

UD 01 - ASPECTOS BÁSICOS DE LOS LENGUAJES DE MARCAS

1. Definición y clasificación de lenguajes de marcas

Los lenguajes de marcas sirven para **codificar un documento** donde, junto con el texto, se incorporan etiquetas, marcas o anotaciones con información adicional sobre la estructura del texto o su forma de presentarlo.

El lenguaje de marcas especifica cuáles son las etiquetas posibles, dónde deben colocarse y el significado que tendrá cada una de ellas.

Puede explicitar la estructura del documento, su contenido semántico o cualquier información que quiera hacerse presente.

Los lenguajes de marcas están definidos en un documento (DTD) en el que se establecen las marcas, los elementos usados por el lenguaje, sus etiquetas, atributos, sintaxis y normas de uso.

Usos:

- Dar formato a los documentos de texto
- Definir la estructura de datos de un documento
- Intercambio de ficheros entre aplicaciones y plataformas

Los lenguajes de marcas pueden clasificarse en tres grupos:

- **Orientados a presentación:** Usados por los procesadores de textos y codifican cómo ha de presentarse el documento. Ej.: Palabra en negrita, espacio entre caracteres. Las marcas se ocultan al usuario (WYSIWYG)
- **De procedimientos:** Orientados a presentación pero dentro de un marco que permite definir macros (secuencias o acciones). El programa debe interpretar el código en el mismo orden en el que aparece (TeX, LaTeX, Postscript)
- **Descriptivos o semánticos:** Indican qué es esa información. No especifica cómo deben representarse.

Ejemplos de lenguajes de marcado

- Documentación electrónica:
 - RTX** Formato de texto enriquecido. Intercambio de documentos de texto entre procesadores de texto.
 - TeX**. Ecuaciones matemáticas
 - Wikitexto**: Páginas wiki en servidores preparados
 - DocBook**: Generar documentos separando la estructura lógica del documento de su formato.
- Tecnologías de internet
 - HTML, XHTML**. Creación de webs.
 - RSS**. Difusión de contenidos web
- Otros
 - **MathML**: Formalismo matemático para ser entendido por sistemas y aplicaciones
 - **VoiceXML**: Intercambio de información entre usuario y app de reconocimiento de voz
 - **MusicXML**: Intercambio de partitura entre editores de partituras

2. Evolución de los lenguajes de marcas

Los lenguajes de marcas surgen a finales de los 60 para introducir anotaciones en documentos electrónicos (lenguajes informáticos distintos de los lenguajes de programación y orientados a la gestión de información)

También surgen los lenguajes y sistemas de bases de datos, lenguajes orientados a representación, almacenamiento y consulta de grandes cantidades de datos.

Inicialmente LLMM surgen como lenguajes formados por códigos de formato que los procesadores de texto introducen para dirigir el proceso de presentación mediante una impresora. Ligados a las características de máquina, programa, procesador concreto... (Idea de marca de negrita, subrayado...)

Después, medio de presentación a la pantalla. Es suficiente con pulsar combinación de teclas o botones para los resultados requeridos.

GML y SSGML

Para resolver el problema de falta de estandarización en los formatos de información usados por los programas.

IBM construcción de sistema de edición, almacenamiento y búsqueda de documentos legales.

Necesidad de un formato estándar. Válido para los distintos tipos de documentos. --> Se crea GML para describir los documentos de forma independiente a la plataforma y aplicación. --> Evoluciona hasta SGML (ISO 8879). Era complejo y requería de herramientas caras, relegado a aplicaciones industriales.

HTML

1989/1990. Tim Berners-Lee crea World Wide Web y, conociendo SGML, se encuentra con la necesidad de organizar, enlazar y compatibilizar gran cantidad de información. A partir de SGML creó HTML, combinación de los dos estándares existentes:

- ASCII: Formato que permite transferencia de datos entre ordenadores al ser fácilmente interpretable
- SGML: Permite dar estructura al texto, resaltando títulos o aplicando formato

HTML es hoy día el tipo de documento más empleado del mundo. Gran sencillez. Pero caos debido a su facilidad de uso, provocando muchos documentos pero mal estructurados y sin respetar estándares.

XML

En respuesta, aparece XML (1998), lenguaje de marcas estructural sin información de diseño que permite crear etiquetas adaptadas a necesidades (extensible).

Es un metalenguaje y permite:

- Definir etiquetas propias
 - Asignar atributos a etiquetas
 - Usar esquema para definir de forma exacta etiquetas y atributos
 - Estructura independiente del diseño
-
- **XSL**, eXtensible Style Language. Permite definir hojas de estilo para los documentos XML e incluye capacidad para la transformación de documentos.
 - **XML Linking Language**, incluye Xpath, Xlink y Xpointer. Determinan aspectos sobre los enlaces entre documentos XML.
 - **XML Namespaces**. Proveen un contexto al que se aplican las marcas de un documento de XML y que sirve para diferenciarlas de otras con idéntico nombre válidas en otros contextos.
 - **XML Schemas**. Permiten definir restricciones que se aplicarán a un documento XML. Actualmente los más usados son las DTD.

Comparativa XML - SGML

XML	SGML
<ul style="list-style-type: none"> Su uso es sencillo. 	<ul style="list-style-type: none"> Su uso es muy complejo.
<ul style="list-style-type: none"> Trabaja con <u>documentos bien formados</u>, no exige que estén validados. 	<ul style="list-style-type: none"> Sólo trabaja con <u>documentos válidos</u>.
<ul style="list-style-type: none"> Facilita el desarrollo de aplicaciones de bajo coste. 	<ul style="list-style-type: none"> Su complejidad hace que las aplicaciones informáticas para procesar SGML sean muy costosas.
<ul style="list-style-type: none"> Es muy utilizado en informática y en más áreas de aplicación. 	<ul style="list-style-type: none"> Sólo se utiliza en sectores muy específicos.
<ul style="list-style-type: none"> Compatibilidad e integración con HTML. 	<ul style="list-style-type: none"> No hay una compatibilidad con HTML definida.
<ul style="list-style-type: none"> Formateo y estilos fáciles de aplicar. 	<ul style="list-style-type: none"> Formateo y estilos relativamente complejos.
<ul style="list-style-type: none"> No usa etiquetas opcionales. 	

Comparativa XML-HTML

XML	HTML
<ul style="list-style-type: none"> Es un perfil de SGML. 	<ul style="list-style-type: none"> Es una aplicación de SGML.
<ul style="list-style-type: none"> Especifica cómo deben definirse conjuntos de etiquetas aplicables a un tipo de documento. 	<ul style="list-style-type: none"> Aplica un conjunto limitado de etiquetas sobre un único tipo de documento.
<ul style="list-style-type: none"> Modelo de <u>hiperenlaces complejo</u>. 	<ul style="list-style-type: none"> Modelo de hiperenlaces simple.
<ul style="list-style-type: none"> El navegador es una plataforma para el desarrollo de aplicaciones. 	<ul style="list-style-type: none"> El navegador es un visor de páginas.
<ul style="list-style-type: none"> Fin de la guerra de los navegadores y etiquetas propietarias. 	<ul style="list-style-type: none"> El problema de la 'no compatibilidad' y las diferencias entre navegadores ha alcanzado un punto en el que la solución es difícil.

3. Etiquetas elementos y atributos

- Etiquetas (tag):** Encerradas entre "<" y ">". Normalmente una de inicio y otra de fin.
- Elementos:** Se organiza el contenido del documento o acciones que se desencadenan. Tiene etiqueta de inicio, de fin y lo que hay entre ambas.
- Atributos:** Par nombre-valor. Dentro de la etiqueta de inicio del elemento. Indica propiedades que deben llevar asociados.

4. Herramientas de edición

Para trabajar en XML es necesario editar documentos y luego procesarlos.

"Un blog de notas y un navegador"

Editores XML

Basta un editor de texto para construir documento XML (texto plano). Importante que no formatee el texto para evitar código basura.

Para más complejos usar software de edición XML. Ayudará a crear estructuras, etiquetas... DTD, CSS, XSL... Amaya, Notepad++, Sublime Text, Netbeans...

Procesadores XML

Permite leer los documentos XML y acceder a contenido y estructura. Es un conjunto de módulos de

software entre los que se encuentra un parser o analizador de XML, que comprueba que se cumplen las normas establecidas. Puede ser necesario que sea válido (validadores) o solo que esté bien formado. (no validadores). Siguen recomendaciones de W3C.

Para publicar XML en internet se usan procesadores XSLT para generar HTML a partir de XML. Puesto que XML se usa para intercambio de datos se puede recurrir a motores que se ejecutan sin que nos demos cuenta (XML para Java de IBM, JAXP de Sun...) Puede usarse cualquier navegador.

UD 02 - UTILIZACIÓN DE LENGUAJES DE MARCAS EN ENTORNOS WEB (Parte 1 - HTML)

1. Del HTML al XHTML. Evolución y versiones

HTML es el lenguaje utilizado para crear la mayor parte de las páginas web. Es un estándar reconocido en todos los navegadores.

Evolución de sus versiones:

Origen: Sistema de hipertexto para compartir documentos electrónicos en 1980. La primera propuesta para convertir HTML en estándar se realizó en 1993. (Dos propuestas HTML y HTML+)

HTML 2.0 Primera versión oficial de HTML

HTML 3.2. Incorpora las applets de Java y texto alrededor de imágenes. 1997.

HTML 4.0. Hojas de estilos CSS y pequeños programas en webs. 1998.

HTML 4.01. Se detiene la actividad de estandarización hasta 2007.

Tras el estándar HTML 4.01 se detecta su incompatibilidad en XML y se crea el XHTML que combina la sintaxis de HTML 4.0 con la de XML.

XHTML 1.0. Primera versión (2000). Adaptación de HTML 4.01 añadiendo restricciones XML.

XHTML 1.1. Modularizar XHTML

Borrador XHTML 2.0.

HTML 5.0. Especificación actual de HTML. (Y sigue evolucionando...)

1.1. HTML5

Publicado en 2014. Especifica dos variantes de sintaxis para HTML: La variante HTML 5 y una variante conocida como sintaxis XHTML5 que debe ser servida como XML.

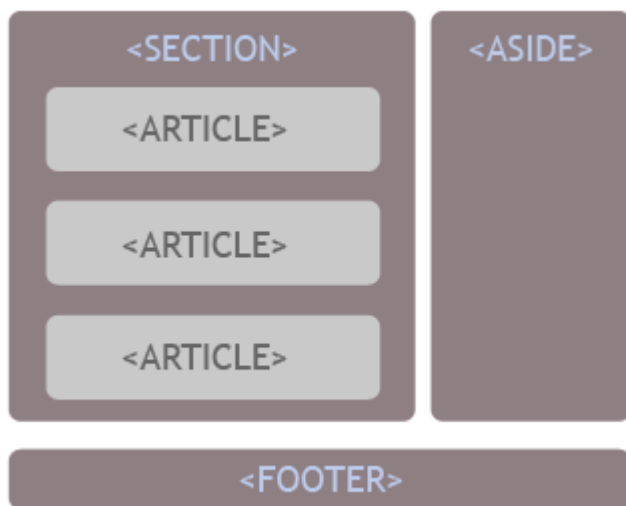
HTML5 ofrece características para desarrollo de páginas web más simples y para formas más complejas de manejo. La especificación HTML 5 se basa en XHTML 4.01 Strict pero no usa DTD.

Usa como base el Modelo de Objetos del Documento (el DOM, árbol creado por la estructura del documento) en vez de un conjunto determinado de reglas sintácticas.

En esta especificación se incluyen instrucciones detalladas de cómo los navegadores deben usar el HTML mal especificado.

Novedades principales:

- **Archivos multimedia:** Soporte integrado para el contenido multimedia gracias a los elementos `<audio>` y `<video>` ofreciendo la posibilidad de incluir contenido multimedia en HTML de forma nativa. El elemento `<canvas>` puede ser usado para dibujar gráficos.
- **Web semántica:** Se introducen etiquetas como `header`, `nav`, `section`, `aside`, `footer`, `article`, `hgroup`, `figure`... para hacer más descriptivo el lenguaje. Así el desarrollador puede definir mejor la importancia o finalidad de cada parte de la web y el código es más entendible para los buscadores.
- Los formularios también han sido mejorados con validaciones sin necesidad de Javascript, atributos nuevos y gran cantidad de elementos semánticos.



2. Estructura de un documento HTML

La estructura de HTML tiene un prólogo y un ejemplar.

Prólogo

Declaración del tipo de documento en la que se indica el tipo de documento a iniciar y la versión HTML utilizada para que lo interprete correctamente el navegador.

En HTML 4.0 hay tres prólogos distintos:

HTML 4.0 Strict. DTD usada por defecto en HTML 4.0. No se permiten los elementos declarados como deprecated por otras versiones o recomendaciones.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 //EN" "http://www.w3.org/TR/REC-html40/strict.dtd">
```

HTML 4.0 Transitional. Permite el uso de todos los elementos de HTML 4.0 Strict y los deprecated.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 transitional//EN" "http://www.w3.org/TR/REC-html40/loose.dtd">
```

HTML 4.0 Frameset. Variante de HTML 4.0 Transitional para documentos que usan frames. En ellos el body es reemplazado por un frameset.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN" "http://www.w3.org/TR/REC-html40/frameset.dtd">
```

En **HTML 5.0** se simplifica el prólogo enormemente:

```
<!DOCTYPE html>
```

Ejemplar

Está delimitado por las etiquetas `<html>` y `</html>`.

El ejemplar se divide en:

- **Cabecera.** Delimitada por `<head>` `</head>`. Información sobre el título de la página, el autor, palabras clave. Es obligatorio definir el título del documento con `<title>` `</title>`. En la cabecera puede utilizarse también el elemento `<META>` que puede forzar a que la página activa se cargue cada cierto tiempo (indicado en segundos mediante el atributo CONTENT). Debe usarse con precaución porque puede sobrecargar el servidor.

```
<HEAD>
  <TITLE> Título de la página </TITLE>
  <META HTTP-EQUIV="Refresh" CONTENT="10">
</HEAD> ``
```

El elemento `BASE HREF="URL"` define la información a prefijar a todo URL incompleto en el documento. Por ejemplo, el URL relativo `"/html/test.html"` corresponderá de hecho a `"URL/html/test.html"`.

- **Cuerpo**: Contiene información que se va a presentar en pantalla. Tiene las etiquetas `<body>` y `</body>` salvo en HTML 4.0 Frameset donde se sustituyen por `<frameset>` y `</frameset>`

3. Identificación de etiquetas y atributos de HTML

- El documento HTML tiene etiquetas y atributos.
- La cantidad de etiquetas de HTML **está limitada** a las existentes en el lenguaje (a diferencia de XML).
- La información adicional se añade en los atributos (que tienen un conjunto de valores definidos).
- Si no es un valor válido, el navegador lo ignora. Muchos de los atributos son comunes a muchas etiquetas.

3.1. Clasificación de los atributos comunes según su funcionalidad

Podemos distinguir:

Atributos básicos: Se pueden usar en casi todas las etiquetas HTML

`name`: Asignar nombre al elemento HTML

`title`: Asignar título al elemento HTML, mejorando la accesibilidad (navegadores lo muestran cuando se pasa por encima)

`id`: Identificar al elemento HTML de forma única

`style`: Aplicar estilo al elemento HTML

`class`: Aplicar al elemento HTML definido en CSS.

Los atributos `id` y `class` solo pueden contener guiones medios, guiones bajos, letras o números pero no pueden empezar por números. Se distingue entre mayúsculas y minúsculas., No se recomiendan caracteres no ingleses, acentos...

Atributos para internacionalización: Para webs multiidioma

`dir`: Dirección del texto (ltr, de izquierda a derecha valor por defecto; rtl, de derecha a izquierda)

`lang`: Código de idioma.

`xml:lang`: Código según recomendación RFC 1766

En XHTML `xml:lang` tiene más prioridad que `lang` y es obligatorio incluirlo siempre que se incluya `lang`

Atributos de eventos y atributos para los elementos que pueden obtener el foco: No lo cubre la asignatura.

3. Elementos HTML

El elemento HTML está formado por etiqueta de apertura, cero o más atributos, texto encerrado por la etiqueta (opcional), etiqueta de cierre.

Podemos distinguir entre:

- **Elemento en línea**: Solo ocupa el espacio necesario para mostrar sus contenidos (pueden ser texto u otros elementos en línea). Ej.: Enlaces, texto en negrita.
- **Elemento de bloque**: Siempre empiezan en una nueva línea y ocupan todo el espacio disponible hasta el final de la línea aunque sus contenidos no lleguen allí. El contenido puede ser texto, elementos en línea u otros elementos de bloque. Ej.: Encabezados, párrafos

Hay elementos que pueden tener un comportamiento en línea o en bloque según las circunstancias.

3.1. Estructura básica del documento

`html`

`head`

`body`: Permite definir formatos del documento que se aplican a los elementos de la página de forma global (color del fondo, márgenes, color de enlaces)

3.2. Sección de cabecera

Contenedores

Elemento	Descripción
-----	-----
<code>title</code>	Título del documento

`script`	Script referenciado o incrustado. Se puede agregar dentro de `head` y de `body`
`style`	Estilo aplicado usando CSS

****No contenedores****

Elemento	Descripción
-----	-----
`base`	URL base que se usará para las URL relativas contenidas en el documento
`link`	Añadir información externa afín al documento (CSS), ayuda a navegación, RSS, información de contacto
`meta`	Términos por los que debe ser encontrada ` <meta >`<="" content="Free Web tutorials" name="description" td=""/>

3.3 Formato al texto de un párrafo

Elemento	Descripción
-----	-----
`p`	Párrafo
`h1`	Encabezado de nivel i (Entre 1 y 6). El tamaño de la letra es mayor a menor i. No deben usarse estas etiquetas para formatear texto. Solo para títulos de párrafos.
`b`	Negrita
`i`	Cursiva
`u`	Subrayado
`sup`	Superíndice
`sub`	Subíndice
`strong`	Elemento resaltado. Normalmente en negrita aunque podrían ponerlo en cursiva y en naranja.

3.4. Listas

Elemento	Descripción
-----	-----
`ul`	Lista desordenada (viñetas)
`ol`	Lista ordenada (numeritos de orden)
`li`	Cada elemento de una lista
`dl`	Lista de definición (Tiene sangría)
`dt`	Cada término de una lista de definición
`dd`	Cada definición de una lista de definición

```
```html
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 transitional//EN" "http://www.w3.org/TR/REC-html40/loose.dtd">
```

```
<html>
```

```
 <head>
```

```
 <title>Listas</title>
```

```
 </head>
```

```
 <body>
```

```
 <h3>Ejemplo de lista desordenada: Modulos de 1Âº de ASIR</h3>
```

```
 Fundamentos de Hardware
 Gestión de Bases de Datos

```

```
 <h3>Ejemplo de lista ordenada: Modulos de 1Âº de ASIR</h3>
```

```
 Fundamentos de Hardware
 Gestión de Bases de Datos

```

```
 <h3>Ejemplo de lista de definición: Modulos de 1Âº de ASIR</h3>
```

```
 <dl> <dt>Fundamentos de Hardware</dt>
 <dd>Componentes físicos de un ordenador</dd>
 <dt>Gestión de Bases de Datos</dt>
```



```

 <dd>Diseño y uso de bases de datos relacionales</dd>
 </dl>
</body>
</html>

```

## 3.5. Tablas

Elemento	Descripción
<code>table</code>	Contenido de una tabla
<code>tr</code>	Cada línea de la tabla
<code>td</code>	Cada celda de la tabla
<code>colgroup</code>	Para agrupar columnas
<code>tbody</code>	Para agrupar líneas de la tabla
<code>thead</code>	Línea cabecera de la tabla
<code>th</code>	Cada celda de la cabecera
<code>tfoot</code>	Fila pie de la tabla

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 transitional//EN" "http://www.w3.org/TR/REC-html40/loose.dtd">

<html>
 <head>
 <title>Tablas</title>
 </head>

 <body>
 <h3>Ejemplo de tabla: Notas del môdulo de LMSGI de 1º de ASIR</h3>
 <table border = "1">
 <thead> <tr> <th>Eval</th> <th>LMSGI</th> </tr> </thead>
 <tfoot align="center"> <tr> <td>Media</td> <td>8</td> </tr> </tfoot>
 <tbody align="center">
 <tr> <td>Primera</td> <td>6</td> </tr>
 <tr> <td>Segunda</td> <td>7</td> </tr>
 <tr> <td>Tercera</td> <td>8</td> </tr>
 </tbody>
 </table>
 </body>
</html>

```

## 3.6. Formularios

Elemento	Descripción
<code>form</code>	Contenido de un formulario (Solicitar información al usuario)
<code>input</code>	Caja de texto para texto corto. Según el atributo <code>type</code> puede introducirse un texto sin más, un campo en el que al escribir lo que se visualicen sean asteriscos, un boton de radio.. <div> <code>text</code>  <code>password</code>  <code>email</code>  <code>checkbox</code>  <code>radio</code>  <code>submit</code>  <code>reset</code>  <code>date</code>  <code>submit</code> (habria que decirle el method o lo mandará todo por GET)....         </div>
<code>textarea</code>	Caja de texto para texto largo
<code>select</code>	Menú desplegable para elegir de una lista de opciones

Elemento	Descripción
<code>option</code>	Cada una de las opciones del desplegable
<code>button</code>	Un botón. Así puede introducirse en el botón cualquier otro elemento HTML como imágenes (en el creado con <code>input</code> no se puede eso)
<code>fieldset</code>	Agrupar elementos de un formulario
<code>legend</code>	Título al fieldset
<code>label</code>	Etiqueta del campo del formulario

### 3.7. Frames (En desuso por completo)

Elemento	Descripción
<code>frameset</code>	Partición de la ventana del navegador en marcos (files o columnas), Para partirlo en filas y columnas deben anidarse frames
<code>frame</code>	Define marco que contiene información

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN" "http://www.w3.org/TR/REC-html40/frameset.dtd">

<html>
 <head> <title>Frames</title> </head>

 <frameset rows="20%,30%,*">
 <frame src="LMSGI02_Ejemplo07_1.html" /> <frame src="LMSGI02_Ejemplo03.html" />
 <frame src="LMSGI02_Ejemplo05.html" />
 </frameset>
</html>

```

### 3.8. Capas o divs

- Las capas son páginas que se pueden incrustar dentro de otras: Bloques con contenido HTML que pueden situarse en la página de forma dinámica.
- Los atributos de una capa pueden y deben definirse dentro de una hoja de estilo (posición, visibilidad..). El contenido debe estar en la parte principal de la página.
- Las capas tienen sentido cuando se les pueden aplicar estilos CSS
- Las capas pueden ser movidas y modificadas desde un lenguaje de script (es lo más importante) Aunque antes era difícil que las implementaciones de los distintos navegadores -incompatibles entre sí- permitiesen un código que funcionase de la misma forma en todos ellos.

Elemento	Descripción
<code>div</code>	Sección dentro del documento XHTML
<code>id</code>	Identificador único
<code>style</code>	Estilo al contenido
<code>class</code>	Estilo al contenido

### 3.9. Otros elementos

Elemento	Descripción
<code>a</code>	Enlace a página web
<code>img</code>	Insertar imagen en página web
<code>abbr</code>	Forma abreviada
<code>acronym</code>	Acrónimo
<code>blockquote</code>	Texto con sangría
<code>q</code>	Cita. Se añaden marcas de citación

Elemento	Descripción
<code>br</code>	Línea en blanco.

```
Mi Google

El <abbr title="señor">Señor</abbr>
El <acronym title="Partido Socialista Obrero Español">PSOE</acronym>
```

## 4. XHTML frente a HTML

- Lenguaje XHTML es similar al HTML. De hecho, es adaptación de HTML a XML.
- Estándar XHTML 1.0 añade pequeñas mejoras y modificaciones a HTML 4.01 por lo que pasar de HTML 4.01 a Strict no requiere ningún cambio casi.
- XHTML no tiene sintaxis tan permisiva. añade normas a la forma de escribir etiquetas y atributos.

### 4.1. XHTML: Diferencias sintácticas y estructurales con HTML

El esquema básico del documento para cumplir la especificación debe cumplir:

- Debe haber declaración DOCTYPE en el prólogo del documento y su identificador público debe hacer referencia a alguna de las tres DTD definidas por el W3C:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

- El elemento raíz debe ser `<html>`
- El elemento raíz debe indicar el espacio de nombres XHTML usando `<html xmlns="http://www.w3.org/1999/xhtml">`
- Las etiquetas se tienen que cerrar en el orden inverso al que se abren, nunca pueden solaparse
- Nombres de etiquetas y atributos siempre en minúsculas
- Valor de atributos siempre entre comillas y con algun valor
- Todas las etiquetas deben cerrarse siempre (permitido `<br/>`)
- Antes de acceder al valor de un atributo se eliminan todos los espacios en blanco que se encuentran antes y después del valor. Además se eliminan todos los espacios en blancos sobrantes dentro del valor del atributo.
- El código Javascript debe encerrarse entre etiquetas especiales (`<![CDATA[ ]]>`) para evitar que el navegador interprete de forma errónea caracteres como & y <.
- Las páginas XHTML no deben usar el atributo `name` sino el atributo `id`
- En XHTML debe separarse el formato del contenido. Los párrafos deben separarse consistentemente y las cabeceras h1-h6 solo deben usarse para destacar los apartados. Debe darse formato usando CSS.

Introducción a XHTML: <https://uniwebsidad.com/libros/xhtml?from=librosweb>

### 4.2. Ventajas e inconvenientes de XHTML sobre HTML

Ventajas:

- Compatibilidad parcial con navegadores antiguos. Información visualizada aunque sin formato
- Un mismo documento puede adoptar diseños distintos en diferentes apartados
- Sencillo de editar y mantener

- Compatible con los estándares que desarrollaba (en su día) W3C como recomendale para futuros navegadores (agentes de usuario)
- Documentos XHTML 1.0 tienen (en su día tenían) mejor rendimiento en las herramientas web
- Separación de contenidos y presentación los hace más adaptables
- Se pueden usar herramientas para procesar XML genéricos (editores, XSLT,..)

Inconvenientes:

- Navegadores antiguos no son totalmente compatible con los estándares.
- Muchas herramientas de diseño web no generan código XHTML correcto.

## 5. Herramientas de diseño web

Hay herramientas de edición de código, de software de diseño web y recursos.

**Adobe Color CC:** Mejora el color de las fotografías. Analiza el tono de las ilustraciones y forma soluciones armónicas. Gama de tonalidades para usar. Ajustes de color, tono, saturación, curvas, máscaras de luminosidad. Gratuita y fácil de usar. Disponible en Adobe InDesign y Adobe AfterEffects. Accesible también desde Photoshop.

