

Comenzado el lunes, 21 de octubre de 2019, 15:55

Estado Finalizado

Finalizado en lunes, 21 de octubre de 2019, 22:32

Tiempo empleado 6 horas 37 minutos

Calificación 9,67 de 10,00 (97%)

Pregunta 1

Correcta

Puntúa 0,67 sobre
1,00

Los _____ de un documento XML están organizados según una estructura jerárquica

Seleccione una:

- a. atributos
- b. prólogos
- c. elementos ✓
- d. Todos los anteriores
- e. Ninguno de los anteriores

La respuesta correcta es: elementos

Correcta

Puntos para este envío: 1,00/1,00. Contando con los intentos anteriores, daría 0,67/1,00.

Pregunta 2

Correcta

Puntúa 1,00 sobre
1,00

Todos los datos de un documento XML han de pertenecer a un elemento ✓ del mismo

La respuesta correcta es: elemento

Correcta

Puntos para este envío: 1,00/1,00.

Pregunta 3

Correcta

Puntúa 1,00 sobre
1,00

Escoge la versión correcta del siguiente prólogo:

Seleccione una:

- a. <?XML version="1.0" standalone="yes" encoding="UTF-8"?>
- b. <?xml version="1.0" encoding="UTF-8" standalone="yes" ?> ✓

La respuesta correcta es: <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

Correcta

Puntos para este envío: 1,00/1,00.

Pregunta 4

Correcta

Puntúa 1,00 sobre
1,00

Todas las marcas han de estar formadas por una etiqueta de inicio y otra de fin

Seleccione una:

- Verdadero ✓
- Falso

La respuesta correcta es 'Verdadero'

Correcta

Puntos para este envío: 1,00/1,00.

Pregunta 5

Correcta

Puntúa 1,00 sobre
1,00

Para crear documentos XML es suficiente:

Seleccione una:

- a. Software especializado para la tecnología XML
- b. Herramientas de validación de XML
- c. Un block de notas y un navegador ✓
- d. Al menos, un editor XML

La respuesta correcta es: Un block de notas y un navegador

Correcta

Puntos para este envío: 1,00/1,00.

Pregunta 6

Correcta

Puntúa 1,00 sobre
1,00

Establecer las relaciones correctas

- | | | | |
|------|--|---|---|
| XML | No exige trabajar con documentos bien formados | ▼ | ✓ |
| SGML | Uso muy complejo | ▼ | ✓ |
| HTML | Utiliza un conjunto limitado de etiquetas | ▼ | ✓ |

La respuesta correcta es: XML – No exige trabajar con documentos bien formados, SGML – Uso muy complejo, HTML – Utiliza un conjunto limitado de etiquetas

Correcta

Puntos para este envío: 1,00/1,00.

Pregunta 7

Correcta

Puntúa 1,00 sobre
1,00

Los lenguajes de marcas descriptivos permiten:

Seleccione una:

- a. Dar formato a los documentos
- b. Definir la estructura de los datos de un documento ✓
- c. Permitir el intercambio de ficheros entre diferentes aplicaciones y plataformas
- d. Todas las anteriores

La respuesta correcta es: Definir la estructura de los datos de un documento

Correcta

Puntos para este envío: 1,00/1,00.

Pregunta 8

Correcta

Puntúa 1,00 sobre
1,00

Un parser **analiza** ✓ el código XML comprobando que el documento cumple las normas establecidas para que pueda abrirse.

La respuesta correcta es: analiza

Correcta

Puntos para este envío: 1,00/1,00.

Pregunta 9

Correcta

Puntúa 1,00 sobre
1,00

Los procesadores XSLT sirven para _____ un documento XML

Seleccione una:

- a. analizar
- b. publicar en Internet ✓
- c. validar
- d. Todos los anteriores
- e. Ninguno de los anteriores

La respuesta correcta es: publicar en Internet

Correcta

Puntos para este envío: 1,00/1,00.

Pregunta 10

Correcta

Puntúa 1,00 sobre
1,00

El siguiente documento XML está "bien formado"

```
<?xml version="1.0"?>

<mensaje>
    <destinatario>Tomas</ destinatario>
    <remitente>Juan</ remitente>
    <asunto>
        <contenido>No olvides ir a recogerme al aeropuerto mañana por la mañana!
        </contenido>
    </mensaje>
```

Seleccione una:

- Verdadero
- Falso ✓

La respuesta correcta es 'Falso'

Correcta

Puntos para este envío: 1,00/1,00.

1.A. Lenguajes de marcas

Sitio: Aula Virtual CIERD (CIDEAD)
Curso: Lenguaje de marcas y sistemas de gestión de información
Libro: 1.A. Lenguajes de marcas
Imprimido por: MANUEL JOSE GONZALEZ CALVO
Día: jueves, 21 de mayo de 2020, 22:33

Tabla de contenidos

- 1 Introducción
- 2 Evolución de los lenguajes de marcas
 - 2.1 GML (Generalized Markup Language)
 - 2.2 SGML (Standard Generalized Markup Language)
 - 2.3 HTML (HyperText Markup Language)
 - 2.4 XML (eXtensible Markup Language)
 - 2.5 XML vs HTML
 - 2.6 Comparación de XML con SGML
 - 2.7 Etiquetas
 - 2.8 Herramientas de edición

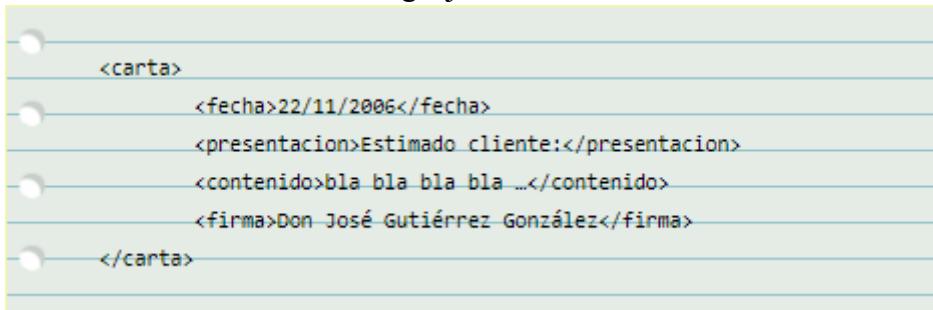
1 Introducción

Un "lenguaje de marcas" es un modo de codificar un documento donde, junto con el texto, se incorporan etiquetas, marcas o anotaciones con información adicional relativa a la estructura del texto o su formato de presentación. Permiten hacer explícita la estructura de un documento, su contenido semántico o cualquier otra información lingüística o extralingüística que se quiera hacer patente.

