>
    <libro>
        <titulo>El juego de Ender</titulo>
        <autor>Orson Scott Card</autor>
        <paginas>325</paginas>
        <fechaPublicacion anyo="1985" />
    </libro>
    ...
```

Si queremos dejar las reglas en un fichero aparte (algo que puede resultar útil para validar con ellas diferentes documentos con la misma sintaxis), entonces dejamos las reglas tal cual en un archivo (típicamente con extensión *.dtd*), y referenciamos el archivo desde el documento XML:

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

<!DOCTYPE biblioteca SYSTEM "fichero.dtd">

<biblioteca>
    <libro>
    ...
```

3.3. Validación de documentos

Existen algunas herramientas que se pueden instalar de forma offline para validar documentos XML, tales como *xmllint*, o *XML Toolkit*, y alguna extensión para Visual Studio Code como XML de RedHat. Sin embargo, para usos más esporádicos puede resultar útil acudir a algún validador online como éste. Simplemente copiamos y pegamos, o adjuntamos el archivo con el DTD incorporado, y nos indicará el resultado de la validación.

> **Ejercicio 2**

Define unas reglas DTD sobre el fichero anterior `DAM.xml` para comprobar que sea válido. En concreto, debes comprobar que cada etiqueta contenga las subetiquetas permitidas con los valores indicados en el ejemplo. En cuanto a los atributos, configúralos de este modo:

- El atributo `codigo` de la etiqueta `ciclo` es opcional. Si no se pone, se asume el valor de "DAM"
- El atributo `curso` de la etiqueta `modulo` es obligatorio, y sólo puede valer 1 o 2.
- El atributo `valor` de la etiqueta `horas` es obligatorio.

Una vez tengas las reglas DTD definidas en el propio archivo XML, comprueba desde la herramienta online explicada antes que todo es correcto.

Ejercicio 3

Dado el siguiente documento XML:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<informacion>
    <software codigo="N1" tipo="gratuito">
        <nombre>Notepad++</nombre>
        <fechaPubli ano="2015" mes="marzo"/>
    </software>
    <software codigo="X1">
        <nombre>XML Copy Editor</nombre>
        <fechaPubli ano="2012" mes="mayo"/>
    </software>
    <software codigo="M1" tipo="comercial">
        <nombre>Microsoft Word</nombre>
    </software>
    <software codigo="P1">
        <nombre>PacketTracer</nombre>
        <fechaPubli ano="2016" mes="enero"/>
    </software>
    <modulo usa="N1 X1">
        <titulo>Lenguajes de Marcas</titulo>
    </modulo>
    <modulo usa="P1">
        <titulo>Redes</titulo>
    </modulo>
    <modulo>
        <titulo>FOL</titulo>
    </modulo>
</informacion>
```

Crea un DTD que lo valide teniendo en cuenta lo siguiente:

- Siempre existirán elementos `software` y `modulo`
- El elemento `informacion` tendrá un atributo llamado `curso` que siempre tendrá el valor 1

- El `codigo` será único y el `tipo` podrá tomar los valores *gratuito* y *comercial*
- La fecha de publicación tendrá por defecto el año 2015, y siempre indicará el mes
- Debemos validar que los módulos usen el software existente en el documento XML

4. Transformación de documentos XML con XSLT

Otra de las ventajas que ofrece el trabajar con documentos XML es que podemos adaptar la información que contiene y exportarla a distintos formatos. Por ejemplo, podemos generar una página web HTML o xHTML con la información contenida en el documento, y también generar un archivo PDF, todo desde los mismos datos de origen. Para ello haremos uso del lenguaje XSLT.

XSLT son las siglas de *eXtensible Stylesheet Language Transformations*, y permite definir una serie de reglas para transformar el contenido de un documento XML a un formato determinado. Para ello, haremos uso de las etiquetas propias de dicho lenguaje, que incorporaremos a partir de su espacio de nombres (*xmlns*).

4.1. Estructura básica de los archivos XSLT

Los archivos XSL se suelen almacenar como ficheros de texto con extensión *.xslt*. Dentro, definimos el archivo como un archivo XML (misma cabecera que los archivos XML normales) y acto seguido incorporamos el espacio de nombres de XSLT. Esta etiqueta que incorpora el espacio de nombres es la que hace de raíz del documento XLST. El resto de reglas de transformación las colocaremos dentro de esta etiqueta:

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

</xsl:stylesheet>
```

Dentro de este elemento contenedor, vamos a ir definiendo reglas que van a permitir procesar cada una de las etiquetas del documento XML y saber qué hacer con ellas. Para ello haremos uso de las etiquetas que veremos a continuación.