**Google Fonts:** Fuentes originales y adecuadas. Catálogo gratuito con más de 800 opciones. Sección Featured con colecciones temáticas creada por la misma herramienta.

**Whatfontis:** Nos dice qué fuente era. Subiendo una imagen o una URL a su sitio. Permite editar dicha imagen y obtener mejores resultados. Tiene comunidad de usuarios.

**Mockflow:** Construir esquemas de diseño de forma amplia. Arrastrar y soltar para colocar los elementos con iconos preconfigurados.

**Tinypng:** Optimizar imágenes comprimiendo contenidos gráficos para evitar la lentitud. Las optimiza de forma automática.

**Adobe Edge Reflow:** Creadores de diseño web responsive. Crea prototipos web funcionales a través de herramientas. Adaptable a cualquier resolución. Cumple HTML5 y CSS3.

**Notepad++:** Identifica expresiones, optimiza rutinas, uso sencillo. También SublimeText existe.

**Pixabay:** Banco de imágenes que puede usarse de forma gratuita.

**Ceros:** Para crear piezas de contenido diseñando infografías, ebooks, banners, micrositos, revistas..

**Canva:** Creación de imágenes, cabeceras para blogs y web. Creación de publicaciones en redes. Sencilla, gratuita y online.

# UD 02 - UTILIZACIÓN DE LENGUAJES DE MARCAS EN ENTORNOS WEB (Parte 2 - Hojas de estilo o CSS)

## 1. Introducción a CSS

CSS (Cascading Style Sheets, Hojas de estilos en cascada) permite a los desarrolladores controlar el estilo y el formato de múltiples páginas web al mismo tiempo.

- Sin él, se debería definir el aspecto de cada elemento dentro de la etiqueta HTML de la página (muy difícil de actualizar).
- CSS desacopla el marcado de los contenidos de la página (HTML/XML, designar la función de cada elemento dentro de la página) y su aspecto o formato (con CSS).
- En una zona reservada se define el formato de cada elemento de la web. Las hojas CSS están compuestas por reglas de estilo que se aplican sobre documentos XHTML o HTML. Cualquier cambio en las reglas afecta a los documentos vinculados a la hoja.

Ventajas CSS:

- Obliga a crear documentos semánticos HTML/XHTML
- Mejora la accesibilidad del documento
- Reduce la complejidad de mantenimiento
- Permite visualizar el mismo documento en dispositivos diferentes

Las hojas de estilos aparecen después del lenguaje SGML, en el año 1970.

### 1.1. Niveles de CSS y prefijos

CSS es desarrollado por el World Wide Web Consortium (W3C). Participan en él casi todas las empresas relacionadas con internet (Appler, Adobe, Akamai, Cisco, Google, Facebook, HP, Intel, LG, Microsoft, Twitter...)

Hay muchos grupos de trabajo (WHATWG, CSSWG, AGWG, WOTWG)

Los niveles de CSS son los siguientes:

Nivel	Año	Descripción
<b>CSS1</b>	1996	Propiedades para tipografías, colores, alineación, etc...
<b>CSS2</b>	1998	Propiedades de posicionamiento, tipos de medios, etc...
<b>CSS2.1</b>	2005	Corrige errores de CSS2 y modifica ciertas propiedades
<b>CSS3</b>	2011	Características de CSS como módulos independientes

Los módulos van evolucionando independientemente. Por ejemplo el módulo Flexbox está en nivel 1, el módulo de Grid en nivel 2, el módulo de características de texto en nivel 3.

Recientemente se está volviendo a implementar la idea de volver al versionado general de CSS (CSS4, CSS5,...)

#### 1.1.1. Prefijos CSS de navegador - Vendor prefixes (obsoleto)

Si las propiedades tienen un comportamiento diferente según el navegador, puede hacerse referencia por separado con los prefijos

```
propiedad: /* Especificacion OFICIAL. Si no tiene soporte completo ignora la
propiedad */
-webkit-propiedad: /* Chrome antiguo / Safari (Motor webkit)*/
```

```
-moz-propiedad: /* Firefox antiguo (Motor Gecko)*/
-ms-propiedad: /* Internet Explorer (Motor Trident) */
-o-propiedad: /* OPERA antiguo (Motor Presto) */
```

Autoprefixer (plugin de PostCSS) añade de forma automática los prefijos basándose en información como de CanIUse

### 1.1.2. Flags CSS de navegador (lo que se lleva ahora)

Habilitar las funciones experimentales desde una ventana del navegador (avisa de que son inestables)

## 1.2. Formas de enlazar CSS

### A. Archivo CSS externo

#### A.1. Mediante enlaces

El enlace con el CSS externo es tal que así:

```
<link rel="stylesheet" href="style.css" />
```

Sus atributos pueden ser

- **rel**: indica el tipo de relación
- **type** (opcional): `type="text/css"` pero ya no es necesario en HTML5
- **href**: URL (puede ser relativa o absoluta, referenciar a algo interno o externo)
- **media** (opcional): medio en el que se aplicarán los estilos CSS

Es la técnica recomendada para que el CSS pueda ser reutilizable y enlazarlo con más documentos HTML.

(Aconsejado escribirlo lo antes posible, sobre todo antes de los scripts, obligando al navegador a aplicar estilos cuanto antes y a que la página no se vea en blanco)

#### A.2. Mediante importación del fichero CSS

Puede obtenerse el mismo resultado usando el elemento `<style>` en lugar de `<link>`

Se hace con la regla `@import "direccionurl"` (Con comillas simples o dobles, pasando la URL del archivo.

También vale `@import url("direccionurl")`

La regla import se evalúa en el navegador al cargar la página (cada regla es petición al servidor para cargar el .css). Herramientas como Sass, PostCSS, LightningCSS tienen mecanismos para realizar imports de forma anticipada y generar un solo fichero con todo el código CSS para que el navegador haga menos peticiones.

```
<head>
 <title></title>
 <style>@import 'formatos.css';</style>
</head>
```

Tenemos añadidos interesantes:

- **Con media query:** `@import url("mobile.css") (with <= 640px);` o `@import url("mobile.css") print` (Si se está imprimiendo la página actual).
- **Si soporta la condición:** `@import url("mobile.css") supports(not (display: grid));`
- **Colocándolo en una capa virtual de CSS** (mantener aislados los estilos de un framework sin usar `!importat` o reescribir selectores para forzarlos) `@import url("mobile.css") layer/framework)`
- **Colocándolo en nueva capa anónima:** `@import url("mobile.css") layer()`

## B. Incluir CSS en el documento HTML: Bloque de estilos en el `<head>`

```
<!DOCTYPE html>
<html>
 <head>
 <title>Título de la página</title>
 <style>
 div {
 background: hotpink;
 color: white;
 }
 </style>
 </head>
 ...
</html>
```

## C. Incluir CSS en el elemento HTML: Estilos en línea con etiqueta `<style>`

```
<p>¡Hola amigo lector!</p>
```

# 2. Estructura de CSS

El código CSS se basa en las ideas de:

Concepto	Descripción
<b>Selector</b>	Elemento (elementos) del documento que se seleccionan para aplicar un estilo
<b>Propiedad</b>	Característica que se define con el valor indicado a continuación
<b>Valor</b>	Valores que pueden asignarse a una propiedad
<b>Comentario</b>	Fragmento entre <code>/* */</code> que el navegador ignora
<b>Regla</b>	Par de propiedad,valor

Por ejemplo, seleccionamos todas las etiquetas `<p>` del documento HTML

```
p {
 color: red
}
```

Fíjate que el selector del elemento html no va encerrado entre signos.  
Lo que están encerrados entre llaves son las reglas o declaraciones (propiedad:valor)  
Consejos:

- Escribe una regla por línea
- Usa la indentación
- Aconsejado escribir el último ; aunque sea opcional para evitar descuidos

# 3. Colores CSS

El **COLOR** puede expresarse  
En el espacio de color RGB

- Mediante palabras clave predefinidas: `red`, `aquamarine`, `brown`... ; `transparent` (Color transparente, el por defecto que haya en `background-color`) `currentColor` (color que se esté usando para el texto); Colores del sistema operativo: `visitedtext` (ya visitado), `mark` (marcado subrayado)...
- Mediante la función `rgb`

```
rgb(r g b)
rgb (r g b / a)
```

`rgb(r, g, b)` (Legacy)  
`rgb(r, g, b, a)` (Legacy)  
`rgba(r, g, b, a)` (Legacy)

Colores: Rangos 0-255 0%-100% de oscuro a claro respectivamente.

Transparencia: Rangos 0-1 0%-100%, siendo 1 y 100% la transparencia total.

- Con **notación hexadecimal** `#rrggbb`

Palabra clave	Función RGB	Hexadecimal	Hex. abreviado
<code>red</code> (rojo)	<code>rgb(255 0 0)</code>	<code>#FF0000</code>	<code>#F00</code>
<code>black</code> (negro)	<code>rgb(0 0 0)</code>	<code>#000000</code>	<code>#000</code>

- Con la función de `hsl()` (color, saturación y brillo)

`hsl(h s l)` h: Matiz de color (grados). s: Porcentaje de saturación. l: Porcentaje de luminosidad.

`hsl(h s l / a)`

`hsl(h s l / a)`

`hsl(h, s, l)`

`hsl(h, s, l, a)`

- Con la función de `hwb()` (color claridad y oscuridad)  
En el espacio de calor independiente del dispositivo
- Con la función `lab()` y `oklab()` (luminosidad CIE, eje A y eje B)
- Con la función `lch()` y `oklch()` (luminosidad CIE, saturación y color)

Por cierto, con `color-mix()` podemos mezclar varios colores en un espacio de color, opcionalmente indicando la cantidad de color y opcionalmente indicando el método de interpolación.

Es útil usar ColorPicker para elegir colores. O Developers tools (Power Toys) para extraerlos de una web ya existente.

### 3. Fondos CSS

Propiedad	Significado	Valores
<code>color</code>	Cambia el <b>color del texto</b> que está en el interior de un elemento.	
<code>background-color</code>	Cambia el <b>color de fondo</b> de un elemento.	
<code>background-image</code>	Imagen de fondo del elemento.	<code>none</code> (ninguna), <code>url("miUrl")</code> (url), <code>image-set(...)</code> (Imagen con fallbacks), <code>//UN_GRADIENTE//</code>  Puedo añadir fondos múltiples separando por comas (La última es la primera que se renderiza) El resto de las propiedades se puede indicar separando por comas para que se apliquen diferenciadamente.
<code>background-repeat</code>	Indica si ha de repetirse la imagen de fondo.	<code>repeat-x</code> (solo horizontalmente) <code>repeat-y</code> (Solo verticalmente) <code>space</code> (rellena con espacios los huecos) <code>round</code> (Amplia cada repetición para ajustar) <code>no-repeat</code> (No se repite)
<code>background-attachment</code>	Especifica si la imagen ha de permanecer fija o hacer un scroll.	<code>scroll</code> o <code>fixed</code>
<code>background-position</code>	Posicionar la imagen.	Porcentaje, tamaño o indicación de posicionamiento <code>[top, center, bottom]</code> <code>[left, center, right]</code>



Propiedad	Significado	Valores
<code>background-clip</code>	Área externa afectada por el fondo (modelo de cajas)	<code>border-box</code> (borde, espaciado y contenido) <code>padding-box</code> (espaciado y contenido) <code>content-box</code> (solo contenido)
<code>background-origin</code>	Área interna afectada por el fondo (Modelo de cajas)	<code>border-box</code> <code>padding-box</code> <code>content-box</code>
<code>background-size</code>	Tamaño a la imagen de fondo.	Si se pasa un parámetro aplica un size de (ancho x auto), mantiene la proporción. Si se pasan dos aplica un size de (ancho x alto) y no mantiene la proporción.
<code>background</code>	Establece las propiedades de una vez.	<code>background-color</code> , <code>background-image</code> <code>background-position</code> / <code>background-size</code> <code>background-repeat</code> <code>background-attachment</code> <code>background-origin</code> <code>background-clip</code> (Solo esperará size si se pone la barra /)

## 4. Fuentes y tipografías. Cargar tipografías.

Propiedad	Significado	Valores
<code>font-family</code>	Indica el nombre de la fuente (o una lista de ellas).	Si la fuente tiene espacios, se usa comilla simple. Se pone valor como <code>Verdana</code> o <code>'PT Sans'</code> ... La buena práctica es establecer una lista de fotografías separadas por comas por si alguna no está admitida. La última opción debe ser una "web-safe fons" (fuente segura) con la que se designa categoría de tipografías para que busque el navegador alguna del sistema.
<code>font-size</code>	Indica el tamaño de la fuente.	Tamaño absoluto, relativo o en porcentaje
<code>font-style</code>	Indica el estilo de la fuente. Por defecto, <code>normal</code> .	<code>normal</code> , <code>italic</code> , <code>oblique</code> (Similar a cursiva pero con inclinación de forma artificial y no la dada por el diseñador de la tipografía)
<code>font-weight</code>	Indica el peso (grosor) de la fuente. Por lo general, un valor entre <code>100</code> - <code>800</code> .	<code>normal</code> , <code>bold</code> , <code>bolder</code> , <code>lighter</code> , <code>100</code> , <code>200</code> ,... <code>900</code>
<code>font-variant</code>	Normal o mayúsculas pequeñas	<code>normal</code> , <code>small-caps</code>
<code>font</code>	Establecer todas de una vez	<code>font-style font-variant font-weight stretch font-size/line-height</code> <code>font-family</code> . Valores separados por espacios.

Ejemplillo del uso del `font`

```
.container {
 /* Opción 1 */
 font: italic normal 400 16px Arial, Verdana, sans-serif;

 /* Opción 2 */
 font: italic normal 400 normal 16px/22px Arial, Verdana, sans-serif;
}
```

Son fuentes seguras:

Fuente	Significado	Fuentes de ejemplo
<code>serif</code>	Tipografía con serifa	Times New Roman, Georgia...
<code>sans-serif</code>	Tipografía sin serifa	Arial, Verdana, Tahoma...

Fuente	Significado	Fuentes de ejemplo
<code>cursive</code>	Tipografía en cursiva	Sanvito, Corsiva...
<code>fantasy</code>	Tipografía decorativa	Critter, Cottonwood...
<code>monospace</code>	Tipografía monoespaciada	Courier, Courier New...

Y más relacionadas con el sistema:

Fuente	Significado
<code>system-ui</code>	Tipografía por defecto del sistema.
<code>ui-serif</code>	Tipografía serif por defecto del sistema.
<code>ui-sans-serif</code>	Tipografía sans serif por defecto del sistema.
<code>ui-monospace</code>	Tipografía monoespaciada por defecto del sistema.
<code>ui-rounded</code>	Tipografía con bordes redondeados por defecto del sistema.
<code>math</code>	Tipografía especializada para conceptos matemáticos.
<code>emoji</code>	Tipografía diseñada especialmente para emojis.
<code>fangsong</code>	Tipografía con caracteres de estilo chino.

## 5. Texto

### 5.1. Trazos que decoran con `text-decoration`

Propiedad	Significado	Valores
<code>text-decoration-line</code>	Indica el tipo de decoración de texto.	<b>none</b>   <code>underline</code> (subrayado)   <code>overline</code> (por encima)   <code>line-through</code> (línea tachada)   <code>blink</code> (quizás parpadee, según el navegador)
<code>text-decoration-style</code>	Trazo de la decoración.	<b>solid</b>   <code>double</code> (doble subrayado)   <code>dotted</code> (punteado)   <code>dashed</code> (rayado)   <code>wavy</code> (ondulado)
<code>text-decoration-color</code>	Indica el color de la decoración. (Trazo)	<b>currentcolor</b>   COLOR
<code>text-decoration-thickness</code>	Indica el grosor del trazo de la decoración.	<b>auto</b>   <code>from-font</code>   SIZE
<code>text-underline-position</code>	Indica donde aparece el trazo del subrayado.	<b>auto</b>   <code>from-font</code>   <code>under</code>
<code>text-underline-offset</code>	Indica el desplazamiento del trazo del subrayado.	<b>auto</b>   SIZE
<code>text-decoration</code>	Propiedad de atajo de las anteriores.	<code>line thickness style color</code>

### 5.2. Alineaciones de texto con `text-align`

Propiedad	Valor	Significado
<code>text-align</code>	<b>start</b>   <code>end</code>   <code>justify</code>   <code>center</code>   <code>match-parent</code>   <code>justify-all</code>	Alineación del texto.

Propiedad	Valor	Significado
<code>text-align-last</code>	<code>auto</code>   <code>start</code>   <code>end</code>   <code>justify</code>   <code>center</code>   <code>match-parent</code>	Alineación de última línea del texto.
<code>text-justify</code>	<code>auto</code>   <code>inter-word</code>   <code>inter-character</code>   <code>none</code>	Método de justificación de textos en <code>justify</code> .
<code>vertical-align</code>	<code>baseline</code>   <code>sub</code>   <code>super</code>   <code>top</code>   <code>middle</code>   <code>bottom</code> <code>text-top</code>   <code>text-bottom</code>	Alineado de textos respecto a elementos.

Valores	Descripción
<code>start</code>	Alinea el texto al principio. También se puede usar <code>left</code> .
<code>end</code>	Alinea el texto al final. También se puede usar <code>right</code> .
<code>center</code>	Alinea el texto en el centro.
<code>justify</code>	Justifica el texto, es decir, procura que ocupe toda la línea.
<code>match-parent</code>	Utiliza la alineación establecida en el elemento padre.
<code>justify-all</code>	Usa <code>justify</code> en las propiedades <code>text-align</code> y en <code>text-align-last</code> .

### 5.3. Espaciado de texto e identaciones

Propiedad	Valor	Significado
<code>letter-spacing</code>	<code>normal</code>   SIZE	Espacio entre letras (interletraje o tracking).
<code>word-spacing</code>	<code>normal</code>   SIZE	Espacio entre palabras.
<code>line-height</code>	<code>normal</code>   NUMBER   SIZE	Establece una altura de línea (interlineado).
<code>text-indent</code>	<code>0</code>   SIZE	Indentación de texto (sangría).

### 5.4. Transformaciones `text-transform`

Propiedad	Significado	Valores
<code>text-decoration-line</code>	Indica el tipo de decoración de texto.	<code>none</code>   <code>underline</code> (subrayado)   <code>overline</code> (por encima)   <code>line-through</code> (línea tachada)
<code>text-decoration-style</code>	Trazo de la decoración.	<code>solid</code>   <code>double</code> (doble subrayado)   <code>dotted</code> (punteado)   <code>dashed</code> (rayado)   <code>wavy</code> (ondulado)

### 5.5. Énfasis `text-emphasis`

Propiedad	Valor	Significado
<code>text-emphasis-style</code>	<code>none</code>   <code>dot</code>   <code>circle</code>   <code>triangle</code> <code>double-circle</code>   <code>sesame</code>	Indica un carácter para utilizar de énfasis.
<code>text-emphasis-string</code>	String	Indica un carácter personalizado de énfasis.
<code>text-emphasis-color</code>	Color	Indica el color de los caracteres de énfasis.
<code>text-emphasis-position</code>	<code>over right</code>   <code>over left</code> <code>under right</code>   <code>under left</code>	Indica la posición de los caracteres de énfasis.
<code>text-emphasis</code>	Style   Color	Propiedad de atajo de las dos primeras.

La propiedad `text-emphasis-style` permite establecer unos caracteres para enfatizar los textos, que por defecto se establecen sobre los mismos. Se pueden indicar los valores `dot`, `circle`, `double-circle`, `triangle` o `sesame`, los cuales pueden combinarse con las palabras clave `open` y `filled` si queremos que sean signos huecos o rellenos, respectivamente:

## 5.6. Ajustes y balanceos del texto con `text-wrap`

Valor	Descripción
<code>wrap</code>	El texto se ajusta para ocupar el ancho del contenedor, y se dividirá en varias líneas si es necesario.
<code>nowrap</code>	El texto no se ajusta, por lo que sobresale del contenedor si es más largo que su ancho.
<code>balance</code>	El texto se ajusta de forma uniforme, evitando líneas muy largas o muy cortas. Ideal para títulos.
<code>pretty</code>	El texto se ajusta de forma uniforme, minimizando la diferencia de longitud de las líneas. Ideal para párrafos.
<code>stable</code>	⚠ El texto se ajusta de forma uniforme, manteniendo los espacios entre palabras uniforme.
<code>auto</code>	El navegador determina que tipo de ajuste aplicar.

## 5.7. Espacios en blanco

Con `white-space` se puede elegir qué hacer con los espacios en blanco.

Valor	Espacios en blanco consecutivos	Contenido
<code>normal</code>	Los espacios consecutivos se transforman en uno solo.	Se ajusta al contenedor.
<code>nowrap</code>	Los espacios consecutivos se transforman en uno solo.	Ignora saltos de línea.
<code>pre</code>	Respeto y muestra literalmente los espacios.	Ignora saltos de línea.
<code>pre-wrap</code>	Respeto y muestra literalmente los espacios.	Se ajusta al contenedor.
<code>pre-line</code>	Respeto literalmente los espacios y suprime los espacios del final.	Se ajusta al contenedor.

Con `tab-size` se puede establecer el número de espacios que se mostrarán en el cliente con un TAB (visibles en elementos como `<textarea>` o ```<pre>`)

## 5.8. Límites de línea y de palabra

Propiedad	Valor	Significado
<code>word-break</code>	<code>normal</code>   <code>keep-all</code>   <code>break-all</code>   <code>break-word</code>	Indica si se pueden partir palabras de forma natural.
<code>line-break</code>	<code>auto</code>   <code>loose</code>   <code>normal</code>   <code>strict</code>   <code>anywhere</code>	Determina como dividir líneas.
<code>hyphens</code>	<code>manual</code>   <code>none</code>   <code>auto</code>	Indica si se debe dividir las palabras por guiones.
<code>overflow-wrap</code>	<code>normal</code>   <code>break-word</code>   <code>anywhere</code>	Indica si puede forzar partir palabras y evitar desbordamiento

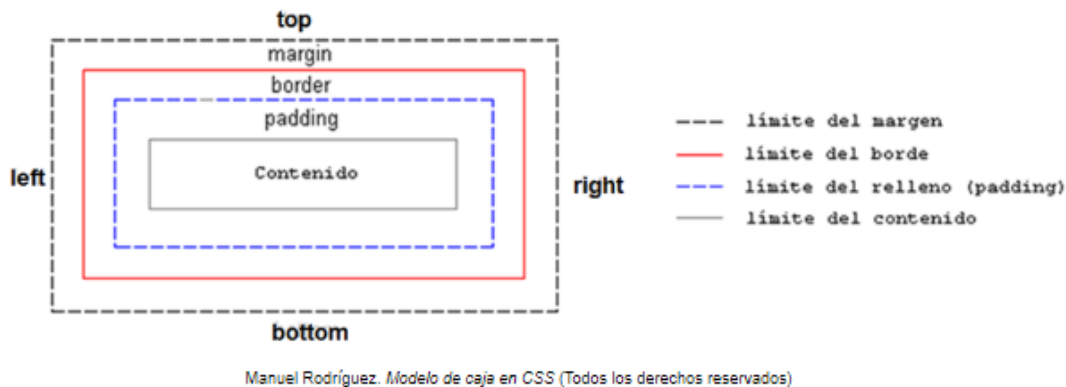
## 5.9. Contornos con `text-shadow`

Se usa para sombras de texto.

Puede crearse múltiples formar y desplazarlas hacia un lado. En conjunto creará una sombra exterior.

```
.text {
 font-family: sans-serif;
 font-weight: bold;
 font-size: 3rem;
 color: black;
 text-shadow:
 1px 0 0 white, /* Desplaza a la derecha */
 -1px 0 0 white, /* Desplaza a la izquierda */
 0 1px 0 white, /* Desplaza abajo */
 0 -1px 0 white; /* Desplaza arriba */
}
```

## 6. Modelo de cajas



- Las cajas de un documento HTML se crean automáticamente porque **cada etiqueta HTML da lugar a una caja**
  - A simple vista no son visibles, puesto que no se muestran con ningún color de fondo ni bloque
  - El navegador considera que:
    - Elementos de bloque no permiten a otro elemento situarse en la misma línea `<div>`, `<p>`, `<h1>`, `<form>`, `<ul>`, `<li>`
    - Elementos de línea que permiten a otros elementos situarse en la misma línea `<span>`, `<a>`, `<label>`, `<input>`, `<img>`, `<strong>`, `<em>`
- Con CSS podemos
- `display: block`: Hacemos que un elemento de línea se comporte como bloque
- `display: inline`: Hacemos que un elemento de bloque se comporte como línea

La caja tiene los siguientes elementos:

- Content** (Contenido)
- Padding** (Relleno)
- Border** (Borde)
- Background image** (Imagen de fondo)
- Background color** (Color de fondo)
- Margin** (Margen)

### Content

Contenido HTML del elemento (palabras, imagen, texto, lista de elementos...).

Tiene `width`, `height`, `color`, `clear` (Si tiene a su altura imágenes u otros elementos alineados a derecha o a izquierda. Limpia las partes... `none`, `left`, `right`, `both`), `float` (alinear elemento a izquierda o a derecha haciendo que el texto se agrupe alrededor de dicho elemento)

### Padding o relleno

Espacio libre opcional existente entre el contenido y el borde. Es transparente, se muestra del color de la imagen de fondo.

Propiedades `padding-top`, `padding-right`, `padding-bottom`, `padding-left`, `padding`

(Establecernos de una vez en orden superior, derecho, inferior, izquierdo)  
(Longitud, unidades CSS, porcentaje).

## Background image

## Border

Línea que encierra completamente su contenido y su relleno.

Ancho

Propiedades `border-top-width`, `border-left-width`, `border-right-width`, `border-bottom-width`, `border-width`. Podría ser `thin`, `medium`, `thick` o tamaño.

Color

`border-top-color`, `border-left-color`, `border-right-color`, `border-bottom-color`, `border-color`. Establecerlos de una vez en superior, derecho, inferior, izquierdo.

Estilo

`border-top-style`, `border-left-style`, `border-right-style`, `border-bottom-style`, `border-style`

Valor	Descripción
<code>hidden</code>	Oculto. Idéntico a <code>none</code> , salvo para conflictos con tablas.
<code>dotted</code>	Establece un borde basado en puntos.
<code>dashed</code>	Establece un borde basado en rayas (línea discontinua).
<code>solid</code>	Establece un borde sólido (línea continua).
<code>double</code>	Establece un borde doble (dos líneas continuas).
<code>groove</code>	Establece un borde biselado con luz desde arriba.
<code>ridge</code>	Establece un borde biselado con luz desde abajo. Opuesto a <b>groove</b> .
<code>inset</code>	Establece un borde con profundidad «hacia dentro».
<code>outset</code>	Establece un borde con profundidad «hacia fuera». Opuesto a <b>inset</b> .