Todo lenguaje de marcas está definido en un documento denominado **DTD (Document Type Definition)**. En él se establecen las marcas, los elementos utilizados por dicho lenguaje y sus correspondientes etiquetas y atributos, su sintaxis y normas de uso.

Ejemplo

Aspecto de un documento realizado en un lenguaje de marcas:



En la práctica, en un mismo documento pueden combinarse varios tipos diferentes de lenguajes de marcas.

Los lenguajes de marcas se pueden clasificar como sigue:

- De presentación: Define el formato del texto.
- De procedimientos: Orientado también a la presentación pero, en este caso, el programa que representa el documento debe interpretar el código en el mismo orden en que aparece.
- Descriptivo o semántico: Describen las diferentes partes en las que se estructura el documento pero sin especificar cómo deben representarse.

Algunos ejemplos de lenguajes de marcado agrupados por su ámbito de utilización son:

Documentación electrónica:

- **RTF (Rich Text Format)**: Formato de Texto Enriquecido, fue desarrollado por Microsoft en 1987. Permite el intercambio de documentos de texto entre distintos procesadores de texto.
- **TeX**: Su objetivo es la creación de ecuaciones matemáticas complejas.
- **Wikitexto**: Permite la creación de páginas wiki en servidores preparados para soportar este lenguaje.
- **DocBook**: Permite generar documentos separando la estructura lógica del documento de su formato. De este modo, dichos documentos, pueden publicarse en diferentes formatos sin necesidad de realizar modificaciones en el documento original.

Tecnologías de internet:

- **HTML, XHTML**: (Hypertext Markup Language, eXtensible Hypertext Markup Language): Su objetivo es la creación de páginas web.
- **RSS**: Permite la difusión de contenidos web

Otros lenguajes especializados:

- **MathML (Mathematical Markup Language)**: Su objetivo es expresar el formalismo matemático de tal modo que pueda ser entendido por distintos sistemas y aplicaciones.
- **VoiceXML (Voice Extended Markup Language)** tiene como objetivo el intercambio de información entre un usuario y una aplicación con capacidad de reconocimiento de habla.

- **MusicXML:** Permite el intercambio de partituras entre distintos editores de partituras.

2 Evolución de los lenguajes de marcas

En los años 70 surgieron unos lenguajes informáticos, distintos de los lenguajes de programación, orientados a la gestión de información. Con el desarrollo de los editores y procesadores de texto aparecieron los primeros lenguajes informáticos especializados en tareas de descripción y estructuración de información: los lenguajes de marcas. Paralelamente, también, emergieron otros lenguajes informáticos, orientados a la representación, almacenamiento y consulta eficiente de grandes cantidades de datos: lenguajes y sistemas de bases de datos.

Los lenguajes de marcas surgieron, inicialmente, como lenguajes formados por el conjunto de códigos de formato que los procesadores de texto introducen en los documentos para dirigir el proceso de presentación (impresión) mediante una impresora. Como en el caso de los lenguajes de programación, inicialmente estos códigos de formato estaban ligados a las características de una máquina, programa o procesador de textos concreto y, en ellos, inicialmente no había nada que permitiese al programador (formateador de documentos en este caso) abstraerse de las características del procesador de textos y expresar de forma independiente a éste la estructura y la lógica interna del documento.

Ejemplo

Código de marcas anterior a GML. Las etiquetas son de invención propia.

Dado el siguiente documento:

```
<times 14><color verde><centrado> Este texto es un ejemplo para mostrar la utilización primitiva de las marcas</centrado></color></times 14>
<color granate><times 10><cursive>Para realizar este ejemplo se utilizan etiquetas de nuestra invención. </cursive>
Las partes importantes del texto pueden resaltarse usando la
<negrita>negrita</negrita>, o el <subrayar>subrayado</subrayar></times 10></color>
```

Al imprimirlo se obtendría:

Este texto es un ejemplo para mostrar la utilización primitiva de las marcas

Para realizar este ejemplo se utilizan etiquetas de nuestra invención. Las partes importantes del texto pueden resaltarse usando la negrita , o el subrayado

Posteriormente, se añadieron este tipo de características, como medio de presentación a la pantalla. Los códigos de estilo de visualización anteriores ya no aparecían, y se empleaban otros medios para marcados, distintos de la inclusión a mano de cadenas formateadoras. Ese proceso se automatizó y bastaba pulsar una combinación de teclas, o un botón, para lograr los resultados requeridos. Aunque esto era sólo una abstracción, para su uso interno, las aplicaciones seguían utilizando marcas, para delimitar aquellas partes del texto que tenían un formato especial.

Este marcado estaba exclusivamente orientado a la presentación de la información, aunque pronto se percataron de las posibilidades del marcado y le dieron nuevos usos que resolvían una gran variedad de necesidades. De este modo apareció el formato generalizado.

2.1 GML (Generalized Markup Language)

Uno de los problemas que se conocen desde hace décadas en la informática es la falta de estandarización en los formatos de información usados por los distintos programas.

Para resolver este problema, en los años sesenta **IBM** encargó a Charles F. Goldfarb la construcción de un sistema de edición, almacenamiento y búsqueda de documentos legales. Tras analizar el funcionamiento de la empresa llegaron a la conclusión de que para realizar un buen procesado informático de los documentos había que establecer un formato estándar para todos los documentos que se manejaban en la empresa. Con ello se lograba gestionar cualquier documento en cualquier departamento y con cualquier aplicación, sin tener en cuenta dónde ni con qué se generó el documento. Dicho formato tenía que ser válido para los distintos tipos de documentos legales que utilizaba la empresa, por tanto, debía ser flexible para que se pudiera ajustar a las distintas situaciones.