4.2. Primeras etiquetas básicas: *template*, *value-of* y *apply-templates*

template

Esta etiqueta permite aplicar la transformación a la etiqueta que coincida con la que pongamos en su atributo `match`. Si queremos hacer referencia a la etiqueta raíz, podemos referenciarla por su nombre, o directamente por `/`.

```
<xsl:template match="/">
    ...
</xsl:template>

<xsl:template match="libro">
    ...
</xsl:template>
```

Internamente, podemos directamente generar el contenido en el formato que indiquemos (por ejemplo, definir etiquetas HTML) o hacer uso de otras etiquetas XSLT para seguir procesando información del documento XML. Veremos después algunos ejemplos.

El proceso debe comenzar siempre por la etiqueta principal o raíz. Indicaremos dentro de la correspondiente etiqueta *xsl:template* qué queremos hacer con ella, y progresivamente podemos hacer que vaya profundizando y analizando el resto de etiquetas, como veremos a continuación.

value-of

Esta etiqueta se emplea para obtener el valor de una subetiqueta o atributo perteneciente a la etiqueta en la que estamos. Si por ejemplo queremos sacar el título del libro en el que estamos, podemos hacer algo así:

```
<xsl:template match="libro">
    <xsl:value-of select="titulo" />
</xsl:template>
```

Si queremos sacar el año de publicación del libro, haríamos lo siguiente (desde la etiqueta `fechaPublicacion`):

```
<xsl:template match="fechaPublicacion">
    <xsl:value-of select="@anyo" />
</xsl:template>
```

También podemos utilizarla para mostrar el valor de la etiqueta actual. Por ejemplo, el valor del título del libro en el que estamos (en el caso de que no lo queramos sacar desde la etiqueta contenedora *libro*):

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

Notar que, en el caso de los atributos, se referencian precedidos de una arroba `@`.

apply-templates

Se emplea cuando dentro de una etiqueta hay otras subetiquetas que queremos seguir procesando automáticamente. Esta etiqueta hace que se "invoquen" las instrucciones *template* asociadas a las subetiquetas en cuestión. Por ejemplo si desde dentro de un libro queremos seguir procesando las subetiquetas que contiene, haríamos algo así:

```
<xsl:template match="libro">
    <xsl:apply-templates />
</xsl:template>
```

Ejemplo

El siguiente ejemplo muestra cómo podríamos generar un documento HTML con un listado de libros, utilizando la información del documento XML original visto anteriormente:

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

    <xsl:template match="/">
        <html>
            <body>
                <h1>Listado de libros</h1>
                <table>
                    <tr>
                        <th>Título</th>
                        <th>Autor</th>
                        <th>Páginas</th>
                        <th>Año</th>
                    </tr>

                    <xsl:apply-templates />

                </table>
            </body>
        </html>
    </xsl:template>

    <xsl:template match="libro">
        <tr>
            <td><xsl:value-of select="titulo" /></td>
            <td><xsl:value-of select="autor" /></td>
            <td><xsl:value-of select="paginas" /></td>
            <xsl:apply-templates />
        </tr>
    </xsl:template>

    <xsl:template match="titulo"></xsl:template>
    <xsl:template match="autor"></xsl:template>
    <xsl:template match="paginas"></xsl:template>

    <xsl:template match="fechaPublicacion">
        <td><xsl:value-of select="@anyo" /></td>
    </xsl:template>