## Margin o margen

Separación original existente entre la caja y el resto de cajas adyacentes. Es transparente.

Propiedades `margin-top`, `margin-right`, `margin-bottom`, `margin-left`, `margin` (Establecernos de una vez en orden superior, derecho, inferior, izquierdo)  
(Longitud, unidades CSS, porcentaje).

## Background image

Imagen por detrás del contenido y del espacio de rellano

Propiedades: `background-image`, `background-repeat`, `background-position`, `background-attachment`, ...

## Background color

Color que se muestra por detrás del contenido y del espacio de relleno

Propiedades: `background-color`, `background`...

## 6.1. Clasificación

Elemento	Descripción	Valores
<code>display</code>	Determina si el elemento es de bloque, de línea, de lista o ninguno de ellos.	<code>block</code> , <code>inline</code> , <code>list-item</code> , <code>none</code>

Elemento	Descripción	Valores
<code>list-style-type</code>	Símbolo que se usa como viñeta en las listas	<code>disc</code> , <code>circle</code> , <code>square</code> , <code>decimal</code> , <code>lower-roman</code> , <code>upper-roman</code> , <code>lower-alpha</code> , <code>upper-alpha</code> , <code>none</code>
<code>list-style-image</code>	Una imagen como varcador en la lista	
<code>list-style-position</code>	Posición del marcador en la lista	<code>outside</code> , <code>inside</code>
<code>list-style</code>	Reúne las características de tipo, posición e imagen en una sola.	

## 6.2. Posicionamiento y visualización

Elemento	Descripción
<code>clip</code>	Permite seleccionar una zona. Los valores que puede tomar son: <code>shape</code> o <code>auto</code> .
<code>height</code>	Permite establecer la altura de un elemento. Los valores que puede tomar son: <code>auto</code> o un tamaño.
<code>width</code>	Permite establecer la anchura de un elemento. Los valores que puede tomar son: <code>auto</code> , un tamaño o porcentaje.
<code>display</code>	Modifica la forma en la que se visualiza un elemento.
<code>visibility</code>	Indica si el elemento sobre el que actúa será visible o no. Los valores que puede tomar son: <code>hidden</code> y <code>collapse</code> .
<code>left</code>	Indica la posición del lado izquierdo del elemento. Los valores que puede tomar son: <code>auto</code> , un tamaño o porcentaje.
<code>top</code>	Indica la posición del lado superior del elemento. Los valores que puede tomar son: <code>auto</code> , un tamaño o porcentaje.
<code>overflow</code>	Indica si el elemento será visible o no en caso de superar los límites del contenedor. Los valores que pueden tomar son: <code>visible</code> , <code>hidden</code> , <code>scroll</code> o <code>auto</code> .
<code>position</code>	Determinan si el posicionamiento de un elemento es absoluto, relativo o estático. Los valores que puede tomar son: <code>absolute</code> , <code>relative</code> o <code>static</code> .
<code>z-index</code>	Define la posición del elemento en el tercer eje de coordenadas, permitiendo superponer unos elementos sobre otros como si fueran capas.

`display` y `visibility`

**display: none** Oculta por completo un elemento haciendo que los demás elementos ocupen su lugar  
**visibility: hidden** Hace que el resto de elementos respeten la posición.

Elemento	Descripción
<code>visibility: hidden</code>	Convierte una caja en invisible para que no muestre sus contenidos.
<code>visibility: collapse</code>	Sólo se puede utilizar en las filas, grupos de filas, columnas y grupos de columnas de una tabla. Su efecto es similar al de la propiedad <code>display</code> , ya que oculta completamente la fila y/o columna y se pueden mostrar otros contenidos en ese lugar. Si se utiliza el valor <code>collapse</code> sobre cualquier otro tipo de elemento, su efecto es idéntico al valor <code>hidden</code> .

## 7. Unidades de medida CSS

Tipo de unidad	Unidades	Descripción
Unidades absolutas	px, cm, mm, Q, in, pt, pc	Unidades estáticas o de tamaño fijo.
Unidades relativas		Unidades que dependen de otros factores.
	%	Unidades basadas en el tamaño del padre inmediato.
	em, rem	Unidades basadas en el tamaño de una tipografía.
	ex, rex	Unidades basadas en la altura de una minúscula.
	cap, rcap	Unidades basadas en la altura de una mayúscula.
	ch, rch	Unidades basadas en las medidas de un carácter europeo.
	ic, ric	Unidades basadas en las medidas de un carácter CJK.
	lh, rlh	Unidades basadas en en el interlineado.
Relativas al viewport	vw, vh, vmin, vmax, vi, vb	Unidades basadas en la región visible del navegador.
	svw, svh, svmin, svmax, svi, svb	Idem, en pantallas pequeñas (small viewport)
	lvw, lvh, lvmin, lvmax, lvi, lvb	Idem, en pantallas grandes (large viewport).
	dvw, dvh, dvmin, dvmax, dvi, dvb	Idem, en pantallas dinámicas (dynamic viewport).
Relativas al contenedor	cqw, cqh, cqmin, cqmax, cqi, cqb	Unidades basadas en un contenedor padre específico.
Relativas al grid	fr	Unidad basada en la fracción restante (sólo para grids).
Unidades de dirección	deg, grad, rad, turn	Unidades para indicar una dirección.
Unidades de duración	s, ms	Unidades para indicar un tiempo concreto.
Unidades de frecuencia	hz, khz	Unidades para indicar una frecuencia.
Unidades de resolución	dpi, dpcm, dppx	Unidades para indicar resoluciones.

**Absolutas:** Medida indicada por unidades absolutas que está completamente definida. Su valor no depende de otro valor de referencia. Ventaja: Es directamente lo que va a usarse, sin cálculos intermedios. Desventaja: son poco flexibles y no se adaptan fácilmente a los medios.

Pulgadas (in) 2,54cm.  
Puntos (pt) 1/72 in  
Picas (pc) 12 pt

**Relativas:** No están completamente definidas porque su valor siempre está referenciado a otro valor. Son más "difíciles" pero son las usadas en diseño web por adaptarse a los diferentes medios.

**em** : 1em equivale a la anchura de la letra M del tipo y tamaño de letra del elemento. 1em , si se usa una tipografía de 12 puntos, equivale a 12 puntos.

**ex** : Equivale al tamaño de la letra "x". 1 ex podría ser 0.5 em.

**px** : Tamaño relativo respecto a la resolución de pantalla del dispositivo.



**Porcentajes:** También es una unidad de medida relativa. Siempre está referenciado a otra medida. Los tamaños establecidos para `<h1>` y `<h2>` son de 2em y 1.5em.

La mayoría de propiedades CSS se hereda de padres a hijos. si se establece el tamaño de un padre, todos los elementos de la página lo heredarán, salvo que se indique otro valor. El valor de las medidas relativas no se hereda directamente sino que lo que se hereda es su valor real una vez calculado.

## 8. Selectores CSS

Los selectores permiten elegir el elemento que se quiere y darle un estilo

### 1.- Selector universal

Selecciona todos los elementos del HTML. Permite poner estilos generales. (Como fuentes, separaciones)

```
*{
 font-family: Verdana, Geneva, Tahoma, sans-serif;
 margin:0px;
 padding:0px;
}
```

`font-family` tiene varias fuentes porque si no coge una será otra.

`margin` Margen exterior

`padding` Margen interior

`text-decoration` Quita la decoración de link por ejemplo si está a `none`

### 2.- Selector de etiqueta

Selecciona un elemento con etiqueta concreta (`h1` , `div`, `h2`)

```
h1{
 color: white;
 font-size: 50px;
 padding:10px;
 background: red;
}

a{
 font-size: 18px;
 color: green;
 text-decoration: none;
}
```

### 3.- Selector de id

Para seleccionar por id (como en el caso `<div id="pepe">`) se usa la #

```
#descripcion{
 border: 5px solid black;
 padding: 15px;
}
```

Solo puede haber un id en la página

### 4.- Selector de clase

Solo puede haber un id en la página. Pero sí que podemos tener varias "clases" que agrupen elementos. En el HTML deberán referenciarse con el atributo `class`.

```
.miClase {
 color:blue
}
```

Puede restringirse que solo la usen ciertos elementos anteponiéndolos:

```
p.miClase {
 color:blue
}
```

## 9. En un futuro...

- Gradientes o degradados
- Cascada y herencia
- Pseudoclases CSS
- Pseudoelementos CSS
- Reglas CSS
- Representación de datos
- Interacciones
- Sombras
- Efectos
- Máscaras y recortes
- Responsive web design
- Animaciones CSS
- Transformaciones 2D/3D
- Dibujar con CSS
- Calidad de código
- Funciones CSS

## 10. CSS Moderno antes vs ahora

### A. Agrupar selectores

Antes:

```
.container .item,
.container .parent,
.container .element {
 /* ... */
}
```

Ahora se agrupan con el combinador `:is()`

```
.container :is(.item, .parent, .element) {
 /* ... */
}
```

### B. Escribir colores con transparencia (canal alpha)

Antes:

```
.container {
 background: rgba(255, 255, 0, 0.5);
}
```

Ahora:

```
.container {
 background: rgb(100% 100% 0 / 50%);
}
```

### C. Anidar código CSS (Un elemento dentro de otro)

Se puede anidar los componentes unos dentro de otros con `&`  
Antes:

```
.parent {
 background: grey;
}

.parent .element {
 background: darkred;
}

.parent .element:hover {
 background: red;
}
```

Ahora:

```
.parent {
 background: grey;

 & .element {
 background: darkred;

 &:hover {
 background: red;
 }
 }
}
```

### D. Centrar contenido de un elemento

Se puede centrar en varios ejes con una sola propiedad.  
Antes:

```
.parent {
 display: grid;
 justify-items: center;
 align-items: center;
}
```

Ahora:

```
.parent {
 display: grid;
 place-items: center;
}
```

### E. Variables CSS

Pueden declararse variables con `--variable: valor;` para reutilizar el valor.

```
.parent {
 width: 300px;
 height: 300px;
 background: grey;
}
```

```
.parent {
 --size: 300px;

 width: var(--size);
 height: var(--size);
 background: var(--color, grey);
}
```

## F. Media Queries

Puede usarse una sintaxis más clara y amigable.

Antes:

```
@media (min-width: 800px) and
 (max-width: 1280px) {
 .menu {
 background: red;
 }
}
```

Ahora:

```
@media (800px <= width <= 1280px) {
 .menu {
 background: red;
 }
}
```

# UD 03 - APLICACIÓN DE LOS LENGUAJES DE MARCAS A LA SINDICACIÓN DE CONTENIDOS

## 1. Sindicación de contenidos

La **sindicación de contenidos** permite a un sitio web utilizar los servicios o contenidos de otra web diferente (cumpliendo las licencias de normas de uso de contenidos o respetando las condiciones del contrato que regula los derechos de ese contenido).

Los **feed** o **canales de contenidos** están formados por el **contenido** y los **metadatos** que tiene asociados en el sitio original.

La redifusión de contenidos web suele hacerse con licencias de normas de uso. Consiste en ofrecer contenido desde una fuente cuyo origen está en una web para darles a los usuarios actualización del mismo.

Las fuentes se suelen codificar en XML, aunque pueden codificarse en cualquier lenguaje que se pueda transportar por el protocolo HTTP.

Para leer una fuente o canal hay que suscribirse usando un **agregador**.

Para que una web sea suministradora de un canal en su cabecera `<head>` tiene que incluir un enlace al canal de contenidos.

- Si está hecho con RSS:

```
<link rel="alternate" type="application/rss+xml" title="titulo_del_enlace" href="/feed/fichero_rss.xml"
```

- Si está hecho con Atom:

```
<link rel="alternate" type="application/atom+xml" title="titulo_del_enlace" href="/feed/fichero_atom.xml"
```

Además debe indicarse el enlace para acceder al archivo.

- Si está hecho con RSS:

```
Pulsa aquí
```

- Si está hecho con Atom:

```
Pulsa aquí
```

Publicar en la web puede verse como un flujo de información. Si el origen son unos ficheros en ordenador local, el trabajo es que llegue a los usuarios que leen. Cuando la información se codifica en HTML, se logra actualizando dicho documento en el directorio del servidor web que contiene la página.

Es habitual usar un CMS (sistema de gestión de contenidos). Los contenidos están en un repositorio y, antes de ser servidos al cliente, son transformados (en HTML, en RSS...). Puede haber incluso más de un repositorio. Como están en un CMS las transformaciones pueden replicarse generando tantos ficheros HTML como canales RSS.

### Tipos de transformaciones:

Documento XML -> Transformación XSLT (Generar documentos a partir de XML

<https://www.mclibre.org/consultar/xml/lecciones/xml-xslt.html>) -> Documento XHTML

Base de datos -> Script en Perl -> Documento HTML

Texto plano -> ASP -> Documento XHTML

Autor -> Bloc de notas -> HTML

### Ventajas:

- Aumentar el tráfico del sitio web

- Ayudar a que se visite con frecuencia
- Favorecer posicionamiento
- Relaciones entre webs de la comunidad
- Permitir a otras personas añadir características a los servicios del sitio web
- Enriquecer internet impulsando tecnología semántica (trabajo con significado de los datos y no solo ocuparse de los datos) y favoreciendo la reutilización

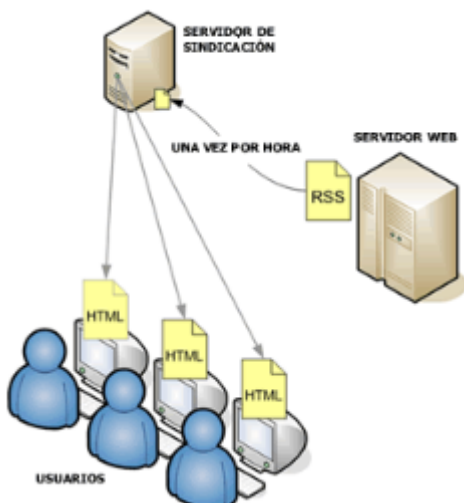
## Ámbitos de aplicación

Lo más habitual es el texto, por ser el formato de datos más habitual de los blogs.

Pero también se puede syndicar cualquier tipo de información (redifusión de vídeos - youtube)

Siempre se han syndicado contenidos y compartido todo tipo de información en XML. Se pueden ofrecer contenidos propios para que los muestre otra web aumentando el valor de la página que muestra el contenido y el valor del sitio web original ya que enlaza con él.

Permite desde el punto de vista la actualización profesional "estar al día" en temas relacionados con profesión, recibiendo noticias e informaciones en su blog o en su programa agregador de noticias.



## 3. Tecnologías de creación de canales de contenidos

Hay dos grupos:

**RSS:** (Really Simple Syndication): Parte de la familia de los formatos XML. Desarrollado para compartir la información que se actualiza con frecuencia entre sitios web. Se usa en conexión con sistemas de mensajería instantánea, la conversión de RSS en mensajes de correo o la capacidad de transformar enlaces favoritos del navegador en RSS. Ha sido desarrollado por tres organizaciones, dando lugar a siete formatos diferentes entre sí:

RSS 0.90. Estándar creado por Netscape en 1999. Especificación RDF de metadatos (titulares de otras webs)

RSS 0.91. Netscape. Simplificada de 0.90. Detenido por falta de éxito. Continuado por UserLandSoftware para blogs. (Que luego rechazan 1.0 por considerarlo complejo y siguen 0.92, 0.93, 0.94 con sintaxis incompleta y sin las normas XML)

RSS 1.0. A partir de 0.90. Es el más estable, permite definir cantidad mayor de datos.

RSS 2.0. Para subsanar los problemas de las versiones de UserLandSoftware.

**Atom** :Estándar propuesto por Atom Publishing Format and Protocol de la IETF en RFC4287.

Alternativa a RSS para evitar la confusión creada por estándares similares entre los que incluso había incompatibilidad. Convive con ellos. Se caracteriza por su flexibilidad. Permite más control sobre cantidad de información a representar en los agregadores.

## 4. Estructura de canales de contenidos

Un canal de contenidos se construye creando un fichero siguiendo especificaciones RSS o Atom, basadas en XML.

El fichero tendrá:

- Declaración de documento XML y definición de codificación empleada (UTF-8 preferiblemente)
- Canal en el que se determina el sitio web asociado a la fuente web a la que hace referencia el fichero
- Secciones: Cada una es una referencia a la web que contiene uno de los servicios que se van a ofrecer. Pueden incluirse tantas secciones como se quiera en un canal. No hay restricción respecto a la cantidad de canales de contenido que puede ofrecer un sitio web.

## 4.1. RSS

- Declaración del documento XML y codificación
- Versión RSS (ejemplar del documento)
- Definición del canal con elemento `<channel>` que debe contener elementos
- Subelementos de `<channel>`

Elemento	Definición
<code>&lt;title&gt;</code>	Título que se da al sitio web
<code>&lt;link&gt;</code>	Dirección web de la página asociada al fichero RSS
<code>&lt;description&gt;</code>	Breve comentario con finalidad del sitio
<code>&lt;language&gt;</code>	Idioma usado en el sitio
<code>&lt;item&gt;</code>	Secciones del canal

Los items son links a otros recursos, cada uno con descripción diferente. Los canales son usados en sistemas en los que el contenido puede estar segmentado en partes independientes que pueden estar enlazadas. Cada item debe incluir:

Elemento	Definición
<code>&lt;title&gt;</code>	Título del enlace al que se referencia (no tiene que coincidir con el canal)
<code>&lt;link&gt;</code>	URL de la página enlazada, que debe pertenecer al dominio del canal
<code>&lt;description&gt;</code>	Comentario que define el contenido

## Ejemplo RSS

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
 <!-- Se define el canal -->
 <channel>
 <title>Canal RSS</title>
 <link>http://www.mipagina.es</link>
 <description>Canal RSS de ejemplo</description>
 <language>es</language>
 <!-- Se definen los distintos items del canal-->
 <item>
 <title>El tiempo en Almería</title>
 <link>http://www.aemet.es/es/eltiempo/prediccion/municipios/almeria-
id04013</link>
 <description>Meteorología en Almería</description>
 </item>
 <item>
 <title>Noticias de interés</title>
 <link>http://desarrolloweb.com/de_interes</link>
 <description>Aquí podras encontrar todo tipo de noticias interesantes para los
desarrolladores web</description>
 </item>
</channel>
</rss>
```

## 4.2. Atom

- Declaración del documento XML y codificación
- Definición del canal con elemento `<feed>`, estándar de atom y lenguaje usado
- Subelementos de `<feed>`

Elemento	Definición
<code>&lt;title&gt;</code>	Título que se da al sitio web
<code>&lt;id&gt;</code>	Identificador del canal (URL)
<code>&lt;link&gt;</code>	enlaces que definen el canal. Son necesarios dos: Uno al fichero .atom (cuyo valor de rel será self) y otro al fichero web que oferta ese canal (rel=alternate)
<code>&lt;updated&gt;</code>	Fecha y hora de actualización con formato CCYY-MM-DDTHH:MM:SSZ (donde T es el separador entre la fecha y la hora y Z indica que la hora hace referencia al sistema de tiempo universal, esto es la hora zulú, o la hora del meridiano de Greenwich. Entonces para indicar que el canal se ha actualizado el 6 de febrero de 2010 a la 17:15 hora española tenemos que poner: 2010-02-06T16:15:00Z)
<code>&lt;author&gt;</code>	Autor del enlace (otros elementos como name o email)
<code>&lt;entry&gt;</code>	Cada sección del canal

Cada entry debe incluir:

Elemento	Definición
<code>&lt;title&gt;</code>	Título del enlace al que se referencia (no tiene que coincidir con el canal)
<code>&lt;id&gt;</code>	Identificador de la sección, debe ser única en el fichero
<code>&lt;link&gt;</code>	enlace a fuente de la sección (rel="alternate")
<code>&lt;updated&gt;</code>	Fecha y hora de actualización. Solo se modifica en casos significativos.
<code>&lt;author&gt;</code>	Autor del enlace
<code>&lt;summary&gt;</code>	Resumen del contenido del enlace

## Ejemplo Atom

```
<?xml version="1.0" encoding="utf-8"?>
<!--Se crea el canal, se determina el estándar a utilizar y el idioma del documento-->

<feed xmlns="http://www.w3.org/2005/Atom" xml:lang="es-es">
 <title type="text">LMSGI03 Tarea</title>
 <id>http://www.mipagina.es</id>
 <link rel="self" type="application/atom+xml"
href="http://www.mipagina.es/feed/canal_atom.atom" />
 <link rel="alternate" type="text/html" href="http://www.mipagina.es"/>
 <updated>2013-11-11T19:20:46Z</updated>
 <author>
 <name>Nombre y Apellidos</name>
 <email>nombre@dominio.com</email>
 </author>

 <!--Se definen las secciones que forman el canal-->
 <entry>
 <title>El tiempo en Almería</title>
 <link rel="alternate" type="text/html"
href="http://www.aemet.es/es/eltiempo/prediccion/municipios/almeria-id0401" />
 <updated>2013-11-12T19:19:46Z</updated>
 <id>http://www.aemet.es/es/eltiempo/prediccion/municipios/almeria-id0401</id>
 <author>
 <name>Nombre y Apellidos</name>
 <email>nombre@dominio.com</email>
 </author>
 <summary>Consulta el tiempo meteorológico de Almería</summary>
 </entry>
```



```

 <entry>
 <title>Noticias de interés web</title>
 <link rel="alternate" type="text/html"
href="http://desarrolloweb.com/de_interes"/>
 <updated>2013-11-12T19:17:55Z</updated>
 <id>http://desarrolloweb.com/de_interes</id>
 <author>
 <name>Nombre y Apellidos</name>
 <email>nombre@dominio.com</email>
 </author>
 <summary>Aquí podras encontrar todo tipo de noticias interesantes para los
desarrolladores web</summary>
 </entry>
</feed>

```

Miro en un periódico y veo en su código fuente:

```



































































<link rel="alternate" title="Últimas Noticias | La Opinión de Málaga" href="/rss/"
type="application/rss+xml" />

```

## La Opinión DE MÁLAGA

Es noticia: Polémica Comisaría Mercado de Teatinos Muerte turista Calendario Tren litoral 'Golden visa'

### PÁGINA RSS

Sección	RSS	Google	Bloglines	Yahoo	Netvibes	Windows
> Actividades extraescolares						
> Activos						
> Agenda						
> Andalucía						
> Andalucía						
> Andalucía						
> Asesinos natos						
> Audiencias						
> Axarquía						
> Becas						
> Borrador de la renta						

```
← → ↻ 🌐 laopiniondemalaga.es/rss/section/16647

This XML file does not appear to have any style information associated with it. The document tree is shown below.