El formato de documentos que se creó como resultado de este trabajo fue **GML**, cuyo objetivo era describir los documentos de tal modo que el resultado fuese independiente de la plataforma y la aplicación utilizada.

2.2 SGML (Standard Generalized Markup Language)

El formato **GML** evolucionó hasta que en 1986 dio lugar al estándar **ISO 8879** que se denominó **SGML**. Éste era un lenguaje muy complejo y requería de unas herramientas de software caras. Por ello su uso ha quedado relegado a grandes aplicaciones industriales.

Ejemplo: Documento GML sencillo

```
<email>
  <remitente>
    <persona>
      <nOMBRE>Pepito</nOMBRE>
      <apellido>Grillo</apellido>
    </persona>
  </remitente>
  <destinatario>
    <dIRECCION>pinocho@hotmail.com</dIRECCION>
  </destinatario>
  <asunto>¿quedamos?</asunto>
  <mensaje>Hola, he visto que ponen esta noche la película que querías ver. ¿Te apetece ir?</mensaje>
</email>
```

2.3 HTML (HyperText Markup Language)

En 1989/90 Tim Berners-Lee creó el **World Wide Web** y se encontró con la necesidad de organizar, enlazar y compatibilizar gran cantidad de información procedente de diversos sistemas. Para resolverlo creó un lenguaje de descripción de documentos llamado **HTML**, que, en realidad, era una combinación de dos estándares ya existentes:

- **ASCII**: Es el formato que cualquier procesador de textos sencillo puede reconocer y almacenar. Por tanto es un formato que permite la transferencia de datos entre diferentes ordenadores.
- **SGML**: Lenguaje que permite dar estructura al texto, resaltando los títulos o aplicando diversos formatos al texto.

HTML es una versión simplificada de **SGML**, ya que sólo se utilizaban las instrucciones absolutamente imprescindibles. Era tan fácil de comprender que rápidamente tuvo gran aceptación, logrando lo que no pudo **SGML**.

HTML se convirtió en un estándar general para la creación de páginas web. Además, desde su creación, tanto las herramientas de software como los navegadores, que permiten visualizar páginas **HTML**, no han parado de mejorar.

A pesar de todas estas ventajas **HTML** no es un lenguaje perfecto, sus principales desventajas son:

- No soporta tareas de impresión y diseño.
- El lenguaje no es flexible, ya que las etiquetas son limitadas.
- No permite mostrar contenido dinámico.
- La estructura y el diseño están mezclados en el documento.

Ejemplo: Documento HTML

```
<html>
  <head>
    <title> Ejemplo de código HTML</title>
  </head>
  <body bgcolor="#ffffff">
    <p></p>
    <p>
      <b>20 de octubre de 2010</b>
    </p>
    <p><b> Bienvenido al modulo de "Lenguajes de Marcas y Sistemas de Gestión de Información" </b></p>
    <p> En este curso aprenderás, entre otras cosas:<br/>
      <ul>
        <li>Las ventajas que ofrece XML </li>
        <li>La creación de documentos bien formados </li>
        <li>La creación de DTD</li>
      </ul>
    </p>
  </body>
</html>
```

En el navegador se visualizaría así:



2.4 XML (eXtensible Markup Language)

Para resolver estos problemas de **HTML** el **W3C** establece, en 1998, el estándar internacional **XML**, un lenguaje de marcas puramente estructural que **no incluye ninguna información relativa al diseño**. Está convirtiéndose con rapidez en estándar para el intercambio de datos en la Web. A diferencia de **HTML** las etiquetas indican el significado de los datos en lugar del formato con el que se van a visualizar los datos.

XML es un metalenguaje caracterizado por:

- Permitir definir etiquetas propias.
- Permitir asignar atributos a las etiquetas.
- Utilizar un esquema para definir de forma exacta las etiquetas y los atributos.
- La estructura y el diseño son independientes.

En realidad **XML** es un conjunto de estándares relacionados entre sí y que son:

- **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.

Ejemplo: Documento XML

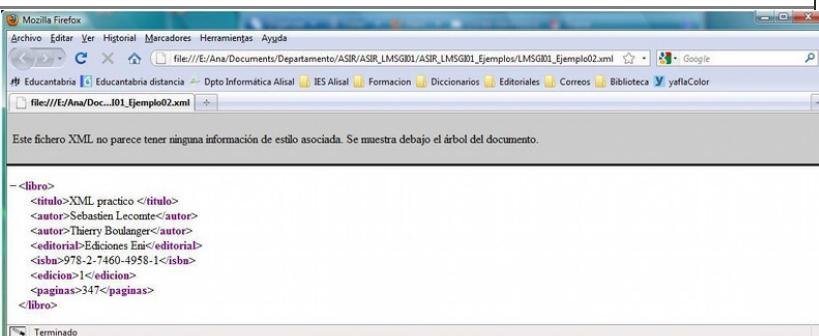
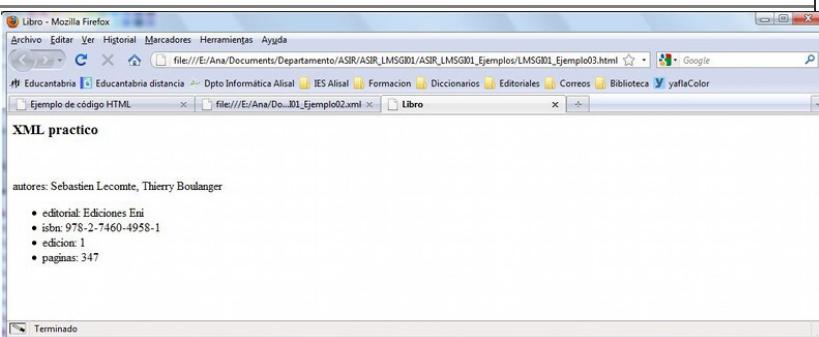
```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE biblioteca>
<biblioteca>
    <ejemplar tipo_ejem="libro" titulo="XML practico" editorial="Ediciones Eni">
        <tipo><libro isbn="978-2-7460-4958-1" edicion="1" paginas="347"></libro></tipo>
        <autor nombre="Sebastien Lecomte"></autor>
        <autor nombre="Thierry Boulanger"></autor>
        <autor nombre="Ángel Belinchon Calleja" funcion="traductor"></autor>
        <prestado lector="Pepito Grillo">
            <fecha_pres dia="13" mes="mar" año="2009"></fecha_pres>
            <fecha_devol dia="21" mes="jun" año="2009"></fecha_devol>
        </prestado>
    </ejemplar>
    <ejemplar tipo_ejem="revista" titulo="Todo Linux 101. Virtualización en GNU/Linux" editorial="Studio Press">
        <tipo>
            <revista>
                <fecha_publicacion mes="abr" año="2009"></fecha_publicacion>
            </revista>
        </tipo>
        <autor nombre="Varios"></autor>
        <prestado lector="Pedro Picapiedra">
            <fecha_pres dia="12" mes="ene" año="2010"></fecha_pres>
        </prestado>
    </ejemplar>
</biblioteca>
```