</xsl:stylesheet>
```

Expliquemos los pasos que hemos seguido:

- La primera etiqueta `template` hace referencia a la etiqueta raíz del documento XML. Cuando la encontremos, montaremos toda la estructura del documento HTML resultado: el *DOCTYPE*, el cuerpo del documento, y la estructura de la tabla que contendrá el listado de libros. Para rellenar las filas de la tabla,

utilizamos la etiqueta `apply-templates` para que se procese cada libro y genere su propia información

- Para el *template* de *libro*, construimos una fila de la tabla, mostrando con `value-of` el título, autor y número de páginas. Como el año de publicación no es un dato al que podamos acceder directamente desde el libro (porque es un atributo de la subetiqueta `fechaPublicacion`), aplicamos plantillas de nuevo con `apply-templates` para ir un nivel más hacia dentro
- Hemos definido unas plantillas vacías para `titulo`, `autor` y `paginas`. De lo contrario, la llamada a `apply-templates` anterior volvería a mostrar la información de estas etiquetas, que ya hemos procesado con `value-of`.
- Finalmente, definimos la plantilla para `fechaPublicacion` y mostramos en una columna el valor de su atributo `anyo`.

Alternativamente, podríamos haber empleado esta otra hoja XSLT con el mismo resultado:

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

    <xsl:template match="/">
        <html>
            <body>
                <h1>Listado de libros</h1>
                <table>
                    <tr>
                        <th>Título</th>
                        <th>Autor</th>
                        <th>Páginas</th>
                        <th>Año</th>
                    </tr>

                    <xsl:apply-templates />

                </table>
            </body>
        </html>
    </xsl:template>

    <xsl:template match="libro">
        <tr>
            <xsl:apply-templates />
        </tr>
    </xsl:template>

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

    <xsl:template match="autor">
        <td><xsl:value-of select="." /></td>
    </xsl:template>

    <xsl:template match="paginas">
        <td><xsl:value-of select="." /></td>
    </xsl:template>

    <xsl:template match="fechaPublicacion">
        <td><xsl:value-of select="@anyo" /></td>
    </xsl:template>

</xsl:stylesheet>
```

En este caso, como antes, comenzamos con la etiqueta raíz que construye la estructura HTML general con la tabla. Después, para cada fila de la tabla llamamos a `apply-templates` para que genere su contenido. Esto va a procesar cada libro (template *libro*), que a su vez va a procesar cada subetiqueta de libro (*titulo*, *autor*, *paginas* y *fechaPublicacion*, a través de sus respectivas *templates*). Observad que en estas etiquetas finales simplemente se muestra la celda (*td*) con la informacion correspondiente de cada etiqueta.

4.3. Prueba de la transformación

Para probar la transformación de la hoja XSLT sobre el documento XML, necesitamos pasar ambos ficheros a una herramienta de transformación. Proponemos dos alternativas:

- Utilizar la extensión xslt-transform de Visual Studio Code. Necesitaremos tener instalada la librería Saxon para Java (podéis descargar una versión aquí), y en los *Settings* de la extensión configurar la propiedad *Xslt:processor* para apuntar a la carpeta donde esté el archivo *.jar* con la librería. Por ejemplo, *C:\SaxonHE10-5J\saxon-he-10.5.jar*. Después, para aplicar la transformación debemos estar editando el archivo XML, y con *Ctrl+Mayus+P* abrir el panel de comandos y elegir *XSLT: Run Transformation*. Después, elegiremos el archivo XSL que queremos aplicar y generará un nuevo documento con la transformación aplicada, que podemos guardar donde queramos.
- Como segunda alternativa, podemos utilizar herramientas online como esta, a la que podemos o bien copiar y pegar el código de ambos ficheros (el XML y el XSLT) o bien subirlos directamente, y obtendremos en un cuadro de texto el resultado de la transformación.