<?xml version="1.0" encoding="UTF-8" ?>
<rss xmlns:media="http://search.yahoo.com/mrss/" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:atom="http://www.w3.org/2005/Atom" version="2.0">
 <channel>
 <title>
 <![CDATA[La Opinión de Málaga - Andalucía]]>
 </title>
 <link>
 <![CDATA[https://www.laopiniondemalaga.es/fotos/andalucia/]]>
 </link>
 <description>
 <![CDATA[La Opinión de Málaga - Andalucía]]>
 </description>
 <language>
 <![CDATA[es-es]]>
 </language>
 <copyright>
 <![CDATA[Copyright La Opinión de Málaga]]>
 </copyright>
 <atom:link href="http://www.laopiniondemalaga.es/rss/section/16647" rel="self" type="application/rss+xml"/>
 </channel>
 <item>
 <title>
 <![CDATA[La gala del 28-F y la entrega de Medallas de Andalucía 2024, en imágenes]]>
 </title>
 <link>
 <![CDATA[https://www.laopiniondemalaga.es/fotos/andalucia/2024/02/28/gala-28-f-entrega-medallas-98782524.html]]>
 </link>
 <description>
 <![CDATA[El Teatro de la Maestranza de Sevilla ha acogido el acto de entrega de las distinciones de Hijo/a Predilecto/a y las Medallas de Andalucía, que han reconocido a Dcoop, Sando, Danza Inv...
 Sevilla, los Romero de la Puebla, María Pérez García, Juan y Medio, el PAV, Eva Mª Loín, Álvaro Domecq, Manuel Titos, Policía Nacional, Cruz Roja de Andalucía, Jarcha y la Confederación Andaluz...
]]>
 </description>
 <dc:creator>
 <![CDATA[Manuel Murillo / EFE]]>
 </dc:creator>
 <guid isPermalink="true">
 <![CDATA[https://www.laopiniondemalaga.es/fotos/andalucia/2024/02/28/gala-28-f-entrega-medallas-98782524.html]]>
 </guid>
 <pubDate>
 <![CDATA[Wed, 28 Feb 2024 15:02:50 +0000]]>
 </pubDate>
 <enclosure url="https://estaticos-cdn.prensaiberica.es/clip/72677083-4b06-4028-b51d-d6638a3defb6_16-9-aspect-ratio_default_0.jpg" length="386215" type="image/jpeg"/>
 <media:content url="https://estaticos-cdn.prensaiberica.es/clip/72677083-4b06-4028-b51d-d6638a3defb6_16-9-aspect-ratio_default_0.jpg" type="image/jpeg" fileSize="386215" width="880" height="495"/>
 <media:title>
 <![CDATA[La gala del 28-F y la entrega de Medallas de Andalucía 2024, en imágenes]]>
 </media:title>
 <media:thumbnail url="https://estaticos-cdn.prensaiberica.es/clip/72677083-4b06-4028-b51d-d6638a3defb6_16-9-aspect-ratio_default_0.jpg" width="880" height="495"/>
 <media:keywords>
 <![CDATA[Medallas de Andalucía,28-F,Sevilla,28F]]>
 </media:keywords>
 </item>
 <item>
 <title>
 <![CDATA[Miércoles, 14 de febrero: los agricultores colapsan las carreteras de Andalucía]]>
 </title>
 <link>
 <![CDATA[https://www.laopiniondemalaga.es/fotos/andalucia/2024/02/14/miercoles-14-febrero-agricultores-colapsan-98162674.html]]>
 </link>
 <description>
 <![CDATA[]]>
 </description>
 </item>
</rss>
```

## 5. Validación

Cuando se crea un fichero fuente, hay que validarlo con algún validador.

Se da la dirección del fichero alojado y comprueban que lo pueden encontrar (que la URL es válida y no contiene errores).

Una vez validado, ofrecen una imagen del tipo XML o RSS de color naranja por lo general que se puede incluir en la dirección principal para enlazar a la dirección del fichero alojado en su dominio. Cuando un visitante pulse sobre este icono, accederá al contenido actual de la fuente y podrá navegar a las páginas que más le interesen.

Algunos servicios de validación también ofrecen imágenes que se pueden incluir en la página para que se compruebe que el canal es válido.

(W3C Feed Validation Services por URI o código, RSS Advisory Board)

## 6. Utilización de herramientas

Hay herramientas para crear y editar fuentes web con interfaces que simplifican al máximo el trabajo con canales de contenidos como PSPad editor con el que se puede:

- Trabajar con distintos estándares
- Importar CSV y HTML
- Editar HTML
- Editar XML e imágenes
- Actualizar fuentes vía FPT
- Exportar documentos RSS a HTML, CSV y Javascript

"Trabajar con algún editor de fuentes web permite, entre otras cosas, generar fuentes con cualquier tecnología de sindicación"

## 7. Directorios de canales de contenidos

El fichero generado y validado debe registrarse en un directorio de canal de contenidos.

Estos permiten que el fichero RSS esté disponible para cualquiera, facilita a usuarios la búsqueda de información ya que clasifican los ficheros RSS. Para ello se registra el fichero RSS en el directorio RSS, como cuando se registra un sitio en un motor de búsqueda.

Buscazoom RSS es un directorio de noticias y canales RSS. "Si tienes una web añade el feed RSS al directorio, aumenta tus visitas y ayúdanos a construir el mayor directorio de internetx es gratuito".

Recomendable codificación UTF-8 y uso de entidades XML que sustituyen a letras con tildes o ñ. `&acute;` en lugar de `á`; `&lt;` en lugar de `<`; `&gt;` en lugar de `>`,...

También puede proporcionarse el canal en Yahoo, Google...

## 8. Agregación

Se llama **agregador o lector de fuentes** a una aplicación de software para suscribirse a fuentes en formatos RSS y Atom. Agregador avisa a usuario de qué webs han incorporado contenido nuevo desde la última lectura y cuál es ese contenido.

Tipos de agregadores:

- **Herramientas de suscripción a canales vía RSS vía web o agregadores en línea:** Residen en algunos sitios web y se ejecutan a través de la web. Uno se tiene que dar de alta y dar de alta un perfil. Recomendable cuando no se accede a internet siempre desde el mismo ordenador (Netvibes, feedly, inoreader)
- **Herramientas de suscripción navegador web:** En el navegador web o programa de correo electrónico. Antes predeterminadas; ahora como plugin. (Softzone)
- **Herramientas de suscripción con programas de escritorio:** En el ordenador del usuario. Útil cuando se accede desde el mismo. Interfaz gráfica parecida a programa de correo electrónico. (FeedReader, RSSReader)
- **Herramientas de suscripción Podcast o podcasting:** El podcast es distribución de archivos multimedia mediante redifusión como RSS (tiene posibilidad de suscribirse y usar programas de descarga para que los usuarios lo escuchen y vean). Podcast de audio (archivos mp3, AAC). Podcast de vídeo (vodcast o videocast): Requieren "conexiones de gran ancho de banda (mp4, m4v). Pueden subirse y alojarse en... ..... Da igual. Esto tiene mil años.

## 9. Servidores

- Ordenador o máquina "al servicio" de otras. Suministra a los clientes todo tipo de información... Se da lo que se conoce como "Esquema cliente-servidor".
- Suelen ser algo más potentes que un ordenador normal
- Tipos: Correo, proxy, web, BBDD, clusters, dedicados...
- Los servidores web almacenan documentos HTML, imágenes, vídeos, textos,... Para crear una web debe contarse con un dominio y un servidor donde alojar los archivos. Recomendable al principio usare servidor local o de pruebas para no perder tiempo con el de pago mientras lo hacemos y para no tener un mal SEO (páginas 404 que vamos cambiando).
- El servidor local es un ordenador con aplicaciones instaladas para servidor de prueba

Para que el ordenador funcione como servidor online necesitaríamos:

- Sistema operativo (windows, MAC, linux)
- Apache. Aplicación madre que permite que sea un servidor
- MySQL. BBDD controlada por apache
- PHP. Tecnología de programación

De código abierto. Las puedo instalar por separado pero hay paquetes que las engloban todas como LAMP (Linux, Apache, MySQL, PHP); que en MAC es MAMP y en Windows WAMP. También hay XAMPP.

# UD 04 - DEFINICIÓN DE ESQUEMAS Y VOCABULARIOS EN XML

"¿Algún modo de garantizar la estructura de datos de los diferentes tipos de documentos XML?"

**XML** (*eXtensible Markup Language*, Lenguaje de Marcado eXtensible) es un lenguaje desarrollado por **W3C** (*World Wide Web Consortium*) que está basado en **SGML** (*Standard Generalized Markup Language*, Lenguaje de Marcado Generalizado Estándar).

- lenguaje de etiquetas
- utilizado para el **almacenamiento e intercambio de datos estructurados** entre diferentes equipos y de manera segura, fiable y sencilla
- **metalenguaje**, es decir, puede ser empleado para definir otros lenguajes, llamados dialectos XML (como XHTML, GML, MathML, SVG, RSS)
- compatible con protocolos como HTTP y URL, legible para los humanos
- extensible, adaptable, aplicable a gran variedad de situaciones
- orientado a objetos
- fácil de crear
- difusión asegurada porque cualquier procesador XML puede leer documento XML

Los datos carácter son los que forman el verdadero XML. El marcado puede ser tan rico como se quiera (detectar futuras necesidades y conseguir documentos fácilmente actualizables). Documentos XML pueden tener comentarios pero no son interpretados por XML. Estos se deben incluir entre `<!-- y --!>` y pueden estar en cualquier lugar salvo antes del prólogo o dentro de una etiqueta.

## Pasos para definir un documento XML

Etapas para definir:

- **Especificación de requisitos:** Recoger la estructura que tiene la información
- **Diseño de etiquetas:** Se crean etiquetas para representarla
- **Marcado de los documentos:** Se utilizan las etiquetas para crear documentos con contenido real. (Todo lo situado entre "<" y ">" o "&" y ";")

## 1. Las partes de un documento XML

Un documento XML básico está formado por PRÓLOGO Y EJEMPLAR :

- **Prólogo:** Informa al intérprete que va a procesar el documento de los datos que necesita para realizar su trabajo.  
Es opcional pero si se incluye debe ir obligatoriamente al inicio del documento.  
Tiene dos partes:
  - **La definición XML:** Va entre `<? ?>` No es obligatoria (en contraste con "de no ser así se genera error que impide que el documento sea procesado") pero, de incluirse, debe estar en la primera línea del documento y el carácter ' <' debe ser el primero de dicha línea (sin espacios en blanco).  
Posee los atributos `version` (obligatorio, indica versión XML utilizada aunque solo hayu 1.0. y 1.1 y la que se usa es la 1.0), `encoding` (opcional debe ir tras version, indica codificación de caracteres empleada, valor por defecto "UTF-8". Podría ser ISO-8859-1 (Caracteres europeos occidentales)) y `standalone` (es opcional, debe ir en último lugar, cuando está a "yes" indica que el documento es independiente de otros (de un DTD externo por ejemplo). Su valor por defecto es "no", o sea, que puede estar asociado a otros.).

(En los apuntes no viene esto): También se pueden indicar en el prólogo líneas de instrucciones de procesamiento (opcionales) para indicar cierta información al intérprete. Estas van también entre `<? ?>` . Por ej. se puede asociar un archivo CSS:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="estilo-animales.css"?>
<animales>
 <animal>
 <nombre>perro</nombre>
 <patas>4</patas>
 </animal>
 <animal>
 <nombre>pato</nombre>
 <patas>2</patas>
 </animal>
</animales>
...
```

Y el CSS, hará que el XML aparezca formateado

```
``css
nombre{color:blue;font-size:40px}
patas{color:red;font-size:22px}
```

- **La declaración del tipo de documento** (La declaración del tipo propiamente dicha es el nombre del ejemplar precedido de `<!DOCTYPE` y separado por un espacio Ej.: `<!DOCTYPE animales>`). Pero esta declaración permite al autor del XML definir restricciones y características del documento (asociar el documento a una definición de tipo DTD).
- **Ejemplar:** Parte más importante del documento. Contiene los datos reales del documento a procesar. El ejemplar es el elemento raíz de un documento XML. Es decir, formado por un elemento, el principal o raíz, que **debe ser único**. Ya que está compuesto de elementos estructurados en árbol. El ejemplar es el elemento raíz (raíz, única) y las hojas son los elementos terminales (los que no contiene más elementos). Los elementos **pueden contener atributos**. Se puede usar el marcador especial `<![CDATA[texto que se quiera]]>` para introducir en un documento texto que pueda contener caracteres como & o < o >. Se tomará como texto.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE ejemplo>
<ejemplo>
</ejemplo>
```

Si preguntan por "componentes de un documento XML": Prólogo OK, Ejemplar OK, Definición de codificación OK, Cabecera NO

Es recomendable mantener todo el criterio en el documento. Etiquetas en minúsculas. Para visualización óptima se anidarán los elementos indentando o tabulando en el código.

## 2. Sintaxis de documentos XML: Elementos vs Atributos.

### Namespaces: espacios de nombres

La estructura del documento XML viene dada por etiquetas de marcado.

**Elemento:** Grupo formado por etiqueta de apertura, etiqueta de cierre y contenido entre ambas. Siguen la estructura de árbol (pueden ser anidadas pero no entrelazadas).

El nombre de la etiqueta debe estar formado por letras mayúsculas o minúsculas, números, guiones y punto. No puede contener espacios y debe comenzar por una letra. El nombre xml está prohibido.

```
<etiqueta>valor</etiqueta>
```

Si está vacío:

```
<etiqueta></etiqueta>
<etiqueta/>
```

Todos los datos del documento XML deben pertenecer a un elemento. Pueden estar formados por otros elementos. Nombres de etiquetas deben ser autodescriptivos.

## Normas:

- Debe existir un elemento raíz y solo uno
- Todos los elementos tienen etiqueta de inicio y de cierre
- No puede cerrarse un elemento con otro que aún no se haya cerrado
- Nombres de etiquetas de inicio y de cierre deben ser iguales
- No puede contener cadena "]]>" por compatibilidad con SGML. Ni usar directamente `>`, `<`, `&`, `"`, `'`, `apostrofe`
- Para usar caracteres especiales deben usarse `&#D;` `&#H;` (números decimal y hexadecimal...)

**Atributos:** Características o propiedades que pueden añadirse a los elementos de un documento. Entrecomillados. No tienen que seguir un orden significativo. Son hojas, no tienen hijos. Tampoco pueden repetirse en un elemento. Si se usa, y se deja vacío, debe escribirse `atributo=""`. No puede haber espacios entre el nombre del atributo y el signo igual, ni entre el signo igual y el valor.

*Recomendado elemento si:* contiene subestructuras, es de tamaño considerable, su valor cambia con frecuencia, su valor va a ser mostrado.

*Recomendado atributo si:* no tiene subestructuras, es de pequeño tamaño, su valor raramente cambia, solo puede tener unos cuantos valores fijos, no va a mostrarse al usuario y solo sirve para guiar el procesamiento XML.

No se debe usar atributo para información susceptible de ser dividida. Solo para proporcionar información adicional sobre el elemento.

**Espacios de nombres:** Asigna un identificador único a elementos y atributos. Son definidos con `xmlns`. Se declaran indicando la URI (localización del conjunto del vocabulario del espacio de nombres al que hace referencia) `xmlns:"URI_namespace"`. Parecerá una URL pero no se usa como tal, solo se usa para que sea distinto a cualquier otro posible espacio de nombres.

Pueden usar, además de la URI, un prefijo que indique cuál es el vocabulario al que está asociada esa definición. `xmlns:prefijo="URI_namespace"`

- Permiten diferenciar elementos y atributos que comparten nombre (pero son de distintos vocabularios, con diferentes significados) evitando conflictos.
- Permiten agrupar los elementos y atributos relacionados de una aplicación XML para que el software lo reconozca con facilidad.

## Ejemplos:

```
<!-- Espacio de nombres -->
<nombre
xmlns="https://educacionadistancia.juntadeandalucia.es/EspacioNombres">Ejemplo</nombre>

<!-- Espacio de nombres con prefijo -->
<EN:nombre
xmlns:EN="https://educacionadistancia.juntadeandalucia.es/EspacioNombres">Ejemplo</EN:nombre
>

<!-- añadir mas de JSF, Icefaces por ejemplo -->
```

(Ojo: Facilita que el software localice y permite tener varios elementos homónimos pero no define los atributos que forman parte del documento, ni determina los elementos que forman parte)

**Entidades:** Forma de almacenar texto para usarlo en el documento. Se representan entre el símbolo `&` y el `;`. Pueden ser definidas por el desarrollador pero también hay 5 entidades predefinidas:

Entidad	Significado
<code>&amp;gt;</code>	Símbolo <code>&gt;</code>
<code>&amp;lt;</code>	Símbolo <code>&lt;</code>
<code>&amp;amp;</code>	Símbolo <code>&amp;</code>
<code>&amp;apos;</code>	Símbolo <code>'</code>
<code>&amp;quot;</code>	Símbolo <code>"</code>

Entidad	Significado
&numero;	Código
&#numero;	Código hexadecimal

Ejemplo: `<fórmula> a &gt; 3</fórmula>`

**Comentarios:** Está prohibido usar comentarios antes del prólogo y dentro de una etiqueta.

**Espacios en blanco:** XML elimina los espacios en blanco al inicio y al final del contenido de un elemento. Si se desea que se mantengan espacios en blanco sin procesar usar el atributo `xml:space = "preserve"`.

```
<text xml:space="preserve">
 Este es un ejemplo de texto
con espacios en blanco
preservados. </text>
```

## El documento bien formado

**Documento bien formado:** Es sintácticamente correcto, es decir, cumple con las reglas de creación de documentos XML ya definidas

**Documentos válidos:** Están bien formados y cumplen definición de estructura (DTD, XML Schema...)

Para estar bien formado deben verificarse las reglas sintácticas que define la recomendación del W3C para el estándar XML:

- El documento **ha de tener definido una declaración XML en el prólogo:**

*éstos serán los valores por defecto si no se incluye el prólogo, pero recordamos que es muy recomendable incluirlo, ya que algunos navegadores nos devolverán errores si no lo hacemos.*

- **Existe un único elemento raíz para cada documento:** es un solo elemento en el que todos los demás elementos y contenidos se encuentran anidados.
- Los elementos se organizan entre sí en **estructura jerárquica y no se permite el solapamiento de los elementos.**
- Hay que **cumplir las reglas sintácticas del lenguaje XML para definir los distintos elementos y atributos del documento.** Los resumimos a continuación:
  - El nombre de los elementos pueden tener como primer carácter [A-Z], [a-z] y "\_", y para el resto de caracteres, además de los citados: [0-9], "-" y ".".
  - **Las etiquetas de apertura y de cierre tienen que ser idénticas.** XML es sensible a las mayúsculas y minúsculas, por lo que, por ejemplo, ... sería incorrecto. Recordemos que se permiten elementos vacíos.
  - **Los valores de los atributos se escribirán siempre entre comillas dobles o simples.** Si queremos incluir un atributo que incluya alguno de esos caracteres, se hará de la siguiente forma: ":" (") y '('. Existen atributos reservados que no podremos usar, salvo para la finalidad en la que están reservados: "xml:", "xml:lang", "xml:space: default | preserve" y "xml:id"
  - **Los comentarios en XML se escribirán así:**

## 3. Declaración del tipo de documento, DOCTYPE

Es la primera línea de código requerida en el documento. Está formada por el elemento `<!DOCTYPE>`:

- **Declaración del tipo de documento propiamente dicha:** El tipo (nombre del ejemplar) precedido de `<!DOCTYPE` y separado por un espacio Ej.: `<!DOCTYPE animales>`
- **Las definiciones de tipo de documento:** Permite asociar al documento una **definición de tipo DTD** con las cualidades de este (tipos de los elementos y atributos, notaciones que pueden usarse, restricciones, valores por defecto) . Esto se hace mediante **declaraciones de marcado** que pueden ser:
  - **Declaraciones internas** (DTD Interno): Son exclusivas del documento y se procesan primero (sobrescriben a las externas). Se localizan dentro de unos corchetes que siguen a la declaración del tipo de documento.
  - **Declaraciones externas** (DTD Externo): Pueden (suelen) ser compartidas por varios XML. Se indican en un documento con extensión DTD (que puede estar en el mismo directorio que el XML).  
Importante considerar:

- La declaración de documento autónomo en el prólogo (`standalone`) debe haberse puesto negativa. (nunca positiva)
- El procesamiento del documento será más lento porque primero el procesador obtendrá las entidades y luego el documento.
- No se usan corchetes. Para localizar las declaraciones del tipo de documento externo se podría utilizar:

Una URI: `<!DOCTYPE nombre_ejemplar SYSTEM "url">` (SYSTEM indica que es definida externamente mediante un indicador del sistema).

Una URI y un identificador (id público), que el procesador XML podría usar para generar un URI alternativo (posiblemente basado en alguna tabla). Se usa suponiendo que hay un repositorio en el que se almacenan los DTDs y mediante un ID se recupera.

`<!DOCTYPE nombre_ejemplar PUBLIC "id_publico" "URI">`

Sería más correcto esto último: `<!DOCTYPE cine PUBLIC "filmoteca">`

`"http://cine.com/filmoteca.dtd">`, siendo filmoteca el nombre de la DTD

Ejemplos de declaraciones internas y externas (recuerda, pueden coexistir ambas)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE pelicula [
 <!ELEMENT pelicula (titulo)>
 <!ELEMENT titulo (#PCDATA)>
]>
<pelicula>
 <titulo>Titanic</titulo>
</pelicula>
```

(Si no pones standalone "yes", tomará por defecto el "no" y en principio no pasa nada malo pero algunos procesadores podrían asumir que necesita esa dependencia y ralentizarse o presentar alguna vulnerabilidad de seguridad)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE pelicula SYSTEM "cine.dtd">
<pelicula>
 <titulo>Titanic</titulo>
</pelicula>
```

(No haría falta poner standalone "no", por defecto lo toma)

## 4. Definiciones de tipo de documento, DTD

Es la descripción de la estructura y sintaxis de un documento XML o SGML (Lenguaje de marcado generalizado estándar)

Permite diseño común y consistencia entre los documentos que usan la misma DTD.

Un DTD permite incluir en el fichero XML ficheros binarios, además no se construyen en XML.

**Un XML válido es...:** Aquel que está bien formado y sigue las especificaciones que dicta la DTD.

Las restricciones de los DTD permiten...

- Especificar estructura
- Restricción de integridad referencial (normas que usan las bases de datos relacionales para garantizar que los registros de tablas relacionadas son válidas) mínima (ID, IDREF)
- Mecanismos de abstracción comparables a las macros (instrucción almacenada en la aplicación que la usa y ejecutada de forma secuencial) -> entidades
- Incluir documentos externos

Inconvenientes...

- Su sintaxis no es XML
- No soportan espacios de nombres
- No definen distintos tipos para los datos (solo hay tipo en elementos terminales)



- No permite secuencias no ordenadas
- No permite claves con varios atributos o elementos (no permiten formación de identificadores compuestos)
- Definido el DTD, no pueden añadirse nuevos vocabularios

Como se ha visto arriba cuando están definidas dentro del documento XML se ubican entre corchetes después del nombre del ejemplar en el elemento `<!DOCTYPE>`. Y cuando están definidas fuera del XML se utiliza la URI con extensión ".dtd" usando una URI o una URI y un IDENTIFICADOR.

Una DTD externa es ventajosa porque si es compartida por varios documentos evita tener que modificarlos todos; porque puede ubicarse en un servidor web permitiendo la validación del XML por cualquiera.

## 4.1. Declaraciones de elementos

Deben incluirse tantas declaraciones de elementos como tipos de etiquetas haya en el vocabulario.

Para cada elemento se utiliza la sintaxis `<!ELEMENT nombre (contenido)>` indicando:

- La palabra clave `ELEMENT`
- El nombre del elemento (etiqueta)
- Los contenidos que se permiten dentro del elemento (otros elementos, texto, combinación de ambos).

**Elemento sin contenido:** Se indica con `EMPTY` `<!ELEMENT br EMPTY>`

**Contenido libre:** Se indica con `ANY`. Estos elementos deben de haber sido declarados en el DTD.  
`<!ELEMENT empresa ANY>`

Por ejemplo:

```
<mascata>
 <nombre>Shiro</nombre> es mi mascota.
 Es un <tipo>gato</tipo> de
 <color>blanco</blanco>
</mascata>
```

(Curioso, pero se puede poner texto fuera de etiquetas: es bien formado y en este caso también es válido)

**Solo texto:** Se indica con `#PCDATA` `<!ELEMENT nombre (#PCDATA)>`. Solo puede contener datos de tipo carácter (exceptuando <, &, ], >).

**Solo otro elemento:** Se indica, debiendo declarar el otro también

```
<!ELEMENT importe (cantidad)>
<!ELEMENT cantidad (#PCDATA)>
```

**Secuencias de elementos:** Se indican en el mismo orden que aparecen en la declaración separando los elementos por una coma.

La repetición o la opcionalidad de elementos en la secuencia debe indicarse.

- Operador opción. **Cero o uno (?)** - Elemento opcional. De aparecer solo lo hará una vez
- Operador cero-o-mas. **Cero o más (\*)** - Elemento opcional. Si aparece puede hacerlo sin límite
- Operador un 111 o-o-mas. **Uno o más (+)** - Elemento obligatorio. Puede aparecer sin límite.

No es posible especificar cardinalidad del tipo "hasta dos" en el DTD  
Las opciones son uno, ninguno o más, uno o más

Operador de elección. **Elementos alternativos:** Contener un elemento y solo uno de entre todos los posibles, separándolos por una barra vertical |

**Combinaciones:** Puede mezclarse lo anterior para representar cualquier tipo de contenido  
No es solo "opciones entre elementos". Puede ser algo más complejo como:

Uno o varios padres o madres en una familia (o ninguno)

```
<!ELEMENT familia (((padre, madre?) | (madre?, padre?) | (padre, padre) | (madre, madre)), hijos)>
```

Mínimo un hijo/hija (apareciendo indistintamente uno u otro):

```
<!ELEMENT hijos ((hijo | hija)+)>
```

nombre y email o teléfono

```
<!ELEMENT nombre, (email | telefono)>
```

calle y número o teléfono e email

```
<!ELEMENT (calle, numero) | (telefono, email)>
```

email o direccion y descripcion

```
<!ELEMENT (email | (direccion, descripcion+))>
```

**Contenido mixto:** Puede especificarse texto y elementos. Por ejemplo:

```
<!ELEMENT parrafo ((#PCDATA | negrita | cursiva)*)>
<!ELEMENT negrita (#PCDATA)>
<!ELEMENT cursiva (#PCDATA)>
```

## 4.2. Declaraciones de atributos

Deben declararse después de haber declarado el elemento en el DTD y solo se puede hacer una por elemento. Con la declaración `<!ATTLIST elemento atributos>`

Se declaran con indicaciones de nombre, tipo y valor.

El tipo puede ser:

- `CDATA`: Texto

```
<!ATTLIST ejemplo color CDATA #REQUIRED>
```

- `NMTOKEN`: Solo letras dígitos y otros caracteres permitidos en nombres XML (letras ,números, caracteres : , \_ - .. Ni espacios en blanco, ni caracteres especiales.
- `NMTOKENS`: Varios tokens separados por espacios.
- `ENTITY`: Representa a una entidad dentro del documento (referencia a elemento externo o valor predefinido en el documento)
- `ENTITIES`: Múltiples entidades separadas por espacios
- `ID`: No puede repetirse
- `IDREF`: Referencia a elemento que tiene ID definido en el mismo documento. Restricción de integridad referencial.
- `IDREFS`: Múltiples referencias separados por espacios
- `NOTATION`: Define el tipo de notación que puede ser usada en un atributo particular. Se utiliza para describir el formato de los datos (como MIME type u otra notación)

```
<!DOCTYPE imagen [
<!ELEMENT imagen (ruta)>
<!ELEMENT ruta (#CDATA)>
<!ATTLIST imagen formato NOTATION (JPEG | PNG | PDF) #REQUIRED >
 <!NOTATION JPEG SYSTEM "image/jpeg">
 <!NOTATION PNG SYSTEM "image/png">
 <!NOTATION GIF SYSTEM "image/gif">
]>
<imagen formato="JPEG">
 <ruta>/ruta/a/imagen.jpg</ruta>
</imagen>
```

- **Enumeración:** Especificar lista de posibles valores poniéndolos separados por barra vertical y entre paréntesis. Solo puede contener letra, números, subrayado y guion no pudiendo contener

nada no permitido.

```
<!ATTLIST fecha dia_semana (lunes|martes|miércoles|jueves|viernes|sábado|domingo) #REQUIRED>
```

El valor son restricciones sobre el valor del atributo:

- Valor entre comillas. Opcional. Puede aparecer en el elemento o no. Si no aparece el procesador supone que tiene el valor indicado.
- **#REQUIRED**: Obligatorio y toma el valor que se le da
- **#IMPLIED**: Opcional y si no aparece, no se le da valor por defecto
- **#FIXED**: Obligatorio y debe llevar el valor indicado en la declaración

Ejemplo:

```
<!ELEMENT body ANY>
<!ATTLIST body
background-color CDATA "#FFFFFF"
href CDATA #REQUIRED
version CDATA #IMPLIED
padre CDATA #FIXED "html"
alineacion (izquierda | centro | derecha) "derecha">
```

### 4.3. Declaraciones de entidades

Se pueden definir entidades en el DTD que más tarde se referencien en el documento como:

```
<!ENTITY nombre contenido
```

Puede haber

#### Entidades generales

Se proporciona el texto a sustituir

```
<!ENTITY nombre "texto"
```

Ejemplo:

```
<!ENTITY onu "Organización de Naciones Unidas"
```

```
<parrafo>La &onu; es la más chula</parrafo>
```

El contenido de la entidad debe ser bien formado. Las entidades generales no pueden referenciarse desde dentro del DTD

#### Entidades externas

El contenido está en archivo externo al documento en el que se declara el DTD. Se relacionan a través de la URI de este último.