2.5 XML vs HTML

A continuación encontrarás una tabla comparativa de ambos lenguajes

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 hiperenlaces complejo.	<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.

Ejemplo: XML vs HTML

Código	Visualización en navegador
<pre><?xml version="1.0" encoding="iso-8859-1"?> <!DOCTYPE libro> <libro> <titulo>XML practico </titulo> <autor>Sebastien Lecomte</autor> <autor>Thierry Boulanger</autor> <editorial>Ediciones Eni</editorial> <isbn>978-2-7460-4958-1</isbn> <edicion>1</edicion> <paginas>347</paginas> </libro></pre>	 <p>Mozilla Firefox Este fichero XML no parece tener ninguna información de estilo asociada. Se muestra debajo el árbol del documento.</p>
<pre><html> <head> <title>Libro</title> </head> <body> <h3>XML practico</h3>
 <p>autores: Sebastien Lecomte,
 Thierry Boulanger</p> editorial: Ediciones Eni isbn: 978-2-7460-4958-1 edicion: 1 paginas: 347 </body> </html></pre>	 <p>Libro - Mozilla Firefox XML practico</p> <p>autores: Sebastien Lecomte, Thierry Boulanger</p> <ul style="list-style-type: none">editorial: Ediciones Eniisbn: 978-2-7460-4958-1edicion: 1paginas: 347

2.6 Comparación de XML con SGML

XML	SGML
Uso sencillo	Uso complejo
Trabaja con documentos bien formados. No exige que estén validados	Solo trabaja con documentos válidos
Facilita el desarrollo de aplicaciones de bajo coste	Su complejidad hace que las aplicaciones informáticas para procesar SGML sean muy costosas
Es muy utilizado en informática y en más áreas de aplicación	Solo se utiliza en sectores muy específicos
Compatibilidad e integración con HTML	No hay una compatibilidad con HTML definida
Formato y estilos fáciles de aplicar	Formateo y estilos relativamente complejos
No usa etiquetas optionales	

Para saber más

La recomendación de **XML** publicada por el W3C es pública y accesible en:

<https://www.w3.org/TR/xml/>

2.7 Etiquetas

Los lenguajes de marcas utilizan una serie de etiquetas especiales intercaladas en un documento de texto sin formato. Dichas etiquetas serán posteriormente interpretadas por los intérpretes del lenguaje y ayudan al procesado del documento.

Las etiquetas **se escriben encerradas entre ángulos**, es decir < y >. Normalmente, **se utilizan dos etiquetas: una de inicio y otra de fin** para indicar que ha terminado el efecto que queríamos presentar. La única diferencia entre ambas es que la de cierre lleva una barra inclinada "/" antes del código.

<etiqueta>texto que sufrirá las consecuencias de la etiqueta</etiqueta>

Ejemplo: Etiqueta HTML de subrayado (Underline)



En el navegador el texto se verá:

Esto está subrayado

Las últimas especificaciones emitidas por el **W3C** indican la necesidad de que vayan escritas **siempre en minúsculas** para considerar que el documento está correctamente creado.

2.8 Herramientas de edición

Para trabajar en **XML** es necesario editar los documentos y luego procesarlos, por tanto tenemos dos tipos de herramientas:

Editores XML

Una característica de los lenguajes de marcas es que se basan en la utilización de ficheros de texto plano por lo que basta utilizar un procesador de texto normal y corriente para construir un documento **XML**.

Para crear documentos **XML** complejos e ir añadiendo datos es conveniente usar algún editor **XML**. Estos nos ayudan a crear estructuras y etiquetas de los elementos usados en los documentos, además algunos incluyen ayuda para la creación de otros elementos como **DTD**, hojas de estilo **CSS** o **XSL**, ... El W3C ha desarrollado un editor de **HTML**, **XHTML**, **CSS** y **XML** gratuito cuyo nombre es Amaya (<https://www.w3.org/Amaya/>).

Procesadores XML

Los procesadores **XML** permiten leer los documentos **XML** y acceder a su contenido y estructura. Un procesador es un conjunto de módulos de software, entre los que se encuentra un **parser** o analizador de **XML**, que comprueba que el documento cumple las normas establecidas para que pueda abrirse.

Los procesadores **XML** pueden obliguen a trabajar sólo con documentos de tipo válido (entonces se denominan "validadores") o pueden sólo exigir que el documento esté bien formado ("no validadores").

El modo en que los procesadores deben leer los datos **XML** está descrito en la recomendación de **XML** establecida por **W3C**.

Para publicar un documento **XML** en Internet se utilizan los procesadores **XSLT**, que permiten generar archivos **HTML** a partir de documentos **XML**.

Para interpretar el código **XML** se puede utilizar cualquier navegador.

XML también se puede utilizar para el intercambio de datos entre aplicaciones. En este caso, hay que recurrir a motores independientes, que se ejecutan sin que nos demos cuenta. Por ejemplo JAXP de Sun.