> **Ejercicio 4**
>
> Define una hoja `ciclos.xslt` que permita obtener una lista no ordenada con los módulos del archivo `DAM.xml` creado anteriormente. En cada *item* de la lista, mostraremos el nombre del módulo y, entre paréntesis, las horas que lo componen. Quedará algo así:
>
> - Programación (256 horas)
> - Bases de Datos (160 horas)
> - Lenguajes de Marcas (96 horas)
> - Acceso a Datos (120 horas)

4.4. Otras etiquetas adicionales

Veremos a continuación otras etiquetas que podemos aplicar para generar contenido algo más específico en algunos casos.

attribute

Esta etiqueta genera un atributo con el nombre indicado en el contenido de salida. Por ejemplo, esta expresión genera la etiqueta `<etiqueta numero="1">`

```
<etiqueta>
    <xsl:attribute name="numero">1</xsl:attribute>
</etiqueta>
```

variable

Esta etiqueta permite definir variables que almacenen temporalmente valores de ciertas etiquetas o atributos. El siguiente ejemplo almacena el valor del atributo *attr1* de la etiqueta actual dentro de la variable *var1*

```
<xsl:variable name="var1" select="@attr1" />
```

if

La etiqueta *if* permite ejecutar un conjunto de elementos si se cumple la condición indicada en su atributo *test*. Podemos, por ejemplo, comprobar si el valor de un atributo es igual a un cierto valor, o mayor que un valor, o directamente, si tiene valor:

```
<xsl:if test="@atributo">
    <!-- El atributo tiene valor -->
</xsl:if>

<xsl:if test="@atributo='es'">
    <!-- El atributo tiene valor 'es' -->
</xsl:if>

<xsl:if test="@atributo &gt; 1">
    <!-- El atributo tiene valor mayor que 1 -->
</xsl:if>
```

choose/when/otherwise

La combinación de estas etiquetas permiten hacer algo parecido a lo que sería un *if..else* en un lenguaje de programación. Si la condición indicada en el atributo *test* de la etiqueta *when* es cierta, se ejecuta el contenido de esa etiqueta. De lo contrario, se ejecuta el contenido de la etiqueta *otherwise*.

```
<xsl:choose>
    <xsl:when test="@atributo='es'">
        <!-- El atributo vale 'es' -->
    </xsl:when>
    <xsl:otherwise>
        <!-- El atributo vale otra cosa -->
    </xsl:otherwise>
</xsl:choose>
```

for-each

Esta etiqueta permite iterar sobre un conjunto de etiquetas. Por ejemplo, este código itera sobre todas las etiquetas *prueba* de un archivo XML:

```
<xsl:for-each test="prueba">
    <!-- Aquí podemos acceder a atributos de la etiqueta, por ejemplo -->
</xsl:for-each>
```

También podemos hacer uso de ciertas funciones predefinidas, como `count`, que nos permite contar cuántos elementos cumplen un determinado criterio. Por ejemplo, esta expresión nos podría servir para mostrar cuántos pasajeros de un archivo XML son adultos:

```
<xsl:value-of select="count(pasajero[@adulto='si'])" />
```

Existen otras muchas etiquetas y funciones disponibles, pero estas nos servirán para hacernos una idea de lo que se puede hacer.

> **Ejercicio 5**
>
> Dado el documento XML `rusia2018.xml` con algunos de los equipos y partidos de fútbol jugados en el mundial de Rusia 2018...

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rusia2018>
    <equipos>
        <equipo grupo="B" nombre="ESP">España</equipo>
        <equipo grupo="A" nombre="ASA">Arabia Saudí</equipo>
        <equipo grupo="B" nombre="POR">Portugal</equipo>
        <equipo grupo="A" nombre="RUS">Rusia</equipo>
    </equipos>
    <partidos jornada="1">
        <partido equi1="RUS" equi2="ASA">
            <gol nombre="Golovin" equipo="RUS">94</gol>
            <amarilla nombre="Al-Jassim" equipo="ASA">93</amarilla>
            <gol nombre="Cheryshev" equipo="RUS">91</gol>
            <gol nombre="Dzyuba" equipo="RUS">71</gol>
            <gol nombre="Cheryshev" equipo="RUS">43</gol>
            <gol nombre="Gazinsky" equipo="RUS">12</gol>
        </partido>
        <partido equi1="POR" equi2="ESP">
            <gol nombre="Ronaldo" equipo="POR">4</gol>
            <amarilla nombre="Busquets" equipo="ESP">17</amarilla>
            <gol nombre="Costa" equipo="ESP">24</gol>
            <amarilla nombre="Fernandes" equipo="POR">28</amarilla>
            <gol nombre="Ronaldo" equipo="POR">44</gol>
            <gol nombre="Costa" equipo="ESP">55</gol>
            <gol nombre="Nacho" equipo="ESP">58</gol>
            <gol nombre="Ronaldo" equipo="POR">8</gol>
        </partido>
    </partidos>
</rusia2018>
```

... se solicita la escritura del documento `Rusia.xsl` que realice la transformación del mismo documento en otro XML `resultados.xml` donde se puedan ver los resultados de los partidos tal y como se muestra a continuación:

```
<?xml version="1.0" encoding="UTF-8"?>
<resultados>
    <jornada numero="1">
        <partido>
            <selecciones>RUS-ASA</selecciones>
            <resultado>5-0</resultado>
        </partido>
        <partido>
            <selecciones>POR-ESP</selecciones>
            <resultado>3-3</resultado>
        </partido>
    </jornada>
</resultados>
```

5. Usos de XML

Entre los principales usos que se dan hoy en día de los documentos y el formato XML, podemos destacar los siguientes:

- **Almacenamiento de información estructurada**. Como hemos visto, XML proporciona una forma estructurada y formal de almacenar la información, definiendo una serie de etiquetas y atributos, y garantizando que éstas van a estar dispuestas en un cierto orden y cantidad. Esto, unido a ciertas aplicaciones y librerías específicas que son capaces de procesar el contenido de ficheros XML y extraer la información, convierten a los documentos XML en valiosas "bases de datos" sencillas, que podemos emplear en diferentes tipos de aplicaciones.
- **Configuración de aplicaciones**. Existen ciertos subtipos de formato XML que sirven para definir la configuración inicial de ciertos tipos de aplicaciones. Por ejemplo, el formato SVG, empleado para definir imágenes vectoriales, utiliza nomenclatura XML para definir los elementos de la imagen. El formato FXML, utilizado en aplicaciones JavaFX para definir los diferentes elementos gráficos que va a tener la aplicación, también es un subtipo de XML. Aquí podemos ver un fragmento de un archivo FXML de ejemplo:

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

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<BorderPane maxHeight="-Infinity" maxWidth="-Infinity"
 minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0"
 prefWidth="600.0" xmlns="http://javafx.com/javafx/11.0.1"
 xmlns:fx="http://javafx.com/fxml/1" fx:controller="sample.Controller">
   <left>
      <ListView fx:id="listGames" onMouseClicked="#updateForm"
      prefHeight="400.0" prefWidth="170.0" BorderPane.alignment="CENTER" />
   </left>
   <center>
      <AnchorPane prefHeight="200.0" prefWidth="200.0" BorderPane.
      alignment="CENTER">
         <children>
            <Label alignment="CENTER" layoutX="162.0" layoutY="55.0"
            prefHeight="17.0" prefWidth="105.0" text="Title" />
            <Label alignment="CENTER" layoutX="162.0" layoutY="135.0"
            prefHeight="17.0" prefWidth="105.0" text="Price" />
            <TextField fx:id="txtTitle" layoutX="140.0" layoutY="90.0" />
            <TextField fx:id="txtPrice" layoutX="140.0" layoutY="165.0" />
            <Button layoutX="106.0" layoutY="221.0" mnemonicParsing="false"
            onAction="#addVideoGame" prefHeight="25.0" prefWidth="67.0"
            text="Add" />
            <Button layoutX="200.0" layoutY="221.0" mnemonicParsing="false"
            onAction="#deleteVideoGame" text="Delete" />
            <Button layoutX="281.0" layoutY="221.0" mnemonicParsing="false"
            onAction="#updateVideoGame" text="Update" />
         </children>
      </AnchorPane>
   </center>
</BorderPane>
```

- **Serialización de datos**. Podemos almacenar información en formato XML para ser enviada a través de algún canal de comunicación. Por ejemplo, el formato SOAP es muy utilizado en servicios web para enviar datos entre clientes y servidores, de forma remota.
- etc.