```
<!ENTITY nombre SYSTEM "url">
```

Ejemplo

```
<!ENTITY cabecera SYSTEM "cabecera.xml">
```

Declara la entidad cabecera y le asocia el contenido del archivo. El documento debe estar bien formado.

Cuando va a ser usada por varias aplicaciones es:

```
<!ENTITY nombre_entidad PUBLIC "identificador público formal" "camino hasta la DTD (uri)">
```

Si se quiere incluir ficheros binarios (que no sean analizados), se usa la palabra reservada **CDATA** en la definición de la entidad y debe asociarse a dicha entidad una **declaración de notación**.

## Entidades paramétricas

**Internas:** Pueden usarse dentro del DTD. Deben declararse de la forma:

```
<!ENTITY % nombre "valor">
```

Por ejemplo si en HTML todos los atributos deben llevar id, class y style puede hacerse:

```
<!ENTITY % common-attrs ("id ID #IMPLIED class ID #IMPLIED style CDATA #IMPLIED")>
<!ATTLIST html %common-attrs;>
<!ATTLIST html %common-attrs;>
`<!ATTLIST html %common-attrs;>
```

**Externas:** Permiten incluir en un DTD elementos externos (se divide la definición DTD en varios documentos)

```
<!ENTITY %persona SYSTEM "persona.dtd">
```

## 4.4. Declaraciones de notación

Para indicar cual debe ser la aplicación que ha de procesar un fichero binario asociado a un XML deben usarse declaraciones de notación... `<!NOTATION nombre SYSTEM aplicacion>`

Por ejemplo una notación gif referenciando al editor de GIF

```
<!NOTATION gif SYSTEM "gifEditor.exe">
```

Por ejemplo para asociar una entidad externa no analizada

```
<!ENTITY dibujo SYSTEM "imagen.gif" NDATA gif>
```

## 4.5. Secciones condicionales

Permiten incluir o ignorar partes de la declaración de un DTD. Para ello se usa:

- **INCLUDE**: Permite que se vea esa parte de la declaración. ( `
- **IGNORE**: Permite ocultar esa sección de declaraciones dentro del DTD. ( `

```
<![%datos_basicos; [
 <!ELEMENT persona (nombre, edad)>
]]>

<![%datos_ampliados; [
 <!ELEMENT persona (nombre, apellidos, edad, ciudad)>
]]>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE persona SYSTEM "persona.dtd" [
 <!ENTITY % datos_basicos "INCLUDE">
 <!ENTITY % datos_ampliados "IGNORE">
]>

<persona>
 <nombre>Elsa</nombre>
 <edad>23</edad>
</persona>
```

"Las sentencias condicionales permiten definir unos elementos u otros dentro del fichero XML en función de una determinada condición." --> Falso

## 4.6. Ejemplito sencillo de DTD

Para:

```

<?xml version="1.0"?>
<!DOCTYPE familias SYSTEM "familias.dtd">
<familias>
 <familia codigo="P01-12">
 <madre>
 <nombre>Maria</nombre>
 <apellido1>González</apellido1>
 <apellido2>López</apellido2>
 <dni>12345678A</dni>
 <edad>35</edad>
 </madre>
 <padre>
 <nombre>&FJ;</nombre>
 <apellido1>Pérez</apellido1>
 <apellido2>Rodríguez</apellido2>
 <dni>87654321B</dni>
 <edad>40</edad>
 </padre>
 <hijos>
 <hijo posicion="1">
 <nombre>Carlos</nombre>
 <apellido1>González</apellido1>
 <apellido2>Pérez</apellido2>
 <edad>10</edad>
 </hijo>
 <hijo posicion="2">
 <nombre>Sofía</nombre>
 <apellido1>Pérez</apellido1>
 <apellido2>González</apellido2>
 <edad>8</edad>
 </hijo>
 </hijos>
 </familia>
 <familia codigo="M02-24">
 <madre>
 <nombre>Lucía</nombre>
 <apellido1>Ruiz</apellido1>
 <apellido2>García</apellido2>
 <dni>98765432C</dni>
 <edad>38</edad>
 </madre>
 <madre>
 <nombre>Patricia</nombre>
 <apellido1>Gómez</apellido1>
 <apellido2>Fernández</apellido2>
 <edad>39</edad>
 </madre>
 <hijos>
 <hija posicion="1">
 <nombre>Marta</nombre>
 <apellido1>Ruiz</apellido1>
 <apellido2>Gómez</apellido2>
 <edad>12</edad>
 </hija>
 </hijos>
 </familia>
</familias>

```

<!--

1. En una familia puede haber un padre y una madre, o bien un solo padre, o una sola madre o ninguno de ellos. También hay posibilidad de que sean dos padres o dos madres.
2. El elemento padre y el elemento madre puede aparecer en cualquier orden
3. Posteriormente a los padres aparecerán los hijos.
4. El orden de aparición de hijo o hija es indiferente.
5. Una familia tiene al menos un hijo/a
6. Tanto el elemento padre, como madre, hijo o hija contienen los elementos: nombre, apellido (siempre dos), dni (opcional) y edad (opcional).
7. Limitar el tamaño de nombre y apellidos a una longitud de 30
8. El atributo posición toma valores del 1 al 30
9. El atributo código de la familia está formado por Letra mayúsculas y dos números, un guión y dos números, por ejemplo P01-12
10. La edad es un número entero positivo.

```

!-->
<!ENTITY FJ "Francisco José">
<!-- Entidad general-->
<!ENTITY % info_familia "(nombre, apellido1, apellido2, dni?, edad?)">
<!-- Las entidades paramétricas solo funcionan en DTD externo. Pueden hacerse al declarar
elementos o atributos -->
<!ELEMENT familias (familia*)>
<!ELEMENT familia (((padre, madre?) | (madre?, padre?) | (padre, padre) | (madre, madre)),
hijos)>
<!-- cubramos los casos
padre padre
madre madre
padre madre
madre padre
padre
madre
ninguno
-->
<!ELEMENT hijos ((hijo | hija)+)>

<!ELEMENT padre (%info_familia;)>
<!ELEMENT madre (%info_familia;)>
<!ELEMENT hijo (%info_familia;)>
<!ELEMENT hija (%info_familia;)>

<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido1 (#PCDATA)>
<!ELEMENT apellido2 (#PCDATA)>
<!ELEMENT dni (#PCDATA)>
<!ELEMENT edad (#PCDATA)>

<!ATTLIST familia codigo ID #REQUIRED>
<!ATTLIST hijo posicion NMTOKEN #REQUIRED>
<!ATTLIST hija posicion NMTOKEN #REQUIRED>

```

Más ejemplos en:

<https://www.youtube.com/playlist?list=PLM8XywipQpGA4Siojsb3yhSaZth5IFgSD>

<https://www.abrirlave.com/>

## 5. XML Schema

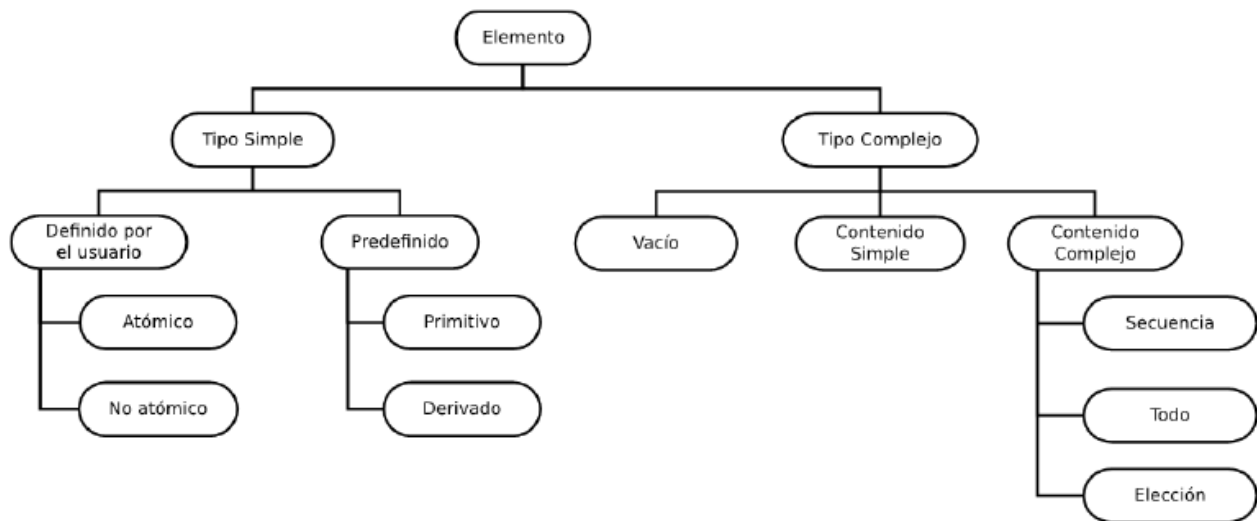
### 5.1. Introducción a XML Schema

- Lenguaje basado en XML
- Se usa para crear otros lenguajes basados en XML y modelos de datos
- Define los nombres de los elementos y los atributos del documento XML y especifica la estructura que deben seguir y el tipo de documento que puede haber dentro de cada elemento o atributo.
- Los documentos XML que intentan seguir un XML Schema se llaman instancias del esquema. Si siguen el esquema se dice que son válidos.

(No puede tenerse asociado a la vez un DTD y un XSD)

XML Schema es más potente que DTD porque:

- Los DTD no tienen tipos de datos predefinidos
- Los DTD no soportan tipos de datos derivados
- Los DTD tienen un control limitado de la cardinalidad
- Los DTD no soportan espacios de nombres ni formas simples de reutilizar o importar de otros esquemas



Es un lenguaje basado en XML que se usa para crear otros lenguajes basados en XML y modelos de datos.

## Algunos principios

- Los elementos pueden ser de tipo simple o complejo
- Los elementos de tipo simple solo contienen texto. No tienen ni hijos, ni atributos
- Los tipos predefinidos son simples
- Se pueden derivar nuevos tipos a partir de los simples, restringiéndolos.
- Los tipos simples pueden ser atómicos (cadenas, números) o no atómicos (listas)
- Elementos de tipo complejo pueden contener otros elementos, atributos y texto
- Elementos de tipo complejo tienen, por defecto, elementos hijos
- Elementos de tipo complejo pueden ser limitados para que tengan contenido simple (solo texto), pero también podrán tener atributos (cosa que los simples no)
- Elementos de tipo complejo pueden limitarse para no tener contenido pero sí atributos
- Tipos complejos pueden contener contenido mixto, mezcla entre texto y elementos hijos.

## Cómo empezar un XML Schema

El fichero `.xsd` debe estructurarse como:

myElement.xsd

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">
 <!-- Reglas XML Schema -->
</xs:schema>

```

`xmlns:xs="http://www.w3.org/2001/XMLSchema"` (<http://www.w3.org/2001/XMLSchema%22>)

- indica que los elementos y tipos de datos usados en el esquema vienen del espacio de nombres `"http://www.w3.org/2001/XMLSchema"` (<http://www.w3.org/2001/XMLSchema%22>). También especifica que los elementos y los tipos de datos que provengan de dicho espacio de nombres deben tener el prefijo `xs:`. **Este fragmento es el único obligatorio para que la definición sea correcta.** `targetNamespace="https://www.w3schools.com"` (<https://www.w3schools.com%22>).
- indica que los elementos definidos en el esquema pertenecen al espacio de nombres de `"https://www.w3schools.com"` (<https://www.w3schools.com%22>). Por defecto toma este valor. `xmlns="https://www.w3schools.com"` (<https://www.w3schools.com%22>).
- indica que el espacio de nombres por defecto es `"https://www.w3schools.com"` (<https://www.w3schools.com%22>). `elementFormDefault="qualified"`

- indica que cualquier elemento usado en una instancia XML que ha sido declarada con este esquema debe ser identificado con el espacio de nombres. Por defecto toma este valor.

## Asociación con documentos XML

Se asocia mediante un espacio de nombres al ejemplar del documento en el elemento raíz.

Debe indicarse el espacio por defecto de nombres en el documento (coincide con el declarado en el propio archivo del esquema), se indica el espacio de nombres correspondiente al esquema (siempre es la misma dirección de Internet) y se asocia a este espacio el prefijo xs (se puede elegir otro prefijo, pero no es nada conveniente).

En el caso de que en el Schema únicamente definamos el espacio de nombres `xmlns:xs="http://www.w3.org/2001/XMLSchema"` (<http://www.w3.org/2001/XMLSchema%22>), en el documento xml habrá que indicar:

```
<?xml version="1.0" encoding="UTF-8"?>
<documento xmlns:xs="http://w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="esquema.xsd">
</documento>
```

Y, si hace uso de espacios de nombres, pueden relacionarse cada uno con su XSD como:

```
<myElement xmlns:bk="http://example.com/books"
 xmlns:au="http://example.com/author"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://example.com/books book.xsd
 http://example.com/author author.xsd">
 <!-- Contenido del documento XML -->
</myElement>
```

## 5.2. Elementos de tipo simple

No tienen hijos, ni atributos. Pueden ser predefinidos o definidos por el usuario.

### Tipos simples predefinidos

| Tipo                  | Descripción                                                                                                                           | Tipo                      | Descripción                                                  |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------|---------------------------|--------------------------------------------------------------|
| <code>string</code>   | Caracteres                                                                                                                            | <code>gYear</code>        | Un año <code>yyyy</code>                                     |
| <code>boolean</code>  | 0, 1, true, false                                                                                                                     | <code>gMonthDay</code>    | Día en un año. (12 de noviembre) Format <code>--mm-dd</code> |
| <code>decimal</code>  | Real con decimales. Con punto                                                                                                         | <code>gDay</code>         | Día de cualquier mes de cualquier año <code>---dd</code>     |
| <code>float</code>    | Con notación científica                                                                                                               | <code>gMonth</code>       | Mes del año <code>--mm</code>                                |
| <code>double</code>   | Float pero con doble precisión                                                                                                        | <code>hexBinary</code>    | Hexadecimal. Longitud par.                                   |
| <code>duration</code> | Duración de tiempo en formato <code>PnYnMnDTnHnMnS</code> Número de años, meses, días, horas, minutos y segundos (P y T para separar) | <code>base64Binary</code> | Base 64. Alfanumericos, +, /, =                              |
| <code>dateTime</code> | Con formato <code>yyyy-mm-ddThh:mm:ss.nnnn:zzz</code>                                                                                 | <code>anyURI</code>       | URL, absoluta o relativa                                     |



| Tipo                    | Descripción                                   | Tipo                  | Descripción                                                           |
|-------------------------|-----------------------------------------------|-----------------------|-----------------------------------------------------------------------|
| <code>time</code>       | Hora en un día <code>hh:mm:ss.nnnn:zzz</code> | <code>QName</code>    | Nombre cualificado XML. Cadena que valga como nombre de elemento XML. |
| <code>date</code>       | Fecha en un día <code>yyyy-mm-dd</code>       | <code>NOTATION</code> | Notación                                                              |
| <code>gYearMonth</code> | U mes. Formato <code>yyyy-mm</code>           | ``                    |                                                                       |

(gDayMonth no existe. Cuidao)

## Tipos derivados predefinidos

Derivan de los primitivos, introduciendo alguna restricción.

normalizedString, token, language, NMTOKEN, NMTOKENS, Name, NCName, ID, IDREF, IDREFS, ENTITY, ENTITIES, integer, nonPositiveInteger, negativeInteger, long, int, short, byte, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte, positiveInteger.

Veamos algún ejemplo sencillo de uso

```
<?xml version="1.0"?>
<xs:schema xmlns="http://www.w3.org/2001/XMLSchema">
 <xs:element name="autor">
 <xs:complexType>
 <xs:sequence>
 <xs:element ref="nombre" type="xs:string">
 <xs:element ref="apellidos" type="xs:string">
 </xs:sequence>
 </xs:complexType>
 </xs:element>
</xs:schema>
```

## Tipos simples derivados por el usuario. Uso de facetas

Se definen con `xs:simpleType`

A base de tomar un tipo simple base y restringirlo `xs:restriction` con facetas

Las facetas disponibles son:

- `length`: Longitud exacta
- `minLength`: Longitud mínima
- `maxLength`: Longitud máxima
- `pattern`: Patrón mediante expresión regular
- `enumeration`: Conjunto de valores válidos (Solo uno de los dados)
- `whiteSpace`: Cómo deben tratarse los espacios en blanco
- `minInclusive`: Valor mínimo del rango incluido
- `minExclusive`: Valor mínimo del rango excluido
- `maxInclusive`: Valor máximo del rango incluido
- `maxExclusive`: Valor máximo del rango excluido
- `totalDigits`: Total de dígitos de un número
- `fractionDigits`: Número de dígitos decimales

### Elementos para hacer patrones.

| Patrón    | Significado                                   |
|-----------|-----------------------------------------------|
| [A-Z a-z] | Letra.                                        |
| [A-Z]     | Letra mayúscula.                              |
| [a-z]     | Letra minúscula.                              |
| [0-9]     | Dígitos decimales.                            |
| \D        | Cualquier carácter excepto un dígito decimal. |
| (A)       | Cadena que coincide con A.                    |
| A B       | Cadena que es igual a la cadena A o a la B.   |

### Elementos para hacer patrones.

| Patrón  | Significado                                          |
|---------|------------------------------------------------------|
| AB      | Cadena que es la concatenación de las cadenas A y B. |
| A?      | Cero o una vez la cadena A.                          |
| A+      | Una o más veces la cadena A.                         |
| A*      | Cero o más veces la cadena A.                        |
| [abcd]  | Alguno de los caracteres que están entre corchetes.  |
| [^abcd] | Cualquier carácter que no esté entre corchetes.      |

Ejemplo, restricción de longitud mínima y máxima de contraseñas

```
<xs:complexType>
 <xs:sequence>
 <xs:element name="PW" type="password"/>
 </xs:sequence>
</xs:complexType>
<xs:simpleType name="password">
 <xs:restriction base="xs:string">
 <xs:minLength value="8"/>
 <xs:maxLength value="12"/>
 </xs:restriction>
</xs:simpleType>
```

Podría hacerse con un patrón regex: `<xs:pattern value="[A-Za-z_]{6,12}"/>`

## Enumeraciones

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:simpleType name="salario">
 <xs:restriction base="xs:decimal">
 <xs:minInclusive value="10000"/>
 <xs:maxInclusive value="90000"/>
 <xs:fractionDigits value="2"/>
 <xs:totalDigits value="7"/>
 </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="puestoTrabajo">
 <xs:restriction base="xs:string">
 <xs:enumeration value="Gestor de Ventas"/>
 <xs:enumeration value="Vendedor"/>
 <xs:enumeration value="Recepcionista"/>
 <xs:enumeration value="Desarrollador"/>
 </xs:restriction>
 </xs:simpleType>
 <xs:element name="empleado">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="salario" type="salario"/>
 <xs:element name="puesto" type="puestoTrabajo"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
```

## Espacios en blanco

Al `xs:string` se le puede restringir el espacio en blanco con `xs:whiteSpace` indicando los valores `preserve` (se queda como está), `replace` (todos los tabuladores, saltos de línea... se reemplazan por espacios en blanco), `collapse` (todos se reemplazan por un único espacio en blanco y se eliminan espacios al inicio y al final)

## Tipos no atómicos

## Listas

Se ponen con `<xs:list itemType="">` Se pueden tener listas de enteros o de fechas. No deben confundirse con las enumeraciones (valor único). Estas representan uno o más valores dentro del elemento.

```
<xs:simpleType name="listaFechas">
 <xs:list itemType="xs:date"/>
</xs:simpleType>
```

Resultará algo como esto:

```
<diasVacaciones>2006-08-13 2006-08-14 2006-08-15</diasVacaciones>
```

## Uniones

Agrupaciones de tipos permitiendo que el valor de un elemento sea más de un tipo.

```
<xs:simpleType name="evento">
 <xs:union memberTypes="carrera gimnasia"/>
</xs:simpleType>
```

## Elementos simples globales vs locales

Cuando se declaran elementos globales estos pueden ser referenciados a lo largo del esquema haciéndolo más modular y fácil de mantener.

El inconveniente es que deben tener un nombre único.

Los elementos globales son aquellos en los que la declaración del elemento es un hijo directo del elemento raíz `<xs:schema>`.

Cuando se quiere hacer uso de él se le llama con el atributo ref.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="nombre" type="xs:string"/>
 <xs:element name="apellidos" type="xs:string"/>
 <xs:element name="autor">
 <xs:complexType>
 <xs:sequence>
 <xs:element ref="nombre"/>
 <xs:element ref="apellidos"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
</xs:schema>
```

## Valores por defecto

Un elemento que no tiene hijos, puede tener un valor por defecto. Para ello se usa el atributo `default`. Este valor solo se usará cuando el elemento aparezca en la instancia y no tiene contenido.

```
<xs:element name="puesto" type="xs:string" default="Vendedor">
```

## Valores fijos

Se obliga a que, si aparecen una instancia, contengan el valor indicado:

```
<xs:element name="estado" type="xs:string"
 fixed="actualizado" minOccurs="0"/>
```

## 5.3. Elementos de tipo complejo

Son aquellos que tienen atributos, elementos hijos o una combinación de ambos.

No es necesario declarar de forma explícita que un elemento simple es simple; pero sí que es necesario indicar que un elemento complejo es complejo con el elemento `xs:complexType`

El modelo de contenido de un elemento complejo puede ser:

- `xs:sequence`: Elementos deben aparecer en el orden especificado
- `xs:all`: Elementos deben aparecer pero el orden no es importante
- `xs:choice`: Solo uno de los elementos debe aparecer

```
<xs:element name="nombreElemento">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="ElementoHijo1" type="xs:string"/>
 <xs:element name="ElementoHijo2" type="xs:string"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="nombreElemento">
 <xs:complexType>
 <xs:all>
 <xs:element name="ElementoHijo1" type="xs:string"/>
 <xs:element name="ElementoHijo2" type="xs:string"/>
 </xs:all>
 </xs:complexType>
</xs:element>

<xs:element name="nombreElemento">
 <xs:complexType>
 <xs:choice>
 <xs:element name="ElementoHijo1" type="xs:string"/>
 <xs:element name="ElementoHijo2" type="xs:string"/>
 </xs:choice>
 </xs:complexType>
</xs:element>
```

Los elementos de tipo complejo pueden contener otros elementos de tipo complejo:

```
<xs:element name="nombreElemento">
 <xs:complexType>
 <xs:choice>
 <xs:element name="ElementoHijo1" type="xs:string"/>
 <xs:element name="ElementoHijo2">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="ElementoNieto1" type="xs:string"/>
 <xs:element name="ElementoNieto2" type="xs:string"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 <xs:element name="ElementoHijo3" type="xs:string"/>
 </xs:choice>
 </xs:complexType>
</xs:element>
```

### Restricciones de ocurrencia

En el `xs:element` Puede ponerse ocurrencias infinitas `maxOccurs="unbounded"`

Si `maxOccurs="unbounded"` es de 1 a infinito

Si `minOccurs="0" maxOccurs="unbounded"` es de 0 a infinito

Si `minOccurs="0"` se pone entre 0 y 1 (opcional)

## Contenido mixto

Un elemento puede contener tanto los elementos hijos como texto. Esto se hace indicando

`mixed="true"`

```
<xs:element name="elementoPadre">
 <xs:complexType mixed="true">
 <xs:sequence>
 <xs:element name="elementoHijo1" type="xs:string"/>
 <xs:element name="elementoHijo2" type="xs:string"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

## 5.5. Atributos en XML Schema

### Elementos vacíos con atributos

Se declara como elemento complejo, solo incluyendo el atributo.

### Elementos con contenido complejo con atributos

Se declaran después del modelo de grupos del elemento

### Elementos con contenido simple con atributos

Se declaran usando `xs:simpleContent` y extendiendo el elemento usando `xs:extension` que debe especificar el tipo del contenido simple usado en el atributo base.