Para saber más

Información sobre analizadores XML - <http://xml.coverpages.org/index.html>

expat - XML Parser Toolkit - <http://www.jclark.com/xml/expat.html>

1.B. XML, estructura y sintaxis

Sitio: Aula Virtual CIERD (CIDEAD)
Curso: Lenguaje de marcas y sistemas de gestión de información
Libro: 1.B. XML, estructura y sintaxis
Imprimido por: MANUEL JOSE GONZALEZ CALVO
Día: jueves, 21 de mayo de 2020, 22:35

Tabla de contenidos

- 1 Introducción
- 2 El prólogo
- 3 El ejemplar. Los elementos.
 - 3.1 Atributos
 - 3.2 Ejercicio: Detectar errores
- 4 Documentos XML bien formados
- 5 Utilización de espacios de nombres en XML
 - 5.1 Ejemplo: Utilización de espacios de nombres

1 Introducción

El **XML**, o Lenguaje de Etiquetas Extendido, es lenguaje de etiquetas, creadas por el programador, que estructuran y guardan de forma ordenada la información. No representa datos por sí mismo, solamente organiza la estructura.

El **XML** ahorra tiempos de desarrollo y proporciona ventajas, dotando a webs y a aplicaciones de una forma realmente potente de guardar la información. Además, se ha convertido en un formato universal que ha sido asimilado por todo tipo de sistemas operativos y dispositivos móviles.

Al igual que en **HTML** un documento **XML** es un documento de texto, en este caso con extensión ".xml", compuesto de parejas de etiquetas, estructuradas en árbol, que describen una función en la organización del documento, que puede editarse con cualquier editor de texto y que es interpretado por los navegadores Web.

Las características básicas de **XML** son:

- Dado que **XML** se concibió para trabajar en la Web, es directamente compatible con protocolos que ya funcionan, como **HTTP** y los **URL**.
- Todo documento que verifique las reglas de **XML** está conforme con **SGML**.
- No se requieren conocimientos de programación para realizar tareas sencillas en **XML**.
- Los documentos **XML** son fáciles de crear.
- La difusión de los documentos **XML** está asegurada ya que cualquier procesador de **XML** puede leer un documento de **XML**.
- El marcado de **XML** es legible para los humanos.
- El diseño **XML** es formal y conciso.
- **XML** es extensible, adaptable y aplicable a una gran variedad de situaciones.
- **XML** es orientado a objetos.
- Todo documento **XML** se compone exclusivamente de datos de marcado y datos carácter entremezclados.

El proceso de creación de un documento **XML** pasa por varias etapas en las que el éxito de cada una de ellas se basa en la calidad de la anterior. Estas etapas son:

- Especificación de requisitos.
- Diseño de etiquetas.
- Marcado de los documentos.

El marcado en **XML** son etiquetas que se añaden a un texto para estructurar el contenido del documento. Esta información extra permite a los ordenadores "interpretar" los textos. El marcado es todo lo que se sitúa entre los caracteres "<" y ">" o "&" y ";"

Los datos carácter son los que forman la verdadera información del documento **XML**.

El marcado puede ser tan rico como se quiera. Puede ser interesante detectar necesidades futuras y crear documentos con una estructura fácilmente actualizables.

Los documentos **XML** pueden tener comentarios, que no son interpretados por el interprete **XML**. Estos se incluyen entre las cadenas "<!--" y "-->", pueden estar en cualquier posición en el documento salvo:

- Antes del prólogo.
- Dentro de una etiqueta.

Los documentos **XML** pueden estar formados por una parte opcional llamada prólogo y otra parte obligatoria llamada ejemplar.

2 El prólogo

Si se incluye, el prólogo **debe preceder al ejemplar del documento**. Su inclusión facilita el procesado de la información del ejemplar. El prólogo está dividido en dos partes:

- **La declaración XML:** En el caso de incluirse ha de ser la primera línea del documento, de no ser así se genera un error que impide que el documento sea procesado.

El hecho de que sea opcional permite el procesamiento de documentos **HTML** y **SGML** como si fueran **XML**, si fuera obligatoria éstos deberían incluir una declaración de versión XML que no tienen.

El prólogo puede tener tres funciones:

- *Declaración la versión de XML usada para elaborar el documento.*

Para ello se utiliza la etiqueta:

```
<?xml versión= "1.0" ?>
```

En este caso indica que el documento fue creado para la versión 1.0 de XML.

- *Declaración de la codificación empleada para representar los caracteres.*

Determina el conjunto de caracteres que se utiliza en el documento. Para ello se escribe:

```
<?xml versión= "1.0" encoding="iso-8859-1" ?>
```

En este caso se usa el código iso-8859-1 (Latin-1) que permite el uso de acentos o caracteres como la ñ.

Los códigos más importantes son:

Estándar ISO	Código de país
UTF-8 (Unicode)	Conjunto de caracteres universal
ISO -8859-1 (Latin-1)	Europa occidental, Latinoamérica
ISO -8859-2 (Latin-2)	Europa central y oriental
ISO -8859-3 (Latin-3)	Sudoeste de Europa
ISO -8859-4 (Latin-4)	Países Escandinavos, Bálticos
ISO -8859-5	Cirílico
ISO -8859-6	Árabe
ISO -8859-7	Griego
ISO -8859-8	Hebreo
ISO -8859-9	Turco
ISO-8859-10	Lapon. Nórdico, esquimal
EUC-JP oder Shift_JIS	Japonés

- *Declaración de la autonomía del documento.*

Informa de si el documento necesita de otro para su interpretación. Para declararlo hay que definir el prólogo completo:

```
<?xml versión= "1.0" encoding="iso-8859-1" standalone="yes" ?>
```

En este caso, el documento es independiente, de no ser así el atributo standalone hubiese tomado el valor "no".

- **La declaración del tipo de documento,** define qué tipo de documento estamos creando para ser procesado correctamente. Toda declaración de tipo de documento comienza por la cadena:

```
<!DOCTYPE Nombre_tipo ...>
```


3 El ejemplar. Los elementos.

Es la parte más importante de un documento XML, ya que **contiene los datos reales del documento**. Está formado por elementos anidados.

Los elementos son los distintos bloques de información que permiten definir la estructura de un documento XML. Están delimitados por una etiqueta de apertura y una etiqueta de cierre. A su vez, los elementos pueden estar formados por otros elementos y/o por atributos.