```
<xs:element name="nombreElemento">
 <xs:complexType>
 <xs:simpleContent>
 <xs:extension base="xs:string">
 <xs:attribute name="nombreAtributo" type="xs:string"/>
 </xs:extension>
 </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

### Restringiendo los valores de los atributos

De la misma forma que los elementos

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="libro">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="titulo" type="xs:string"/>
 <xs:element name="autor">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="nombre" type="xs:string"/>
 </xs:sequence>
 </xs:complexType>
 <xs:attribute name="titulo">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:enumeration value="Sr."/>
 <xs:enumeration value="Sra."/>
 <xs:enumeration value="Dr."/>
 </xs:restriction>
 </xs:simpleType>
 </xs:attribute>
 </xs:element>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
</xs:schema>
```

```

 </xs:simpleType>
 </xs:attribute>
 </xs:complexType>
 </xs:element>
 </xs:sequence>
 </xs:complexType>
</xs:element>

```

### Valores por defecto y requiriendo atributos

Para poner uno por defecto se usa `default`

Para fijarlo se usa el atributo `fixed`

Para requerirlo se pone el valor `use:"required"`

## 5.4. Claves en XML Schema

### Unicidad

Puede usarse `<xs:unique>` para reforzar la unicidad.

Tiene dos elementos hijo:

- `<xs:selector>` tiene un atributo `xpath` con la expresión que referencia a los elementos afectados por la restricción
- `<xs:field>` Atributo `xpath` que especifica qué parte del elemento debe ser única

### Claves

El uso de claves y referencias a claves se puede hacer mediante `xs:key` y `xs:keyref`

Ambos contienen también los elementos `xs:selector` y `xs:field`

El elemento `xs:key` se usa para identificar los elementos que van a ser referenciados por los elementos especificados con `xs:keyref`

Este mecanismo asegura que:

- Los valores de claves deben ser únicos. No puede haber dos con el mismo valor.
- Debe haber integridad referencial. No puede haber referencia a una clave por un valor que no exista entre las claves.

## 5.5. Documentación del schema

Los analizadores no garantizan que los comentarios no se modifiquen al procesar los documentos, luego estos podrían perderse.

Para documentación, XML Schema tiene definido un elemento `<xs:annotation>` para guardar información adicional.

Además puede contener:

- `<xs:documentation>` Puede contener elementos de esquema y elementos XML bien estructurados.
- `<xs:appinfo>` Parecido a `<documentation>` aunque inicialmente se pensó que `<documentation>` fuese legible para usuarios y que este guardase información para los programas.

## 6. Creación y validación

- Notepad++
- Editix XML
- XML Copy Editor
- Netbeans
- Utilidades online

# UD 05 - CONVERSIÓN Y ADAPTACIÓN DE DOCUMENTOS XML

Con frecuencia es necesario transformar la información de un documento XML en otro.

Para transformar los documentos entran en juego tres especificaciones:

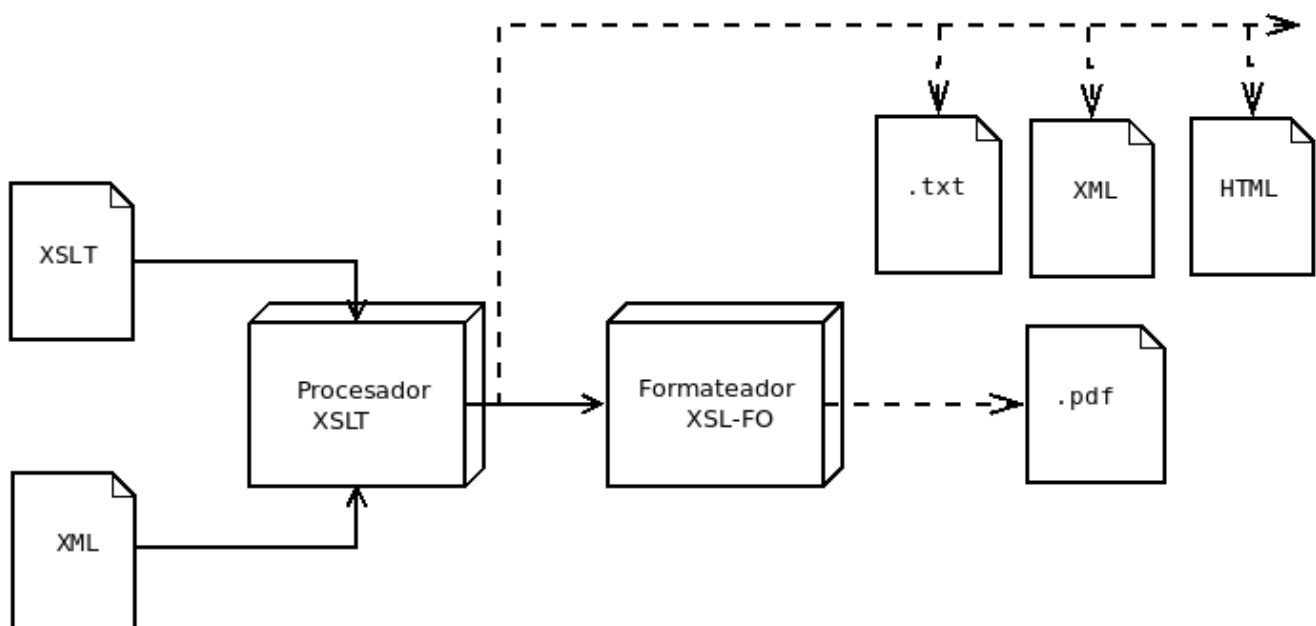
- **XSL** o **XSLT** (eXtensible Stylesheet Language Transformations): Definir modo de transformar un XML en otro
- **XSL-FO** (eXtensible Stylesheet Language Transformations - Formatting Objects): Transformar XML en formato legible e imprimible por una persona, como un PDF
- **Xpath**(Ruta X): Acceso a componentes de documento XML

La parte de transformaciones ganó en importancia y se llega a la terminología actual que comprende a las anteriores: **XSL** (eXtensible Stylesheet Language).

Es un lenguaje que interpreta hojas de estilo.

En 1997 se propone a W3C un lenguaje basado en XML que especificase el formato de documentos XML (como CSS a HTML), separando contenido de presentación. Se empieza a desarrollar XSL (con formato y contenido junto en el mismo documento). Para mantener la independencia se tomó como criterio que el documento con formato se generaría a partir del documento con los contenidos (se desarrolla XSLT; obteniendo por el camino XPath para localizar partes de un XML; observando el potencial de XSLT para transformar en cualquier otro contenido XML, lazando XSL-FO. XSLT y Xpath se siguen usando y se siguieron lanzando especificaciones desgajadas de XLS, que, a su vez, se renombró a XSL-FO. Este no ha tenido mucho éxito ya que es complejo y hay otros estándares CSS que hacen lo mismo.)

Para realizar las transformaciones se usan **procesadores XSL**. A ellos se les indica el archivo de entrada XML y XSL. Si no tienen errores, generan el archivo deseado.



## 1. XPath

Lenguaje, no basado en XML, que permite localizar o acceder a una parte de un documento XML

- Se basa en una relación de parentesco entre los nodos del documento: representación del documento XML llamada árbol de nodos o modelo de datos XPath
- Inicialmente diseñado para ser usado con XSLT y XPointer. Hoy día es usado en XSLT, XML Schema, XQuery, Xlink, Xpointer, Xforms...
- Su notación es **similar a las rutas de los archivos**, salvo que XPath está diseñado para selecciones múltiples.

El documento XML en primer lugar debe ser procesado por un analizador o parser XML que:

- verifica que el documento XML está bien formado
- lo valida contra el DTD / XSD correspondiente
- construye el árbol de nodos

## 1.1. Modelo de datos de XPath: El árbol de nodos

Un árbol es una estructura que representa una serie de elementos (nodos) unidos por líneas (vértices). Entre dos nodos cualesquiera solo ha y un único camino y ninguno de ellos forma un bucle. Tiene forma de copa de árbol invertida.

Hay siete distintos tipos de nodos.

- **Nodo raíz:** Primer nodo del árbol. No tiene padre. No debe confundirse con el elemento raíz del documento XML. Tiene como hijos al ejemplar (elemento raíz del XML) y, en su caso, los comentarios e instrucciones que formen parte del prólogo XML. Se indentifica con "/".
- **Nodos elemento:** Hay uno por cada elemento XML. Tienen un solo padre que puede ser otro nodo elemento o el nodo raíz. Pueden tener un identifiicador único, para lo cual debe estar declarado de tipo ID en su DTD o en el XML Schema asociado.
- **Nodos atributo\*:** Almacenan los atributos del documento XML. El nodo atributo está asociado a un único nodo elemento que es padre de este. Desde el punto de vista del nodo elemento no se considera a sus atributos como nodos hijos. Los nodos atributos son nodos hoja, no pueden tener nodos hijos.
- **Nodos texto** (o contenido): Almacenan los valores alfanuméricos de los contenidos de los elementos XML. Son nodos hoja (sin hijos). Los valores de los atributos no se almacenan en un nodo texto.
- **Nodos de comentario y Nodo de instrucciones de proceso.** Se generan para elementos con comentario e instrucciones de proceso. Son hijos del elemento en el que aparezcan o del nodo raíz si están situados fuera del elemento raíz. Por estos elementos el nodo raíz del modelo no coincide con el elemento raíz del documento ya que puede haber comentarios o instrucciones fuera del raíz.
- **Nodo espacio de nombres:** Cada nodo elemento puede tener un conjunto asociado de nodos espacios de nombres, uno para cada uno de los distintos prefijos de espacio de nombres incluyendo, si es el caso, el espacio de nombres por defecto. Tiene un funcionamiento parecido a los atributos. El nodo espacio de nombres tiene un único nodo elemento como padre. Desde el punto de vista del nodo elemento no son considerados como nodos hijos de este. Son nodos hoja.

(Las etiquetas permiten estructuras la información del documento XML pero no son consideradas nodos)

---



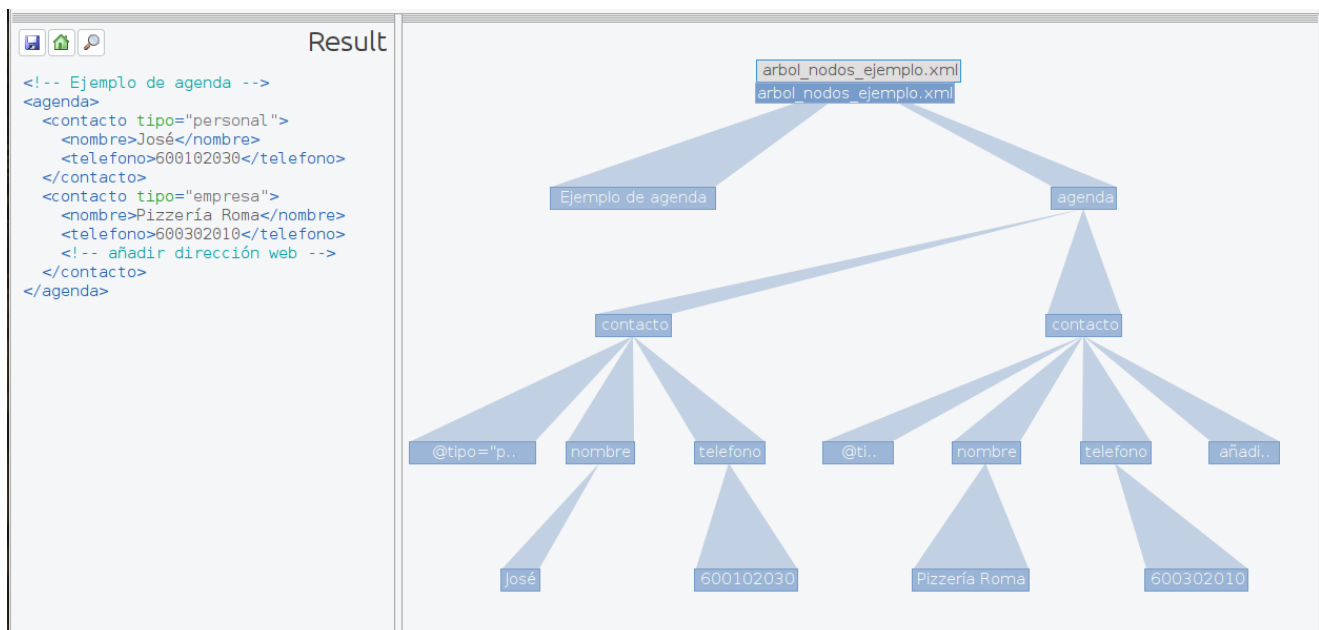
Ejemplo:

Si partimos del siguiente XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Libros de una biblioteca -->
<biblioteca>
 <libro isbn="11111111" prestamo="si">
 <titulo>El nombre del viento</titulo>
 <autor>Patrick Rothfuss</autor>
 </libro>
 <libro isbn="22222222" prestamo="no">
 <titulo>El nombre de la rosa</titulo>
 <autor>Umberto Eco</autor>
 </libro>
 <libro isbn="33333333">
 <titulo>El nombre de los nuestros</titulo>
 <autor>Lorenzo Silva</autor>
 </libro>
</biblioteca>
```

Tendríamos el siguiente modelo de datos:

```
Raiz
+--Comentario: Libros de una biblioteca
+--Elemento: biblioteca
+----Elemento: libro
+-----Atributo: isbn: 11111111
+-----Atributo: prestamo: si
+-----Elemento: titulo
+-----Texto: El nombre del viento
+-----Elemento: autor
+-----Texto: Patrick Rothfuss
+----Elemento: libro
+-----Atributo: isbn: 22222222
+-----Atributo: prestamo: no
+-----Elemento: titulo
+-----Texto: El nombre de la rosa
+-----Elemento: autor
+-----Texto: Umberto Eco
+----Elemento: libro
+-----Atributo: isbn: 33333333
+-----Elemento: titulo
+-----Texto: El nombre de los nuestros
+-----Elemento: autor
```



Observa que del nodo raíz (que no corresponde con el elemento raíz del XML) cuelga el ejemplar y el comentario. También que los valores de los atributos no aparecen en nodos texto sino junto al identificador del atributo en el nodo atributo (identificador y valor).

### 1.1.1. Las relaciones entre nodos

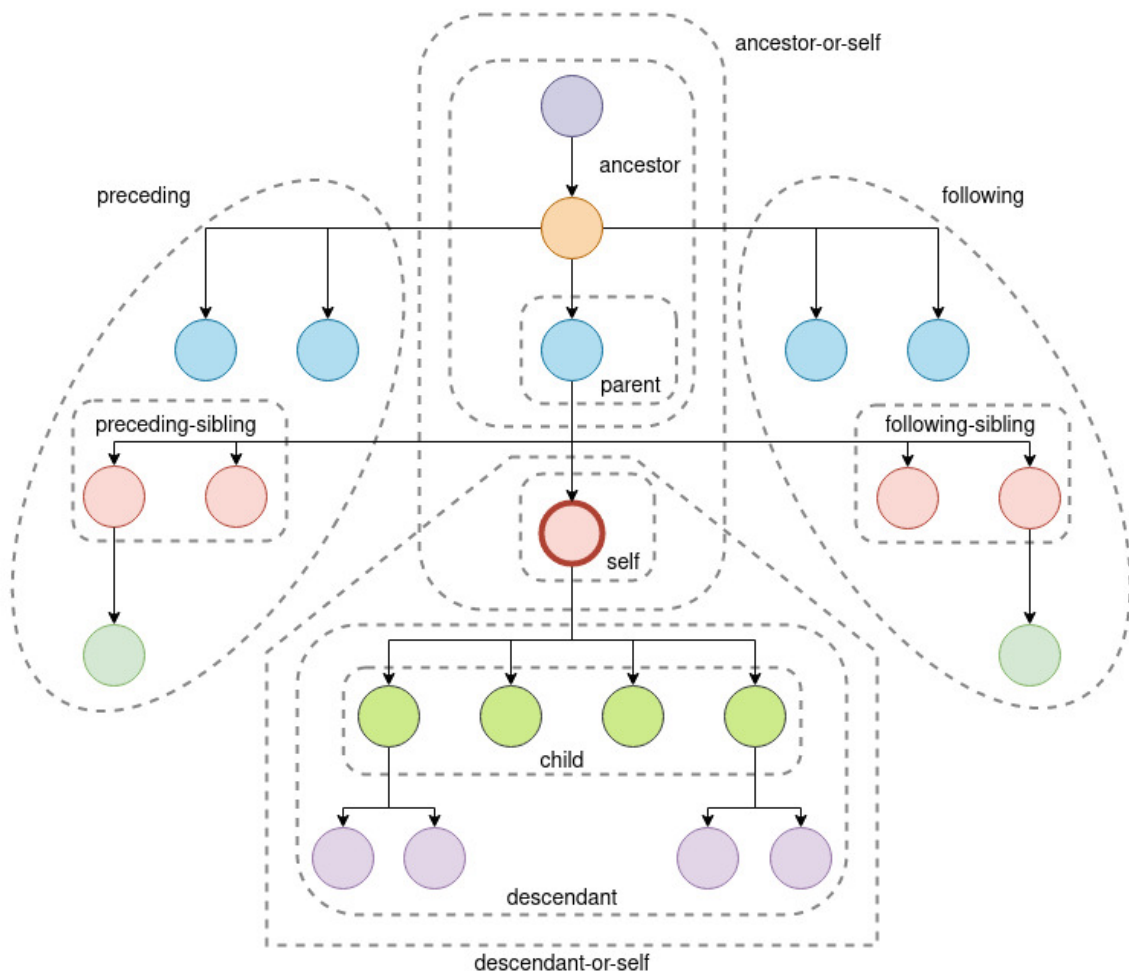
En primer lugar definamos:

- **Nodo actual:** El que se menciona al evaluar una expresión XPath
- **Nodo contexto:** Cada expresión está formada por subexpresiones que se evalúan antes de resolver la siguiente. Los nodos obtenidos tras evaluar una expresión, usada para evaluar la siguiente son el nuevo contexto. -> Es el nodo en el que nos posicionamos para establecer una relación.
- **Tamaño del contexto:** Número de nodos que se evalúan en un momento dado en una expresión XPath.

Hay trece relaciones entre nodos que serán usadas como ejes en los pasos de localización.

#### EJES:

- **self:** Nodo de contexto
- **child:** Hijos del nodo de contexto
- **descendant:** Hijos del nodo de contexto y todos sus descendientes
- **descendant-or-self:** Nodo contexto y sus descendientes
- **parent:** Padre del nodo contexto, si lo hay
- **ancestor:** Antecendos del nodo de contexto: Padre, padre de su padre,... Incluye al nodo raíz salvo que el nodo contexto sea el nodo raíz.
- **ancestor-or-self:** Nodo de contexto y sus ancestros. Incluirá siempre al nodo raíz.
- **preceding:** Nodos del mismo documento que el nodo contexto que están antes de este según el orden del documento excluyendo ancestros, nodos atributo y nodos de espacios de nombres
- **preceding-sibling:** Hermanos precedentes del nodo contexto; Si el nodo contexto es un nodo atributo o un nodo espacio de nombres este eje está vacío.
- **following:** Todos los nodos del mismo documento que el nodo contexto que están después de este según el orden del documento, excluyendo los descendientes y los nodos atributo y nodos de espacios de nombres
- **following-sibling:** Hermanos del nodo contexto. Si el nodo contexto es un nodo atributo o nodo espacio de nombres, este eje está vacío.



- **attribute:** Atributos del nodo contexto. Es vacía si el nodo contexto no es un nodo elemento (otros no pueden tener atributos)
- **namespaces:** Nodos espacio de nombres del nodo contexto. Relación será vacía si el nodo contexto no es nodo elemento (otros no pueden tener espacios de nombres)

## 1.2. Expresiones de la sintaxis de XPath y su resultado

Es parecida a la que se usa en los árboles de directorios y archivos de los sistemas operativos.

Se tiene una versión completa:

```
descendant-or-
self::curso[position()=1]/child::grupo[position()=2]/child::alumno[last()]/child::nombre/child::node()
```

Y una **versión simplificada** (Que es la que habitualmente se usa):

```
//curso[1]/grupo[2]/alumno[last()]/nombre/text()
```

En la expresión XPath se pueden usar llamadas a funciones, operaciones matemáticas y operaciones lógicas.

El camino de localización XPath se evalúa devolviendo un resultado que puede ser:

- **Conjunto de nodos (node-set).** Una lista de nodos. El orden en el que aparecen en la lista es el mismo en que aparecen en el documento. Se devuelve cuando los operadores usados seleccionan nodos del modelo. Se considera que todos los elementos del node-set son hermanos, independientemente de lo que fuesen originalmente. Los subárboles de un nodo no se consideran elementos del conjunto (hijos de los nodos del node-set, que son accesibles) Los nodos pueden ser de 7 tipos: Elemento, Atributo, Texto, Espacio de nombres, Instrucción de procesamiento, Comentario, Raíz, Booleano, Número y Cadena.
- **boolean:** Valor verdadero o falso, devuelto con operadores lógicos o de comparación
- **number:** Un número en punto flotante. Se devuelve con operadores numéricos.
- **string:** Cadena de caracteres. Cuando se seleccionan nodos de texto, comentarios o atributos.

Existen como palabras reservadas:

- Los ejes: `ancestor`, `ancestor-or-self`, `descendent`, `descendent-or-self`, `following`, `following-sibling`, `namespace`, `parent`, `preceding`, `preceding-sibling`, `self`
- Los selectores de nodos: `node()`, `text()`, `comment()`, `processing-instruction()`
- Operaciones lógicas: `and`, `or`, `not()`
- Operaciones matemáticas: `div`, `mod`

Los siguientes símbolos tienen una función definida:

- Agrupación de operaciones con paréntesis `()`
- Predicados con corchetes `[]`
- Abreviatura elemento actual `.`
- Abreviatura elemento padre `..`
- Abreviatura atributo `@`
- Todos los tipos de nodos `*`
- Separador eje-selector `::`
- Separador de pasos de localización `/`
- Abreviación del paso `descendant-or-self::node()` `//`
- Referencia a variable `$`
- Unión de conjuntos de nodos `|`
- Coma `,`
- Operaciones lógicas: `'='`, `'!='`, `'<'`, `'>'`, `'<='`, `'>='`
- Operaciones matemáticas `+`, `-`, `*`

También se usan:

- Nombres cualificados de los identificadores del documento XML
- Nombres de las funciones
- Nombres de referencias a variables (anteponiendo \$)
- Números y literales (Con comillas simples o dobles, posibilidad de anidar alternando comillas)

Ejemplo: Alumnos de 2 ESO A

```
/child::colegio/child::curso[@nivel="2"]/child::grupo[@orden="A"]/child::alumno
//colegio/curso[@nivel="2"]/grupo[@orden="A"]/alumno
```

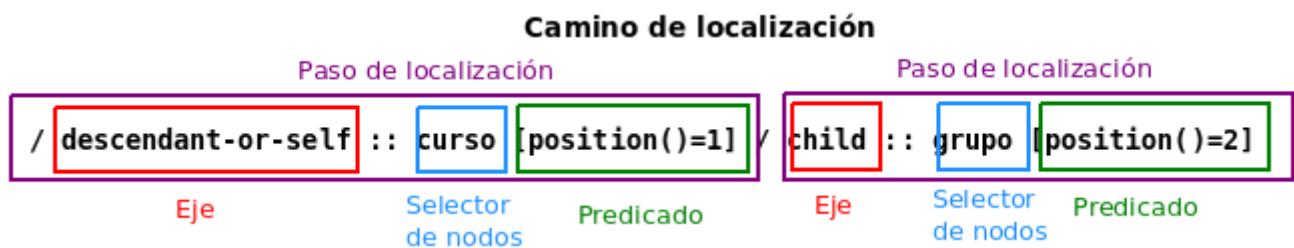
Ejemplo: Alumnas llamadas Ana

```
descendant-or-self::alumno[nombre="Ana"]
//alumno[nombre="Ana"]
```

Ejemplo: Nombre del con la nota más alta

```
descendant-or-self::alumno[child::nota_media=max(/descendant-or-
self::nota_media)]/child::nombre
descendant-or-self::alumno[nota_media=max(//nota_media)]/nombre
```

### 1.3. Caminos de localización y pasos de localización



- Los **caminos de localización** son la ruta que hay que seguir por el árbol de datos para localizar un nodo. No son lo más común en XPath, pero sí lo más importante.
- Están compuestos por **pasos de localización** separados entre sí por `/`.
- Los pasos de localización se separan en **eje** y **selector de nodos** por `::`. En algunos pasos de localización hay instrucciones entre corchetes `[]` después del selector de nodos que reciben el nombre de **predicados**

Los caminos de localización pueden ser:

- Absolutos: Comienzan en el nodo raíz. Fácilmente reconocible por empezar con la barra simple: `/child::colegio/child::curso/child::grupo/attribute::*`
- Relativo: Relativo al nodo contexto que esté posicionado en un determinado lugar: `child::grupo/attribute::*`

Es posible usar el operador de unir para unir el resultado de dos caminos de localización para que devuelvan conjuntos de nodos: `descendant-or-self::apellidos | descendant-or-self::nota_media`

Dentro de los **pasos de localización**:

- El eje: Especifica relación jerárquica entre los nodos seleccionados por el paso de localización y el nodo contextual
- Selector de nodos o prueba de nodos: Especifica el tipo de nodo de los nodos seleccionados por el paso de localización
- Predicados: Usan expresiones lógicas para refinar el conjunto de nodos seleccionados.

Ejes

Respecto a los ejes, recordar que para seleccionar atributos o espacios de nombres deben usarse de forma explícita los ejes `::attribute` o `::namespace`, ya que no estarán contenidos en los otros ejes.

## Selectores de nodos

Cada eje tiene un tipo principal de nodos. Si un eje puede contener elementos, el nodo principal son los elementos. Si no, son los que el eje contenga (attribute atributos; namespace espacios de nombres; el resto elementos).

Los selectores de nodos son

- Nombre cualificado (QName)
- Todos \*
- `text()`
- `comment()`
- `processing-instruction()`
- `node()`  
También se han añadido en versiones posteriores
- `element()`
- `attribute()`
- `document-node()`

`child::alumno` Selecciona los elementos alumno hijos del nodo contexto. Si no lo tiene, selecciona un conjunto de nodos vacíos.

`attribute::nivel`. Selecciona el atributo nivel del nodo contexto. Si no lo tiene, selecciona un conjunto de nodos vacío.

`text()` Para cualquier nodo de texto. `child::text()`, nodos de texto hijos del nodo contexto

`comment()` Cierta para cualquier nodo comentario

`processing-instruction()`, puede tener un argumento literal; verdadero para cualquier instrucción de procesamiento con nombre igual al literal

`node()` Cierta para cualquier nodo

Ejemplos:

`/descendant-or-self::alumno/child::nombre` (Elementos `<nombre>`)

`/descendant-or-self::grupo/attribute::*` (Atributos de `<grupo>`)

`/descendant-or-self::apellidos/text()` (Texto del elemento de apellidos)

`/descendant-or-self::colegio/child::comment()` (Comentarios del elemento colegio)

`/descendant-or-self::grupo/child::node()` (Todos los elementos contenidos en el `<grupo>`)

## Sintaxis abreviada

`child::` puede ser omitida en los pasos `child::colegio/child::curso` equivale a `/colegio/curso`

`attribute::` puede abreviarse como `@` `[attribute::orden="A"]` equivale a `@attribute=orden"A"`

`//` es abreviatura de `descendant-or-self::node()` `/descendant-or-self::node()/child::alumno` equivale a `//alumno`. Igualmente `child::curso/descendant-or-self::node()/child::alumno` equivale a `curso//alumno`

`descendant-or-self::node()/child::nota_media` se abrevia en `//nota_media`

`child::colegio/child::curso/attribute::etapa` se abrevia en `colegio/curso/@etapa` (camino relativo)

`descendant-or-self::alumno/parent::node()/attribute::*` se abrevia en `//alumno/../*`

`self::node()/descendant::telefono` se abrevia en `./descendant::telefono` (camino relativo)

## 1.4. Predicados

Los predicados filtran un conjunto de nodos con respecto a un eje y a un selector de nodos para producir un nuevo conjunto de nodos.

El paso de la localización puede tener cero, uno o más predicados en cascada.  
Un camino de localización `grupo[2]` es equivalente a `grupo[position()=2]`.

Ejemplos:

a) Primer alumno de 2 ESO A.

```
//curso[@nivel="2"]/grupo[@orden="A"]/alumno[1]
```

b) Apellidos de los alumnos con nota superior a 8.

```
//alumno[nota_media>8]/apellidos
```

c) Nombre de los alumnos que no nacieron en el 2017 y tienen media superior a 7.

```
//alumno[anno_nac!=2017][nota_media>7]/nombre
```

```
//alumno[anno_nac!=2017 and nota_media>7]/nombre
```

d) Apellidos de los alumnos que nacieron en el 2019 y que están suspensos.

```
//alumno[anno_nac=2019 or nota_media<5]/apellidos
```

## 1.3. Funciones

### Funciones de conjuntos de nodos

- `last()`: Último de los nodos del contexto seleccionado.

```
//alumno[position()=1]/nombre
//alumno[1]/nombre
```

- `position()`: Posición del nodo actual dentro de los nodos del contexto seleccionado (inicializa en 1)

```
//curso[@nivel="2"]//alumno[last()]/(nombre|apellidos)
```

- `count(node-set)`: Número de nodos del conjunto de nodos pasado como argumento

```
count(//alumno[anno_nac=2018])
//curso[@nivel="2"]/grupo/count(alumno)
count(//curso[@nivel="1"]/grupo/alumno)
```

- `name(?node-set)`: Nombre cualificado del nodo del conjunto de nodos pasado como argumento. Si no se pasa argumento, toma el nodo contexto como argumento. Si no se declara espacio de nombres da el mismo resultado que `local-name()`
- `local-name(?node-set)`: Nombre local (sin URI) del espacio de nombres del nodo del conjunto de nodos pasado por argumento. Si no se pasa el argumento toma nodo contexto como argumento
- `namespace-uri(?node-set)`: Devuelve la URI del espacio de nombres, si en el nombre local de los nodos pasados como argumento. Si no se pasa, toma el nodo contexto como argumento.
- `id(object)`: Selecciona elementos mediante el identificador único. Los nodos deben estar declarados con ese ID en el DTD o XSD.

### Funciones de cadenas de caracteres

- `string(object?)`: Convierte objeto en cadena de caracteres
- `concat(...)`: concatenación de argumentos
- `starts-with(string, with)`: si comienza por la segunda cadena
- `contains(string, contains)`: si contiene la segunda cadena
- `substring-before(string, string)`: subcadena de la primera cadena que precede a la aparición de la segunda
- `substring-after(string, string)`: subcadena de la primera cadena que sigue a la aparición de la segunda
- `substring(string, number, ?number)`: substring comienza en posición especificada en el segundo y tiene longitud especificada en el tercero
- `string-length(string?)`: Longitud
- `normalize-space(string?)`: normalización de espacios en blanco

- `translate(string, string, string)`: el primer argumento se traducen las apariciones del segundo con lo indicado en el tercero

Ejemplos:

a) Alumnos que su nombre comiencen por 'Al'

```
//alumno[starts-with(nombre,"Al")]
```

b) Nombre y apellidos de los alumnos que se apelliden 'Carmona' tanto de primero como de segundo apellido.

```
//alumno[contains(apellidos,"Carmona")]/(nombre|apellidos)/text()
```

c) Del texto "En un lugar de la Mancha" extrae todo lo que hay delante de 'de'.

```
substring-before("En un lugar de la Mancha","de")
```

d) Del texto "En un lugar de la Mancha" extrae todo lo que hay detrás de 'de'.

```
substring-after("En un lugar de la Mancha","de")
```

e) Del texto "En un lugar de la Mancha" extrae todo lo que hay detrás de 'de'.

```
substring-after("En un lugar de la Mancha","de")
```

f) Del texto "En un lugar de la Mancha" extrae todo lo que hay a partir del carácter número 9.

```
substring("En un lugar de la Mancha",9)
```

g) Del texto "En un lugar de la Mancha" calcula el número de caracteres que tienes.

```
string-length("En un lugar de la Mancha")
```

h) Normaliza un texto, quitando los espacios iniciales y finales y poniendo un único espacio entre palabras.

```
normalize-space(" Esto es una prueba ")
```

g) Sustituir los caracteres en el primer texto que aparecen en el segundo texto por los del tercero.

```
translate("abcdefghi","bdg","BDG")
```

## Funciones lógicas y numéricas

- `boolean(object)`: Argumento convertido a booleano
- `not(boolean)`
- `true()`
- `false()`
- `lang(string)`: Verdadero o falso si el lenguaje del nodo especificado en `xml:lang` es igual que o sublenguaje del especificado como argumento
- `number(object?)`: Pasa a número
- `sum(node-set)`: Suma de los nodos del conjunto de nodos
- `floor()`: Redondeo hacia abajo
- `ceiling()`: Redondeo hacia arriba
- `round()`: Número más próximo al argumento y que sea entero

## 1.3. Estrategias de uso

- Buscar en el XML la posición del atributo y del elemento que se pregunten
- Ver cuál debe ser el resultado que queremos obtener y partir de ahí.
- Posicionar predicado:
  - Ponerlo en el primer elemento padre común
  - Mezclar ambos caminos de localización
- Simplificar

Al trabajar con información en distintos niveles es importante no usar doble barra en mitad de los caminos de localización ya que la búsqueda la reiniciará desde el elemento raíz y no con el elemento ya establecido.

A veces habrá que hacer consultas anidadas.

A veces no queremos repetidos (esto se puede hacer con ayuda del eje `preceding`)

```
//alumno[not(nombre=preceding::nombre)]/nombre
```

---

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

```

<universidad>
 <nombre>Universidad de Victoria</nombre>
 <pais>España</pais>
 <carreras>
 <carrera id="c01">
 <nombre>I.T. Informática</nombre>
 <plan>2003</plan>
 <creditos>250</creditos>
 <centro>Escuela de Informática</centro>
 </carrera>
 <carrera id="c02">
 <nombre>Dipl. Empresariales</nombre>
 <plan>2001</plan>
 <creditos>275</creditos>
 <centro>Facultad de Ciencias Sociales</centro>
 </carrera>
 <carrera id="c03">
 <nombre>Dipl. Relaciones Laborales</nombre>
 <plan>2001</plan>
 <creditos>280</creditos>
 <centro>Facultad de Ciencias Sociales</centro>
 </carrera>
 <carrera id="c05">
 <nombre>Lic. Biología</nombre>
 <plan>2001</plan>
 <creditos>175</creditos>
 <centro>Facultad de Ciencias Experimentales</centro>
 <subdirector>Alonso Pérez</subdirector>
 </carrera>
 <carrera id="c06">
 <nombre>Lic. Humanidades</nombre>
 <plan>1980</plan>
 <creditos>475</creditos>
 <centro>Facultad de Humanidades</centro>
 <subdirector>Jesús Martínez</subdirector>
 </carrera>
 </carreras>
 <asignaturas>
 <asignatura id="a01" titulacion="c01">
 <nombre>Ofimática</nombre>
 <creditos_teoricos>3</creditos_teoricos>
 <creditos_practicos>1.5</creditos_practicos>
 <trimestre>1</trimestre>
 </asignatura>
 <asignatura id="a02" titulacion="c01">
 <nombre>Ingeniería del Software</nombre>
 <creditos_teoricos>6</creditos_teoricos>
 <creditos_practicos>1.5</creditos_practicos>
 <trimestre>2</trimestre>
 </asignatura>
 <asignatura id="a03" titulacion="c02">
 <nombre>Me la invento</nombre>
 <creditos_teoricos>6</creditos_teoricos>
 <creditos_practicos>1.5</creditos_practicos>
 <trimestre>2</trimestre>
 </asignatura>
 </asignaturas>
 <alumnos>
 <alumno id="e01">
 <apellido1>Rivas</apellido1>
 <apellido2>Santos</apellido2>
 <nombre>Víctor Manuel</nombre>
 <sexo>Hombre</sexo>
 <estudios>
 <carrera codigo="c01" />
 <asignaturas>
 <asignatura codigo="a01" />
 <asignatura codigo="a03" />
 <asignatura codigo="a05" />
 </asignaturas>
 </estudios>
 </alumno>
 <alumno id="e02" beca="si">
 <apellido1>Pérez</apellido1>

```



```

 <apellido2>García</apellido2>
 <nombre>Luisa</nombre>
 <sexo>Mujer</sexo>
 <estudios>
 <carrera codigo="c02" />
 <asignaturas>
 <asignatura codigo="a02" />
 <asignatura codigo="a01" />
 </asignaturas>
 <proyecto>Web de IBM.com</proyecto>
 </estudios>
 </alumno>
</alumnos>
</universidad>

```

#### 1 - Nombre de la Universidad:

```
/universidad
```

#### 2 - País de la Universidad:

```
/universidad/país
```

#### 3 - Nombres de las Carreras:

```
/universidad/carreras/carrera/nombre
```

```
universidad/carreras/carrera/nombre
```

```
/universidad/carreras/carrera/nombre/text()
```

#### 4 - Años de plan de estudio de las carreras:

```
/universidad/carreras/carrera/plan
```

`/universidad//carrera/plan` (Dentro de Universidad hay un elemento carrera, puede haber intermedios, no tiene por qué ser hijos directos)

```
/universidad//*/carrera/plan
```

#### 5 - Nombres de todos los alumnos:

`//alumnos/alumno/nombre` (Cualquier estructura pero que después encuentre eso)

#### 6 - Identificadores de todas las carreras:

```
//carreras/carrera/@id
```

#### 7 - Datos de la carrera cuyo id es c01:

Hagamos un predicado (filtro):

```
//carreras/carrera[@id='c01']
```

Nombre de la carrera cuyo id es c01: `//carreras/carrera[@id='c01']/nombre`

Contenido del elemento carrera (llaves): `//carreras/carrera[@id='c01']/*`

#### 8 - Centro en que se estudia de la carrera cuyo id es c02:

```
/universidad/carreras/carrera[@id='c02']/centro
```

```
//carreras/carrera[@id='c02']/centro
```

```
//carreras/carrera/centro[../@id='c02']
```

(Ojo, filtro en carrera y no en carreras. Porque si filtro en carreras, no va a encontrar un hijo id )

`//carreras[carrera/@id='c02']/carrera/centro` Aquí estoy preguntando ¿existe alguna carrera con identificador c02, sí; pues pinto todas.

#### 9 - Nombre de las carreras que tengan subdirector:

```
//carreras/carrera[subdirector]/nombre/text()
```

**10 - Nombre de los alumnos que están haciendo proyecto:**

```
//alumnos/alumno[estudios/proyecto]/nombre/text()
```

```
//alumnos/alumno[./estudios/proyecto]/nombre
```

```
//alumnos/alumno[./proyecto]/nombre
```

 (Partiendo del directorio actual, que busque descendientes)

---

MAL Ruta absoluta: (Ojo, aquí uso ruta absoluta y le estoy preguntando que exista algún proyecto en TODO EL ARBOL. Cuidado con las rutas absolutas)

```
//alumnos/alumno[//proyecto]/nombre
```

**11 - Nombre de las carreras en las que hay algún alumno matriculado:**

```
//carreras/carrera[@id=//alumnos/alumno//carrera/@codigo]/nombre/text()
```

```
//carreras/carrera[@id=//alumnos/alumno/estudios/carrera/@codigo]/nombre/text()
```

**12 - Apellido y nombre de alumnos con beca:**

```
//alumnos/alumno[./@beca="si"]/nombre |
//alumnos/alumno[./@beca="si"]/apellido1 |
//alumnos/alumno[./@beca="si"]/apellido2
```

**13 - Nombre de las asignaturas de la titulación c04:**

```
//asignaturas/asignatura[./@titulacion="c01"]/nombre/text()
```

**14 - Nombre de las asignaturas de segundo trimestre**

```
//asignaturas/asignatura[./trimestre="2"]/nombre/text()
```

```
//asignaturas/asignatura[trimestre="2"]/nombre/text()
```

**15 - Nombre de las asignaturas que no tienen 6 créditos teóricos:**

```
//asignaturas/asignatura[creditos_teoricos!="6"]/nombre/text()
```

```
//asignaturas/asignatura[not(creditos_teoricos="6")]/nombre/text()
```

```
//asignaturas/asignatura[not(creditos_teoricos=6)]/nombre/text()
```

**16 - Código de la carrera que estudia el último alumno:**

(Inicializa en "1")

```
//alumnos/alumno[last()]/carrera/@codigo
```

(El penúltimo) `//alumnos/alumno[last()-1]/carrera/@codigo`

(Las que no son 1) `//alumnos/alumno[position()>1]/carrera/@codigo`

**17 - Código de las asignaturas que estudian mujeres:**

```
//alumnos/alumno[sexo="Mujer"]/estudios/asignaturas/asignatura/@codigo
```

 (Se repetirían)

Mejor, sin repetir códigos

```
//asignaturas/asignaturas[@id=//alumnos/alumno[sexo="Mujer"]//asignatura/@codigo]/@id
```

---

---

---

## 2. XSLT

Gracias a XSL Transformations (XSLT) los procesadores pueden transformar un documento XML en otros documentos XML, HTML o de texto plano con estructuras y contenidos distintos de los originales. (No PDF)

La transformación expresada en XSLT da reglas para transformar un árbol de nodos origen en un árbol de nodos resultado.

Se consigue asociando patrones definidos en la plantilla `xsl:template`. Un patrón, expresado en XPath, se compara con los elementos del árbol de origen.

Si cumplen alguna de las reglas de la plantilla, pasan a formar parte del árbol de nodos resultado. La estructura del árbol de nodos resultado puede ser completamente diferente a la estructura del árbol de origen.

En la hoja pueden usarse:

- **Elementos asociados al URI** <http://www.w3.org/1999/XSL/Transform> usando el prefijo `xsl`
- **Elementos de extensión.** Usados por los desarrolladores para funcionalidades extras
- **Elementos de resultado literal:** Se añaden al árbol de nodos resultado y no pertenecen al espacio de nombres (Elementos HTML, texto...)

El XSLT se asocia al XML con la instrucción de procesamiento:

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

En cualquier caso al ejecutar el procesador XSLT se le puede indicar los documentos XML y XSL de la transformación.

### 2.1. Estructura de una hoja XSLT

Debe incluirse el prólogo XML.

La hoja de estilos se declara así, con el siguiente espacio de nombres:

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

Sinónimo de `<xsl:stylesheet>` es `<xsl:transform>`.

Como atributos se le puede indicar `version` que es "1.0" o "2.0", generalmente "1.0"

#### Simplificación de hojas de estilo con elementos de contenido literal

Si son elementos de contenido literal puede simplificarse:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:output method="html"/>
 <xsl:template match="/">
 <html>
 <head>
 <title>Colegio Cervantes</title>
 </head>
 <body>
 <h1>Nombre del colegio: <xsl:value-of select="colegio/nombre"/></h1>
 <p>Teléfono: <xsl:value-of select="colegio/telefono"/></p>
 <p>Suma de notas: <xsl:value-of select="sum(//nota_media)"/></p>
 </body>
 </html>
 </xsl:template>
</xsl:stylesheet>
```

Porque puede omitirse `<xsl:stylesheet>` y `<xsl:output>` como elemento raíz y poner en su lugar `<html>` agregándole el espacio de nombres de XSLT y su versión (atributo obligatorio).

Desde el contenido literal se puede acceder a las instrucciones del espacio de nombres xsl como si se hubiese seleccionado como patrón (`match="/"`) el nodo raíz del árbol de nodos origen.

Después pueden usarse instrucciones que no sean de nivel superior sin problema.

```
<html xsl:version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <head>
 <title>Colegio Cervantes</title>
 </head>
 <body>
 <h1>Nombre del colegio: <xsl:value-of select="colegio/nombre"/></h1>
 <p>Teléfono: <xsl:value-of select="colegio/telefono"/></p>
 <p>Suma de notas: <xsl:value-of select="sum(//nota_media)"/></p>
 </body>
</html>
```

## 2.2. Elementos XSLT de nivel superior

Son elementos del nivel superior aquellos que son hijos directos de `<xsl:transform>` o `<xsl:stylesheet>`

Su ámbito de aplicación es toda la hoja de estilos que se declara. Son estructuras contenedoras de instrucciones. Pueden aparecer varias veces o no aparecer. Salvo `xsl:import` que debe aparecer en primer lugar, el orden de aparición no es relevante.

Estos no pueden usarse dentro de otros elementos salvo `xsl:variable` y `xsl:param`

`xsl:template`:

Tienen el atributo `match` donde se indican los nodos seleccionados (o el atributo `name` que sirve para hacer referencia a una plantilla mediante `xsl:call-template`). O uno u otro serán obligatorios). El atributo `match` no puede incluir uso de variables.

El contenido que tiene las reglas que se ejecutarán cuando la plantilla sea usada.

En la hoja de estilos puede haber más de una plantilla. Cada plantilla puede hacerse coincidir con un nodo determinado.

- Plantilla vacía: Si hay una plantilla que se corresponde con un nodo y no tiene nada, no se muestra nada.
- Nodo sin plantilla asociada: Se muestra el contenido textual del nodo (sin atributos)

A veces puede haber varias plantillas y que no se comporten como esperamos.

Por ejemplo quizás tenemos dos plantillas y vemos que a la segunda "no le hace caso" (porque ya recorrió todo el árbol de origen la primera). (No le pusimos `<xsl:apply-templates>`)

De forma habitual se puede usar una única plantilla que acceda al elemento raíz. Dentro de esta, usar expresiones XPath para seleccionar las partes que se quieran e instrucciones de recorrido para construir la estructura que se desee.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:output method="text" />
 <xsl:strip-space elements="*" />
 <xsl:template match="/">
 <xsl:apply-templates select="//alumno"/>
 </xsl:template>
 <xsl:template match="//alumno">
 <xsl:apply-templates/>
 </xsl:template>
 <xsl:template match="//anno_nac">
 <xsl:text>privado</xsl:text>
 </xsl:template>
</xsl:stylesheet>
```

Simplificado a:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:output method="text" />
 <xsl:template match="/">
 <xsl:for-each select="//alumno">
 <xsl:value-of select="nombre"/>
 <xsl:value-of select="apellidos"/>
 <xsl:text>privado</xsl:text>
 <xsl:value-of select="nota_media"/>
 </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

**xsl:variable** y **xsl:param**:

Nombre que se vincula a un valor. Tiene atributo **name** obligatorio.

Cuando se use, debe ponerse el atributo name con el símbolo \$ delante.

Otro atributo es **select** que si se le especifica una expresión xpath almacena el resultado en la variable.

**xsl:param** es predeterminado que se usa pero que puede ser sobrescrito cuando se invoca a una plantilla con **xsl:with-param**. **xsl:variable**, no.

**xsl:output**:

Especifica el formato de salida deseado. Puede omitirse ya que el procesador XSLT devolverá el resultado como secuencias de caracteres.

Puede especificarse características como:

- method: Formato de salida (xml, html, text)
- version: Versión del método de salida
- indent: Espacios en blanco adicionales (yes/no)
- encoding: Codificación
- standalone: Relación con otros documentos (yes/no)
- media-type, doctype-system, doctype-public, omit-xml-declaration, cdata-section-elements

**xsl:import**: y **xsl:include**: Añadir contenido a la hoja de estilo (Import: con menor preferencia que los de la hoja. Include: con la misma). Ambos tienen un atributo **href** para indicar dónde se encuentran.

**xsl:strip-space** y **xsl:preserve-space**: Para indicar comportamiento respecto a los espacios.

**xsl:key**: Declarar elemento como clave para que pueda usarse como tipos de datos ID, IDREF, IDREFS. Declara una clave que puede usarse en otro lugar de la hoja de estilos con función **key()**

**xsl:decimal-format**: Define formato para convertir números en cadenas de caracteres.

**xsl:namespace-alias**: Prefijo alternativo para el espacio de nombres.

**xsl:attribute-set**: Grupos de atributos reutilizables en los elementos **xsl:element** y **xsl:copy** mediante el atributo use-attribute-sets.

## 2.3. Otros elementos XSLT

### Instrucciones de manipulación de plantillas

**xsl:apply-templates**: Indicar que queremos que se continúe la búsqueda de plantillas en los elementos hijos (sin pasar al siguiente elemento hermano). Se indica mediante el atributo **select**, indicando el conjunto de nodos hijos que queremos que se recorran y buscando plantillas que se correspondan con ellos. Puede aparecer como contenido **xsl:sort** para ordenar los resultados antes de aplicar la plantilla. También puede estar **xsl:with-param** para pasar valores a la plantilla que sobrescriben los por defecto indicados en **xsl:param**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:output method="text" />
 <xsl:strip-space elements="*" />

 <xsl:template match="/">
 <xsl:apply-templates select="//alumno" />
 </xsl:template>

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

 <xsl:template match="//anno_nac">
 <xsl:text>privado</xsl:text>
 </xsl:template>

</xsl:stylesheet>
```

Y en una sola plantilla:

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

 <xsl:template match="/">
 <xsl:for-each select="//alumno">
 <xsl:value-of select="nombre" />
 <xsl:value-of select="apellidos" />
 <xsl:text>privado</xsl:text>
 <xsl:value-of select="nota_media" />
 </xsl:for-each>
 </xsl:template>

</xsl:stylesheet>
```

**xsl:call-template**: Invocar a una plantilla declarada con atributo **name**

**xsl:with-param**: Pasar valores a una plantilla que sobrescribirán los indicados por el elemento **xsl:param** que tengan el mismo atributo **name**.

## Instrucciones de control

**xsl:for-each**: Bucle que recorre de uno en uno todos los nodos indicados. Se hace en el orden en el que son procesados salvo usar **xsl:sort**

**xsl:if**: Bloque condicional. Solo se ejecuta parte de la plantilla si se cumple la condición en el atributo **test**

Vamos a realizar una transformación en la que vamos a ordenar los alumnos por orden de apellido descendente y de cada uno de ellos mostraremos su nombre y su fecha de nacimiento o su nota media dependiendo de si su posición es par o impar.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:output method="html" version="5.0" indent="yes" />

 <xsl:template match="/">
 <html>
 <body>

 <xsl:for-each select="//alumno">
 <xsl:sort select="apellidos" order="descending" />

 <xsl:if test="((position() mod 2)=1)">
 <xsl:value-of select="nombre" />
 <xsl:text>. Fecha de nacimiento: </xsl:text>
 <xsl:value-of select="anno_nac" />
 </xsl:if>

 </xsl:for-each>

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

</xsl:stylesheet>
```

```

 </xsl:if>
 <xsl:if test="((position() mod 2)=0)">
 <xsl:value-of select="nombre"/>
 <xsl:text>. Nota media: </xsl:text>
 <xsl:value-of select="nota_media"/>
 </xsl:if>

</xsl:for-each>

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

</xsl:stylesheet>

```

Esta misma transformación también se podría hacer sustituyendo el `xsl:for-each` por una llamada recursiva a otra plantilla mediante `xsl:apply-templates`.

`xsl:choose`, `xsl:when`, `xsl:otherwise`:

Permite evaluar condicionales con más de un caso posible.

Secuencia de elementos `xsl:when` seguidos de elemento `xsl:otherwise`

Los `when` tienen atributos `test` donde se especifica la expresión.

`xsl:sort`: Ordenar elementos. Debe agregarse como elemento hijo de `xsl:apply-templates` o `xsl:for-each` (En este caso debe ser el primer hijo). En el atributo `select` se indica una expresión que devuelva una cadena de caracteres que servirá como clave de ordenación. El valor por defecto del atributo es el nodo actual (.)

Otros atributos son `order` (ascending, descending); `lang` para el lenguaje de la clave (alfabeto);

`data-type` para el tipo de dato (text, number, nombre cualificado); `case-order` para casos especiales (upper-first, lower-first, para text por ejemplo)

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:output method="html" version="5.0" indent="yes" />

 <xsl:template match="/">
 <html>
 <head>
 <style>
 .aprobado {color:green;}
 .suspenseo {color:red;}
 .sobresaliente {color:blue}
 </style>
 </head>
 <body>
 <table border="0">
 <thead style="background-color:gray; color:white;">
 <th>NOMBRE</th>
 <th>APELLIDOS</th>
 <th>NOTA MEDIA</th>
 </thead>
 <tbody style="background-color:white;">
 <xsl:for-each select="//alumno">
 <xsl:sort select="apellidos"/>
 <xsl:element name="tr">
 <xsl:attribute name="class">
 <xsl:choose>
 <xsl:when test="nota_media < 5">suspenseo</xsl:when>
 <xsl:when test="nota_media < 9">aprobado</xsl:when>
 <xsl:otherwise>sobresaliente</xsl:otherwise>
 </xsl:choose>
 </xsl:attribute>
 <td><xsl:value-of select="nombre"/></td>
 <td><xsl:value-of select="apellidos"/></td>
 <td style="text-align: center;"><xsl:value-of select="nota_media"/></td>
 </xsl:element>
 </xsl:for-each>
 </tbody>
 </table>
 </body>
 </html>
 </xsl:template>
</xsl:stylesheet>

```

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

## Instrucciones de salida

**xsl:value-of**: Generar texto extrayéndolo del árbol de origen ,usando expresión XPath o insertando el valor de una variable. Crea un nodo text en el árbol de nodos destinos. Tiene un atributo obligatorio **select** para la expresión que será evaluada.

Para añadir un conjunto de nodos usar en lugar de este, **xsl:copy-of**

**xsl:number**: Insertar número formateado. Se especifica en el atributo **value**. La expresión se evalúa y el objeto se convierte a número que se redondea a entero y se convierte a cadena de caracteres. Hay atributos que controlan la numeración **level** (niveles del árbol de origen que se usan: single, multiple, any; por defecto single). **count** qué nodos serán contados (semejantes al nodo contexto por defecto); **from** desde donde empezar a contar.

**xsl:element**: Crear elemento con el nombre indicado por **name**. En **namespace** puede indicársele atributo opcional.

**xsl:attribute**: Añadir atributos a los elementos creados en una plantilla mediante **xsl:element** o directamente con etiquetas de apertura y cierre. Se indicará mediante **name**, puede indicársele **namespace**. Puede construirse con uso de llaves (attribute value template): Construir con etiqueta y el valor ponerlo entre llaves. Si tiene varios atributos o atributo que se quiere reutilizar usar **attribute-set**

**xsl:text**: Crear nodos textos. Aplica lo que se haya especificado en **xsl:strip-space** y **xsl:preserve-space**

**xsl:comment**: Crear nodo comentarios en árbol de nodos destino.

**xsl:processing-instruction**: Crear nodo de instrucciones de procesamiento. Atributo **name** debe especificar nombre o clase de la instrucción de procesamiento.