Ejemplo: Dado el siguiente código XML

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE libro>
<libro>
    <titulo>XML practico </titulo>
    <autor>Sebastien Lecomte</autor>
    <autor>Thierry Boulanger</autor>
    <editorial>Ediciones Enik</editorial>
    <isbn>978-2-7460-4958-1</isbn>
    <edicion>1</edicion>
    <pginas>347</pginas>
</libro>
```

El ejemplar es el elemento <libro>, que a su vez está compuesto de los elementos <titulo>, <autor>, <editorial>, <isbn>, <edicion> y <pginas>.

En realidad, **el ejemplar es el elemento raíz (root)** de un documento XML. **Todos los datos** de un documento XML **han de pertenecer a un elemento** del mismo.

Los nombres de las etiquetas han de ser autodescriptivos, lo que facilita el trabajo que se hace con ellas.

La formación de elementos ha de cumplir ciertas normas para que queden perfectamente definidos y que el documento XML al que pertenecen pueda ser interpretado por los procesadores XML sin generar ningún error fatal. Dichas reglas son:

- En todo documento XML debe existir un elemento raíz, y sólo uno.
- Todos los elementos tienen una etiqueta de inicio y otra de cierre. En el caso de que en el documento existan elementos vacíos, se pueden sustituir las etiquetas de inicio y cierre por una de elemento vacío. Ésta se construye como la etiqueta de inicio, pero sustituyendo el carácter ">" por "/>. Es decir, <elemento></elemento> puede sustituirse por: <elemento/>
- Al anidar elementos hay que tener en cuenta que no puede cerrarse un elemento que contenga algún otro elemento que aún no se haya cerrado.
- Los nombres de las etiquetas de inicio y de cierre de un mismo elemento han de ser idénticos, respetando las mayúsculas y minúsculas. Pueden ser cualquier cadena alfanumérica que no contenga espacios y no comience ni por el carácter dos puntos, ":" , ni por la cadena "xml" ni ninguna de sus versiones en que se cambien mayúsculas y minúsculas ("XML", "XmL", "xML",....).
- El contenido de los elementos no puede contener la cadena "]]>" por compatibilidad con SGML. Además no se pueden utilizar directamente los caracteres mayor que, >, menor que, <, ampersand, &, dobles comillas, ", y apostrofe, '. En el caso de tener que utilizar estos caracteres se sustituyen por las siguientes cadenas:

Carácter	Cadena
>	>
<	<
&	&
"	"
'	'

- Para utilizar caracteres especiales, como £, ©, ®,... hay que usar las expresiones &#D; o &#H; donde D y H se corresponden respectivamente con el número decimal o hexadecimal correspondiente al carácter que se quiere representar en el código **UNICODE**. Por ejemplo, para incluir el carácter de Euro, €, se usarían las cadenas € o €

3.1 Atributos

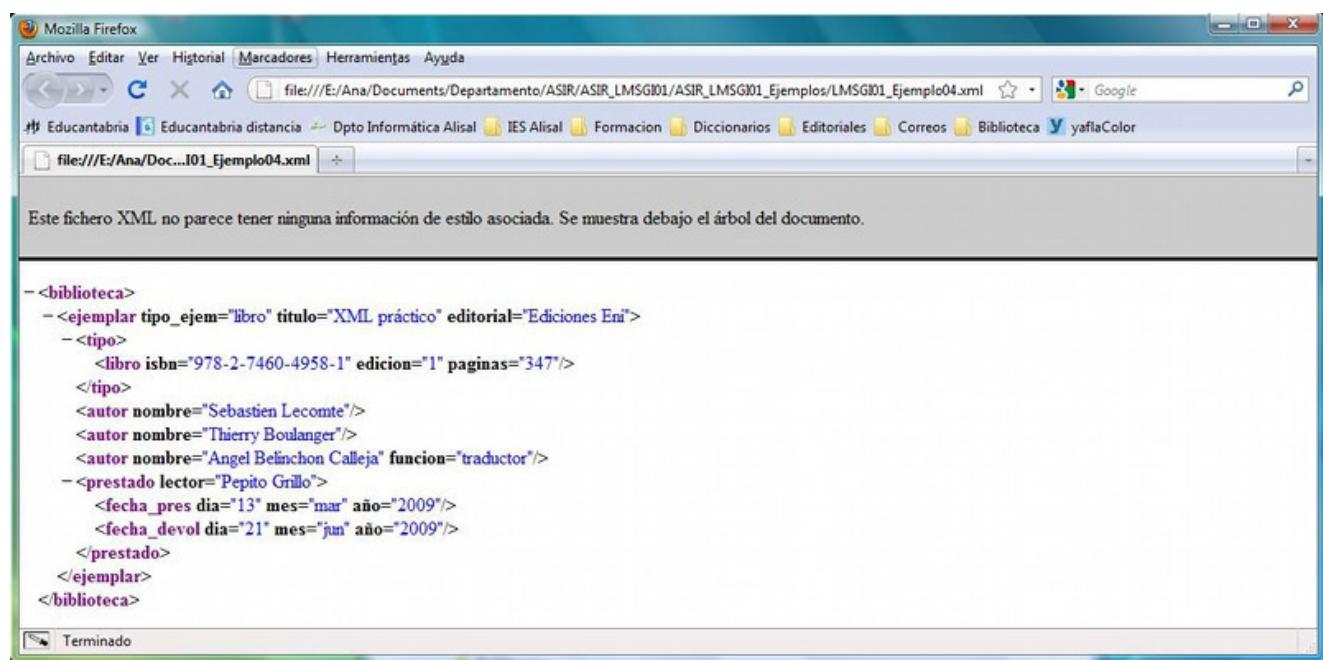
Permiten añadir propiedades a los elementos de un documento. Los atributos no pueden organizarse en ninguna jerarquía, no pueden contener ningún otro elemento o atributo y no reflejan ninguna estructura lógica.

No se debe utilizar un atributo para contener información susceptible de ser dividida.

Ejemplo: Dado el siguiente código XML

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes" ?>
<!DOCTYPE biblioteca >
<biblioteca>
  <ejemplar tipo_ejem="libro" titulo="XML práctico" editorial="Ediciones Eni">
    <tipo> <libro isbn="978-2-7460-4958-1" edicion="1" paginas="347"></libro> </tipo>
    <autor nombre="Sebastien Lecomte"></autor>
    <autor nombre="Thierry Boulanger"></autor>
    <autor nombre="Angel Belinchon Calleja" funcion="traductor"></autor>
    <prestado lector="Pepito Grillo">
      <fecha_pres dia="13" mes="mar" año="2009"></fecha_pres>
      <fecha_devol dia="21" mes="jun" año="2009"></fecha_devol>
    </prestado>
  </ejemplar>
</biblioteca>
```

Al abrir el documento anterior con el navegador obtendremos:



Los nombres de los elementos aparecen en color morado, los atributos en negro y sus valores en azul.