## Ejemplito:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:output method="xml" version="1.0" indent="yes" />
 <xsl:template match="/">
 <xsl:text>
 </xsl:text>
 <xsl:comment>Documento de estilos CSS vinculado al archivo XML resultante</xsl:comment>
 <xsl:text>
 </xsl:text>
 <xsl:processing-instruction name="xml-stylesheet">href="style.css" type="text/css"</xsl:processing-instruction>
 <xsl:text>
 </xsl:text>
 <xsl:element name="listado">
 <xsl:comment>Listado de alumnos ordenados por notas cuyo nombre empieza por
A</xsl:comment>
 <xsl:for-each select="//alumno[starts-with(nombre,'A')]">
 <xsl:sort select="nota_media" order="descending" data-type="number"/>
 <xsl:element name="persona">
 <xsl:attribute name="calificacion">
 <xsl:value-of select="nota_media"/>
 </xsl:attribute>
 <xsl:attribute name="num_orden">
 <xsl:number value="position()" format="#x30A2;"/>
 </xsl:attribute>
 <xsl:value-of select="nombre"/>
 <xsl:text> </xsl:text>
 <xsl:value-of select="apellidos"/>
 </xsl:element>
 </xsl:for-each>
 </xsl:element>
 </xsl:template>
</xsl:stylesheet>
```



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

## Otras instrucciones XSLT

**xsl:copy**: Añade el nodo actual al árbol de destinos. **Se copian también sus nodos de espacios de nombres pero no los nodos atributos ni los hijos**. Puede usar **use-attribute-sets** para añadir conjuntos de atributos declarados anteriormente. Solo puede usarse con nodos que puedan tener atributos o elementos hijos, (nodo raíz y nodos elemento)

**xsl:copy-of**: Añadir fragmento completo del árbol de nodos origen al árbol de destino. No es necesario convertirlo a cadena de caracteres. Tiene atributo **select** donde debe indicarse expresión XPath del árbol a copiar. **Copia propio elemento y nodos atributos, espacios de nombres e hijos**

**xsl:apply-imports**: Indicar cuándo realizar importación para sobrescribir regla de plantillas

**xsl:fallback**: Secuencia ordenada de ejecución

## 2.4. Funciones propias de XSLT

**document()** Puede accederse a los datos de otros ficheros (Se usa en XPath pero es propia de XSLT).

- **document(URI)** (Elemento raíz del documento XML que se localiza en esa URI)
- **document(nodo)** Conjunto de nodos cuya raíz es el nodo dado.
- si no se indica parámetro, se toma el documento actual.

**key(string, object)**: Similar a **id()** de XPath. Primer parámetro nombre de la clave que debe ser nombre cualificado indicado en elemento **xsl:key**. Segundo argumento debe ser el valor de la clave. Devuelve el conjunto de nodos que cumplen esas condiciones.

**format-number(number, number, string?)**: Convierte el primer parámetro a una cadena de caracteres siguiendo el patrón indicado en el segundo parámetro. El tercer argumento es el nombre del **xsl:decimal-format** que se usará.,

**current()**: Devuelve el valor del nodo contexto. Normalmente se sustituye por su versión abreviada. **<xsl:value-of select="current()"/>** equivale a **<xsl:value-of select="."/>**.

**generate-id(node-set?)**: Genera identificador único para el conjunto de nodos pasados por parámetro o del contexto actual.

Otras funciones: **unparsed-entity-uri()**, **system-property()**, **element-available()** y **function-available()**.

## 3. Procesadores XSLT

Un **procesador XSLT** es una aplicación que realiza transformación partiendo de documento XML y siguiendo las reglas indicadas en un documento XSLT para dar lugar a uno de los formatos permitidos: XML, HTML, texto plano.

El procesador XSLT cuando lee el archivo XML inicial y crea un árbol de nodos origen. A continuación va procesando las instrucciones de la hoja de estilos recogida en el documento XSLT. Esta hoja de estilos contendrá una o varias plantillas. Cada plantilla tendrá una expresión de emparejamiento. Estas expresiones serán buscadas en el archivo XML original y si encuentra elementos compatibles se ejecutan las instrucciones oportunas. Estas instrucciones irán generando un árbol de nodos destino. Una vez recorridas todas las plantillas y generado todo el árbol de nodos destino este será escrito en un archivo de salida.

Empresas de procesadores XSLT:

- Saxon
- Gnome
- Apache

- Microsoft  
Procesadores XSLT de línea de comandos (xsltproc, saxonb-xslt, xalan), como librerías de distintos lenguajes de programación, en páginas web accesibles con nuestro navegador (<http://xslttransform.net/>, <https://xslttest.appspot.com/>) o integrados en editores específicos de XML (XML Copy Editor, Oxygen XML Editor, Altova XMLSpy, Liquid XML Studio, Stylus Studio) o generales (Netbeans, Visual Studio, IntelliJ IDEA, Eclipse).

## Depuración

Seguir la generación del documento a partir de XML aplicándole XSLT.

Facilitan la localización y corrección de errores: código paso a paso, ejecutar hasta el cursor, hasta el final, salir, pausar, detener.

Editores XML como Oxygen XML, Editix, Liquid XML Studio, Stylus Studio, Altova XML Spy (de pago todos) incluyen depurador. También editores algo más "open source" como IntelliJ Idea + Plugin XSLT Debugger | Eclipse + Plugin EclipseXML Editors and Tools | NetBeans + plugin netbeans-xslt-debugger (obsoleto)

# UD 06 - ALMACENAMIENTO DE INFORMACIÓN

## 1. Utilización de XML para almacenamiento de información

- XML permite **guardar y comunicar** información acerca de objetos. Codifica la información de una forma separada a la que se le presenta al usuario.
- Para encontrar un fragmento específico de información, debe procesarse todo el XML
- **Base de datos XML**: Colección de documentos XML (archivo en el sistema de archivos con cadena válida XML) en la que cada uno representa un registro.
- La estructura del documento XML puede seguir el mismo esquema pero **no es obligatorio que sea así**. Cada archivo **puede ser configurado de forma independiente**, lo que da **flexibilidad y facilita el desarrollo**.

El almacenamiento de datos se podría dividir en dos categorías:

- **Centrados en los datos**: Documento con estructura bien definida con esquemas altamente estructurados (factura, albarán, ticket, matrícula, acta de notas, órdenes de compra). Apropriados para publicidad. Datos pueden actualizarse. **Muchos elementos con poco contenido, muy estructurados**.  
Relativamente planos. Tipos de datos relativamente complejos. Poca importancia al orden.
- **Centrados en los documentos**: Impredecibles en tamaño. Contenidos con tipos de tamaño limitado y reglas menos flexibles para campos opcionales. Archivos variables dependiendo de quién los realice o escriba (libro de texto, manual, informe). **Pocos elementos con gran cantidad de contenido y desestructurados**.  
Estructura irregular. Tipos de datos simples. Importancia al orden.

Los sistemas de almacenamiento XML deben acomodarse a ambos tipos de requerimientos de datos. La mayoría de productos se enfocan en servir uno de esos formatos mejor que otro: Las **bases de datos tradicionales** son mejores para tratar requerimientos centrados en los datos. Los **sistemas de administración de contenidos y documentos** son mejor para datos centrados en el documento.

Sistemas de bases de datos deben exponer datos relacionales como XML y almacenar el XML recibido como datos relacionales para transferir, obtener y almacenar datos requeridos por la aplicación.

La tecnología XML:

- usa uno o más documentos para almacenar información
- define esquemas sobre información
- tiene lenguajes de consulta específicos para recuperar la información requerida
- Dispone de APIs (SAX, DOM)
- No tiene almacenamiento y seguridad eficientes (índices, seguridad, transacciones, integridad de datos, acceso concurrente, disparadores...). es imposible pensar que XML se use para tareas transaccionales de una organización.

## 2. Sistemas de almacenamiento de información

Las bases de datos relacionales:

Usan relaciones (tablas bidimensionales) para representar los datos del mundo real y están asociadas al lenguaje SQL

No están bien preparadas para almacenar estructuras de tipo jerárquico como documentos XML porque los XML:

- Tienen **carácter heterogéneo** frente a la estructura regular de la base de datos relacional
- Contiene **muchos niveles de anidamiento** frente a los datos planos relacionales
- Contiene un **orden intrínseco** mientras que los datos relacionales no son ordenados

- Son **datos dispersos** que pueden representar la carencia de información mediante la ausencia del elemento, mientras que en los datos relacionales cada columna tiene un valor.

Sin embargo se prefieren bases de datos relacionales y orientadas a objetos para almacenar estructuras XML aunque no sean de forma nativa porque son **bien conocidas**, especialmente en cuanto a tu rendimiento.

## Reglas de transformación de bases de datos relacional a XML. Creación del DTD.

La estructura XML debe acomodar clases primarias, secundarias, tablas y columnas. Pueden existir muchas variaciones de esquemas XML para representar la misma base de datos relacional.

El proceso de traducción de BBDD relacional a XML podría seguir los siguientes pasos:

- **Crear el esquema XML:** con un elemento para cada tabla y los atributos para cada columna no clave. Columnas que no permiten valores nulos deben ser marcadas como requeridas; Columnas que permiten valores nulos deben ser marcadas como opcionales. En lugar de atributos podrían anidarse las columnas como elementos pero eso puede dar problemas si existe el mismo nombre de columna en más de una tabla.
- **Crear claves primarias en el esquema XML:** Para las columnas que son claves primarias puede agregarse un atributo con un ID. Y en el esquema XML definirlo como de tipo ID. Pueden surgir problemas de colisión al crear las claves en el XML porque el ID necesita ser único en todo el documento. Podría añadirse el nombre del elemento (nombre de la tabla), al valor la clave primaria (valor del atributo) para que sea único a través del documento XML.
- **Establecer relaciones de clave migrada o foránea:** Se puede anidar bajo el elemento padre, un ID de esquema XML puede ser usado para apuntar a la estructura XML correspondiente conteniendo un IDREF.

## XML y Bases de datos orientadas a objetos

Las BBDD orientadas a objetos soportan modelo de objetos puros (no se basan en otros modelos clásicos como el relacional; están influidos por los lenguajes de POO; es intento de añadir la funcionalidad de un SGBD a un lenguaje de programación; son alternativa al almacenamiento y gestión de documentos XML)

El estándar lo componen:

- **Modelo de objetos.** Concebido para dar modelo de objetos estándar para las bases de datos orientadas a objetos. En él se basan el lenguaje de definición de objetos y el lenguaje de consultas.
- **Lenguajes de especificación de objetos ODL (Object Description Language)**
- **Lenguaje de consulta OQL (Object Query Language)**
- **Bindings** para C++, Java y Smalltalk. Definen un OML (Object Manipulation Language) para soportar objetos persistentes. Incluye soporte para OQL, navegación y transacciones. Una vez transformado el documento XML en objetos estos los gestiona el SGBDOO. Se consulta con el lenguaje OQL. La indexación, optimización, consultas... son los del SGBDOO y no son específicos para XML.

## XML y Bases de datos nativas

Son bases de datos que soportan transacciones, acceso multiusuario, lenguajes de consulta y que están específicamente diseñadas para almacenar XML.

Se caracterizan por:

- **Almacenamiento de documentos en colecciones.** Base de datos estructurada en colecciones (conjunto de documentos), de modo que es estructura de árbol donde cada documento pertenece a una única colección.
- **Validación de documentos**
- **Consultas:** Utilizando el lenguaje **XQuery**
- **Indexación XML:** Índices que aceleran las consultas
- **Identificadores únicos:** Asociados a cada documento XML

- Actualizaciones y borrados:
  - Almacenamiento basado en texto: Almacena el documento XML entero en forma de texto y proporciona forma de acceder a él
    - Posibilidad 1: Lo almacena como BLOB en base de datos relacional por fichero y da índices que aceleren el acceso a la información
    - Posibilidad 2: Almacena en un almacén adecuado con índices, soporte para transacciones, etc
  - Almacenamiento basado en el modelo: Almacena modelo binario del documento (DOM) en almacén existente o específico
    - Posibilidad 1: Traduce el DOM a tablas relacionales (elementos, atributos, entidades)
    - Posibilidad 2: Traduce el DOM a objetos en BBDDOO
    - Posibilidad 3: Usa almacén para esta finalidad

Características bases de datos nativas: Consultas, Creación de índices y Creación de identificadores únicos.

Los triggers son de bases de datos relacionales.

### 3. XQuery

**XQuery:** Lenguaje funcional diseñado para escribir consultas sobre colecciones de datos en XML. Puede aplicarse a archivos XML y a bases de datos relaciones con funciones de conversión de registros a XML. Extrae información de un conjunto de datos organizados con árbol de etiquetas XML pero no le importa el origen de los datos.

#### Requerimientos técnicos que cumple XQuery:

- Lenguaje declarativo
- Independiente del protocolo de acceso (archivo local, servidor BBDD, archivo XML)
- Consulta y resultados: Respetar modelo XML
- Consulta y resultados: Ofrecer soporte para namespaces
- Soportar XML-Schema y DTO
- Independiente de la estructura del documento
- Soportar tipos simples (enteros y cadenas) y complejos (nodo compuesto)
- Soportar cuantificadores universales (para todo) y existenciales (existe)
- Soportar operaciones sobre jerarquías de nodos y secuencias
- Combinar información de múltiples fuentes en una consulta
- Manipular datos independientemente del origen de estos
- Lenguaje independiente de la sintaxis (varias sintaxis distintas para expresar la misma consulta)

#### Aplicaciones:

- Recuperar información a partir de datos XML
- Transformar estructuras de datos XML en otras estructuras
- Ofrecer alternativa a XSLT para hacer transformaciones de datos en XML, HTML, PDF

#### Motores de XQuery:

De código abierto:

- Qexo
  - Saxon (Saxon-B open source Saxon-SA propietario)
  - Qizx/open: Todas las características salvo importación y validación de XML-schema)
  - Xquark Bridge: Permite importar y exportar a bases de datos relacionales, respetando restricciones de integridad y relaciones implícitas del XML en relaciones explícitas. XQuery, MySQL, Oracle, SQLServer...
- Otras:
- Xquark Bridge: Importar y exportar datos a bases de datos relacionales usando XML.
  - BumbleBee: Entorno de prueba automático para validar motores XQuery y consultas XQuery. Se distribuye con pruebas ya preparadas y permite redactar y ejecutar pruebas propias. Es propietario. Soporta Qexo, Qizx/open y Saxon. Cerisent, Ipedo, IPSI-XQ, X-Hive.

## Modelo de datos

En Xquery el orden en el que se encuentren los datos es importante. No es igual buscar etiqueta `<B>` dentro de `<A>` o todas las etiquetas `<B>` del documento.

La entrada y salida de consulta XQuery se definen según un modelo de datos que proporciona representación abstracta de uno o más documentos XML y que se basa en:

- Definición de secuencia como **colección ordenada de 0 o más items**. Además es **heterogénea**. Pero una **secuencia nunca puede ser el item de otra secuencia**.
- Orden del documento**. Orden en el que los nodos aparecerían si se representase la jerarquía en formato XML. (Si el primer carácter de un nodo precede al primer carácter de otro nodo, también lo precederá en el orden)
- Contempla **valor especial llamado "error value"** al evaluar expresión que contiene un error.

## Expresiones

Una consulta en XQuery es una expresión que lee una secuencia de datos en XML y devuelve otra secuencia de datos XML.

- El valor de la expresión es secuencia heterogénea de nodos y valores atómicos.
- Formada por expresiones y palabras simples unidas mediante palabras reservadas.
- Xpath es lenguaje declarativo** para localizar nodos y fragmentos. XQuery está sobre la base de Xpath. **Toda expresión XPath también es consulta Xquery válida.**
- Comentarios entre** `(: :)`
- \*Caracteres** `{ }` delimitan expresiones evaluadas para crear documento nuevo
- Admite expresiones condicionales del tipo **if-then-else**

En cualquier lugar del documento:

```
&b in doc("libro.xml")//libro
```

- Las consultas XQuery pueden estar formadas por hasta cinco tipos de cláusulas diferentes porque sigue **norma FLWOR**, que equivalen a For, Where, Group by, Having, Order By y Limit.
- Si se incluyen varias consultas en el archivo `.xquery` para ejecución conjunta irán separadas por coma.
- En la sentencia FLWOR debe haber un FOR o un LET (el resto, si existen, deben seguir ese orden de palabra FLWOR)
  - For**: Vincula a expresiones escritas en XPath creando flujo de tuplas vinculando cada tupla a una variable. Si en la consulta aparece más de una cláusula FOR (o más de una variable en cláusula FOR), el resultado es el producto cartesiano de esas variables.
  - Let**: Vincula variable al resultado completo de la expresión añadiendo esos vínculos a tuplas generadas por cláusula for o, si no existe, creando única tupla con esos vínculos.
  - Where**: Filtra tuplas producidas por FOR y LET eliminando las que no cumplen condiciones dadas
  - Order by**: Ordena las tuplas generadas por FOR y LET según criterio dado después de haber sido filtradas por WHERE. Por defecto, orden ascendente (puede ponerse descending)
  - Return**: Resultado de la consulta por la tupla dada tras ser filtrada por Where y ordenada por Order by.

For y let sirven para crear las tuplas con las que trabajará el resto de las cláusulas y pueden usarse tantas veces como se quiere

Order by, where y return solo una vez.

Ninguna cláusula FLWOR es obligatoria en la consulta XQuery. Por eso cualquier expresión XPath es XQuery válida.

La consulta XQuery tiene:

- Prólogo**: Lugar donde se declaran namespaces, funciones, variables...
- Expresión**: Consulta propiamente dicha

## Diferencias entre for y let

- La cláusula for vincula una variable con cada nodo que encuentre en la colección de datos.
- La cláusula let vincula una variable con todo el resultado de una expresión. Por eso aparecen... una única vez.

## Funciones

### Numéricas

- `floor()`: inferior más próximo
- `ceiling()`: superior más próximo
- `round()`: valor dado al más próximo
- `min()` `max()`: mínimo y máximo de los nodos dados
- `avg()`: valor medio
- `sum()`: suma

### Cadenas de texto

- `concat()`: unión de dos cadenas
- `string-length()`: longitud
- `startswith()`, `ends-with()`: si comienza o termina
- `upper-case()`, `lower-case()`: transforma en mayúsculas o minúsculas

### Uso general

- `empty()`: true si no contiene elemento
- `exists()`: true si contiene elemento
- `distinct-values()`: extrae valores de secuencia de nodos y crea secuencia con valores únicos

### Existenciales

- `some`, `every`: devuelven algún o todos los elementos...
- Puede construir sus propias funciones también

```
declare nombre_funcion($param1 as tipo_dato1, $param2 as tipo_dato2,... $paramN as tipo_datoN)
as tipo_dato_devuelto
{
 ...CÓDIGO DE LA FUNCIÓN...
}
```

## Operadores

**De valores:** eq (igual), ne (no igual), lt (menor que), le (menor o igual que), gt (mayor que), ge (mayor o igual que)

**Comparación generales:** = != > >= < <=

**Comparación de nodos:** `is` si están ligadas al mismo nodo; `is not` si no están ligadas al mismo nodo

**De órdenes de los nodos:** << (true si el nodo ligado al primer operando ocurre primero que el nodo ligado al segundo)

**Secuencias de nodos:** Union (suma de los dos), Intersect (en los dos), Except (primer operando y no en el segundo)

**Aritméticos:** + - \* div y mod

## XQuery Update Facility

Pequeña extensión que provee XQuery para modificar documentos XML.

insert

Several modifiers are available to specify the exact insert location: insert into **as first/as last**, insert **before/after** and insert **into**.

```
insert node (attribute { 'a' } { 5 }, 'text', <e/>) into /n
```

delete

```
delete node //n
```

replace

```
replace node /n with <a/>
```

```
replace value of node /n with 'newValue'
```

rename

```
for $n in //originalNode
return rename node $n as 'renamedNode'
```

- ☐ Insert node <autor><nombre>Barbic</autor>  
before  
doc("BD\_Autores.xml")//Autores/Autor[1]
- ☐ Insert node <autor><nombre>Barbic</autor>  
as first into doc("BD\_Autores.xml")//Autores

### Ejemplo 1

Para el XML alumnos.xml:

```
<alumnos>
 <alumno dni="93940">
 <nombre>Jose</nombre>
 <apells>Bernardo</apells>
 <nota>7</nota>
 </alumno>
 <alumno dni="93940">
 <nombre>Juan</nombre>
 <apells>López</apells>
 <nota>4</nota>
 </alumno>
</alumnos>
```

Veamos un ejemplito:

```
for
 $a in doc("alumnos.xml")//alumnos
where
 $a/nota > 5
return
 <aprobado>{$a/@dni,$a/nota}</aprobado>
```

Que nos devuelve:



```
<aprobado dni="93940">
 <nota>7</nota>
</aprobado>
```

## Ejemplo 2

Para XML libro.xml:

```
<bib>
 <libro año="1994">
 <titulo>TCP/IP Illustrated</titulo>
 <autor>
 <apellido>Stevens</apellido>
 <nombre>W.</nombre>
 </autor>
 <editorial>Addison-Wesley</editorial>
 <precio> 65.95</precio>
 </libro>
 <libro año="1992">
 <titulo>Advan Programming for Unix environment</titulo>
 <autor>
 <apellido>Stevens</apellido>
 <nombre>W.</nombre>
 </autor>
 <editorial>Addison-Wesley</editorial>
 <precio>65.95</precio>
 </libro>
</bib>
```

Veamos un ejemplito:

```
for
 &b in doc("libro.xml")//bib/libro
let
 &c := &b//autor
where
 count(&c) > 2
order by
 &b/titulo
return
 &b/titulo
```

Busca nodos `<libro>` hijos de `<lib>` con más de dos autores, los ordena por el título y devuelve su título.

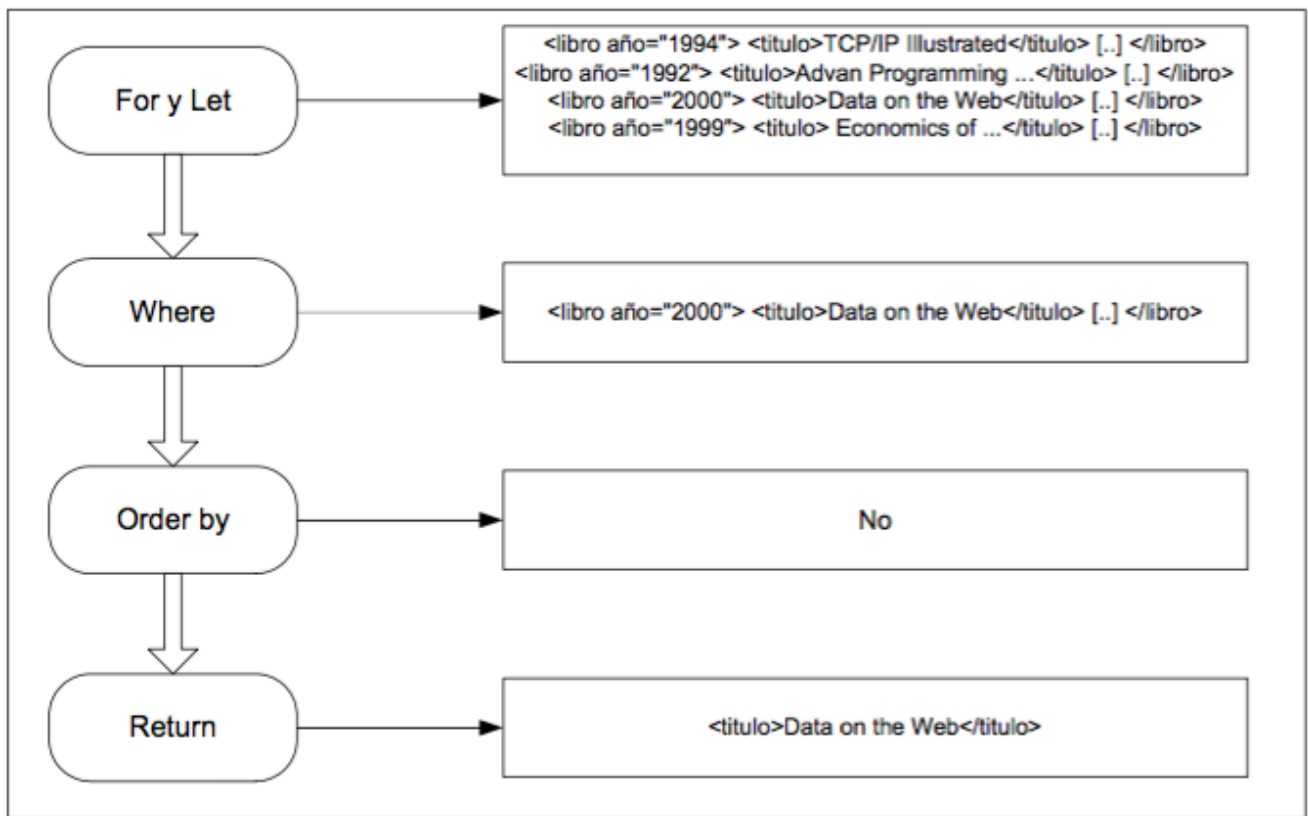
Retorna:

```
<title>Data on the Web</title>
```

## Ejemplos varios

```
for
 $b in doc("libros.xml")//libro
where
 $b/@año = "2000"
return
 $b/titulo
```

Gráficamente:



(:1. Ejemplo de uso de la cláusula FOR. Obtener todos los títulos de los libros del fichero libros.xml.:)

```
for $d in doc("libros.xml")/biblioteca/libros/libro/titulo
return <titulos>{ $d }</titulos>
```

(:2. Ejemplo de uso de la cláusula LET. Obtener todos los títulos de los libros del fichero libros.xml.:)

```
let $d := doc("libros.xml")/biblioteca/libros/libro/titulo
return <titulos>{ $d }</titulos>
```

(:3. Ejemplo de uso de la cláusula FOR y LET juntas. Obtener todos los títulos de los libros del fichero libros.xml junto con los autores de cada libro.:)

```
for $b in doc("libros.xml")//libro
let $c := $b/autor
return <libro>{ $b/titulo, <autores>{ $c }</autores>}</libro>
```

(:Otra solución posible a este ejercicio, en este caso utilizando únicamente la cláusula FOR:)

```
for $b in doc("libros.xml")//libro
return <libro>{ $b/titulo, <autores>{$b/autor}</autores>}</libro>
```

(:4. Ejemplo de uso de las cláusulas WHERE y ORDER BY en una consulta con dos ficheros. Obtiene los títulos de los libros prestados con sus autores y la fecha de inicio y devolución del préstamo, ordenados por la fecha de inicio del préstamo.:)

```
for $t in doc("libros.xml")//libro,
 $e in doc("prestamos.xml")//entrada
where $t/titulo = $e/titulo
order by $e/prestamo/inicio
return <prestamo>{ $t/titulo, $t/autor/*, $e/prestamo/inicio, $e/prestamo/devolucion }</prestamo>
```

(://escrutinio\_sitio[convocatoria='2015']/nombre\_sitio:)

```
(: {
for $a in /escrutinio_sitio
where $a/convocatoria='2019' and $a/votos/contabilizados/porcentaje >= '69'
order by $a/nombre_sitio
```

```

return {$a/nombre_sitio/data()}
}:)

(:let $contenido := (for $a in /escrutinio_sitio
where $a/convocatoria='2019' and $a/votos/contabilizados/porcentaje >= '70'
order by $a/nombre_sitio
return {$a/nombre_sitio/data()})
return {$contenido}:)

(:let $p2019 := distinct-values(for $p in /escrutinio_sitio
where $p/convocatoria=2019
return $p/resultados/partido/nombre)

for $p2015 in distinct-values(for $p in /escrutinio_sitio
where $p/convocatoria=2015
return $p/resultados/partido/nombre)
where not($p2015 = $p2019)

return <partido>{$p2015}</partido>:)
(:For itera cada uno de ellos.
Let almacena conjunto de valores. A la hora de comparar, cuidado porque compara con el
conjunto. A la hora de verificar no compara con el resto:)
(<resultados>
{for $a in ('VOX','PODEMOS-IU')
let $total := sum(/escrutinio_sitio[convocatoria=2019]/
resultados/partido[nombre=$a]/electos)
return <electos partido="{ $a }">{$total}</electos>}
</resultados>:)
for $provincia in /escrutinio_sitio[convocatoria=2019]
let $maximo := max($provincia//partido/votos_numero)
let $partido := $provincia//partido[votos_numero=$maximo]/nombre
order by $provincia/nombre_sitio
return <resultado><provincia>{$provincia/nombre_sitio/data()}</provincia>
<partido>{$partido}</partido>
</resultado>

```