Como se observa en el ejemplo, los atributos se definen y dan valor dentro de una etiqueta de inicio o de elemento vacío, a continuación del nombre del elemento o de la definición de otro atributo, siempre separado de ellos por un espacio. Los valores del atributo van precedidos de un igual que sigue al nombre del mismo y tienen que definirse entre comillas simples o dobles.

Los nombres de los atributos han de cumplir las mismas reglas que los de los elementos, y no pueden contener el carácter menor que, <.

3.2 Ejercicio: Detectar errores

Indica cuál de los errores indicados aparecen en el siguiente documento XML.

```
<?XML version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE biblioteca >
<biblioteca>
  <ejemplar tipo_ejem='libro' titulo='XML práctico' editorial='Ediciones Eni'>
    <tipo> <libro isbn='978-2-7460-4958-1' edicion= paginas='347'></libro> </tipo>
    <autor nombre='Sebastien Lecomte'></autor>
    <autor nombre='Thierry Boulanger'></autor>
    <autor nombre='Angel Belinchon Calleja' funcion='traductor'></autor>
    <prestado lector='Pepito Grillo'>
      <fecha_pres dia='13' mes='mar' año='2009'></fecha_pres>
      <fecha_devol/>
    </prestado>
  </ejemplar>
</biblioteca>
```

Deja aquí tu respuesta

4 Documentos XML bien formados

Todos los documentos **XML** deben verificar las reglas sintácticas que define la recomendación del **W3C** para el estándar **XML**. Esas normas básicas son:

- El documento ha de tener definido un prólogo con la declaración xml completa.
- Existe un único elemento raíz para cada documento: es un solo elemento en el que todos los demás elementos y contenidos se encuentran anidados.
- Hay que cumplir las reglas sintácticas del lenguaje **XML** para definir los distintos elementos y atributos del documento

Ejemplo - ¿Está bien formado el siguiente documento XML?

```
<?xml version="1.0"?>
<mensaje>
    <destinatario>Tomas</destinatario>
    <remitente>Juan</remitente>
    <asunto>
        <contenido>No olvides ir a recogerme al aeropuerto mañana por la mañana!</contenido>
    </asunto>
</mensaje>
```

No, la etiqueta `<asunto>` sigue abierta y el prólogo no tiene una declaración XML completa.

5 Utilización de espacios de nombres en XML

Permiten definir la pertenencia de los elementos y los atributos de un documento XML al contexto de un vocabulario XML. De este modo se resuelven las ambigüedades que se pueden producir al juntar dos documentos distintos, de dos autores diferentes, que han utilizado el mismo nombre de etiqueta para representar cosas distintas.

Los espacios de nombres, también conocidos como "name spaces", permiten dar un nombre único a cada elemento, indexándolos según el nombre del vocabulario adecuado. Además están asociados a un URI que los identifica de forma única.

En el documento, las etiquetas ambiguas se sustituyen por otras en las que el nombre del elemento está precedido de un prefijo, que determina el contexto al que pertenece la etiqueta, seguido de dos puntos, ":". Esto es:

```
<prefijo:nombre_etiqueta></prefijo:nombre_etiqueta>
```

Esta etiqueta se denomina "nombre cualificado". Al definir el prefijo hay que tener en cuenta que no se pueden utilizar espacios ni caracteres especiales y que no puede comenzar por un dígito.

Antes de poder utilizar un prefijo de un espacio de nombres, para resolver la ambigüedad de dos o más etiquetas, es necesario declarar el espacio de nombres, es decir, asociar un índice con el URI asignado al espacio de nombres, mediante un atributo especial xmlns. Esto se hace entre el prólogo y el ejemplar de un documento XML y su sintaxis es la siguiente:

```
<conexion>://<direccionservidor>/<apartado1>/<apartado2>/...
```

Para saber más

Los espacios de nombres tienen una recomendación en XML - <http://www.w3.org/TR/REC-xml-names/>

5.1 Ejemplo: Utilización de espacios de nombres

Supongamos dos documentos que organizan la información sobre los profesores y los alumnos del Ciclo Formativo.

XML de alumnos	XML de profesores
<pre><?xml version="1.0" encoding="iso-8859-1" standalone="yes" ?> <!DOCTYPE alumnos> <alumnos> <nombre>Fernando Fernández González</nombre> <nombre>Isabel González Fernández</nombre> <nombre>Ricardo Martínez López</nombre> </alumnos></pre>	<pre><?xml version="1.0" encoding="iso-8859-1" standalone="yes" ?> <!DOCTYPE profesores> <profesores> <nombre>Pilar Ruiz Pérez</nombre> <nombre>Tomás Rodríguez Hernández</nombre> </profesores></pre>

Si uniéramos los dos documentos en uno único, sin usar espacios de nombres, no se distinguirían los profesores de los alumnos ya que en los dos casos la etiqueta `<nombre>` se llama igual.

Para resolverlo necesitamos definir un espacio de nombres para cada contexto:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes" ?>
<!DOCTYPE miembros>
<miembros>
  <alumnos xmlns:alumnos="http://DAM/alumnos">
    <alumnos:nombre>Fernando Fernández González</alumnos:nombre>
    <alumnos:nombre>Isabel González Fernández</alumnos:nombre>
    <alumnos:nombre>Ricardo Martínez López</alumnos:nombre>
  </alumnos>
  <profesores xmlns:profesores="http://DAM/profesores">
    <profesores:nombre>Pilar Ruiz Pérez</profesores:nombre>
    <profesores:nombre>Tomás Rodríguez Hernández</profesores:nombre>
  </profesores>
</miembros>
```

2.A. Utilización de lenguajes de marcas en entornos web

Sitio: Aula Virtual CIERD (CIDEAD)
Curso: Lenguaje de marcas y sistemas de gestión de información
Libro: 2.A. Utilización de lenguajes de marcas en entornos web
Imprimido por: MANUEL JOSE GONZALEZ CALVO
Día: jueves, 21 de mayo de 2020, 23:03

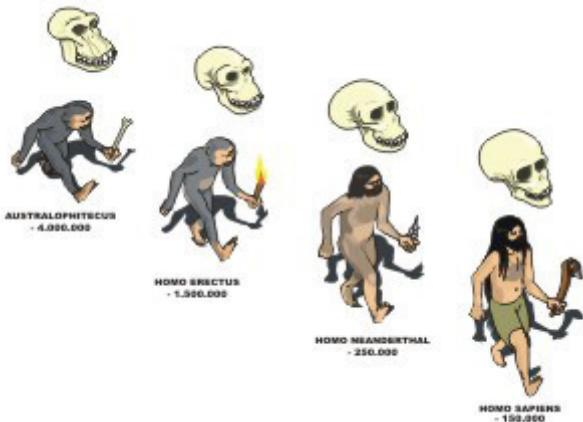
Tabla de contenidos

- 1 HTML: evolución y versiones
- 2 HTML y XHTML
- 3 Estructura de un documento HTML
- 4 Identificación de etiquetas y atributos HTML
- 5 Clasificación de los atributos comunes según su funcionalidad

1 HTML: 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, por lo tanto, todos ellos visualizan una página **HTML** de forma muy similar independientemente del sistema operativo sobre el que se ejecutan.

El origen de HTML fue un sistema de hipertexto para compartir documentos electrónicos en 1990. La primera propuesta oficial para convertir **HTML** en un estándar se realizó en 1993. Ninguna de las dos primeras propuestas de estándar que se hicieron (**HTML** y **HTML+**) consiguieron convertirse en estándares oficiales.



- **HTML 2.0** fue la primera versión oficial de HTML. El IETF publicó el estándar en septiembre de 1995.
- **HTML 3.2** se publicó el 14 de Enero de 1997, por el W3C. Incorporaba los applets de Java y texto alrededor de las imágenes.
- **HTML 4.0** se publicó el 24 de Abril de 1998. Entre las novedades que presentaba se encontraban las hojas de estilos CSS y la posibilidad de incluir pequeños programas en las páginas web.
- **HTML 4.01** se publicó el 24 de diciembre de 1999, como actualización de la versión anterior. En ese momento el W3C detuvo la actividad de estandarización de HTML hasta marzo de 2007, momento en que se retoma debido a la fuerza de las empresas que forman el grupo WHATWG y a la publicación de los borradores de **HTML 5.0**, que será la siguiente versión de este lenguaje.
- **HTML 5** es la versión más avanzada y la que se considera estándar actualmente, aunque ha ido evolucionando por diferentes especificaciones (HTML 5.1 y HTML 5.2). Es una versión viva en la que se sigue trabajando y su estándar actual (HTML 5.2) fue definido el 14 de diciembre de 2017. Puedes encontrar las especificaciones de la última versión de HTML en el siguiente enlace.

Uno de los cambios más importantes en HTML 5 es que todo lo relativo a la presentación del documento se pasa del HTML a las hojas de estilo CSS. Es decir, cuestiones como colores, fondos, tamaño o posicionamiento se especifican mediante CSS. El HTML se encarga de la información que se quiere mostrar, su estructura y su semántica. Esto hace que muchos elementos y atributos de HTML 4 hayan quedado obsoletos (****, **<center>**, **align**,...).

2 HTML y XHTML

El lenguaje



XHTML (EXtensible HyperText Markup Language) es muy similar al lenguaje **HTML**. De hecho, no es más que una adaptación de **HTML** al lenguaje **XML**.

El estándar **XHTML 1.0** sólo añade pequeñas mejoras y modificaciones menores al estándar **HTML 4.01**, por lo que este último está prácticamente incluido en el primero. Pasar del **HTML 4.01 Strict** a **XHTML** no requiere casi ningún cambio.

El lenguaje **HTML** tiene una sintaxis muy permisiva, por lo que es posible escribir sus etiquetas y atributos de muchas formas diferentes. Las etiquetas, por ejemplo, pueden escribirse en mayúsculas, minúsculas e incluso combinando mayúsculas y minúsculas. El valor de los atributos de las etiquetas se pueden indicar con o sin comillas. Además, el orden en el que se abrían y cerraban las etiquetas no era importante.

La flexibilidad de **HTML** da lugar a páginas con un código desordenado, difícil de mantener y muy poco profesional.

XHTML soluciona estos problemas añadiendo ciertas normas en la forma de escribir las etiquetas y atributos. Son las normas que vimos en el capítulo 1 para que un fichero XML se considere bien formado. Así se consigue:

- Sencillez a la hora de editar y mantener el código.
- Al ser más regular, es más fácil escribir código que lo procese.
- Como es **XML** se pueden utilizar fácilmente herramientas creadas para procesar documentos **XML** genéricos (editores, **XSLT**, etc.).

Diferencias sintácticas y estructurales con HTML

Para que el código HTML se pueda considerar XML bien formado, y por tanto, XHTML, tiene que cumplir:

- Toda la página debe estar contenida en un elemento raíz, **<html>**.
- Los nombres de las etiquetas y atributos siempre se escriben en minúsculas.
- El valor de los atributos, incluso los numéricos, siempre se encierra entre comillas.

- En los atributos en los que el nombre coincide con su valor, no puede darse el valor por entendido, es decir, no se pueden comprimir. Este tipo de atributos no son muy habituales.
- Todas las etiquetas deben cerrarse siempre. **XHTML** permite que en lugar de abrir y cerrar de forma consecutiva la etiqueta (`

`) se puede utilizar la sintaxis `
` para indicar que es una etiqueta vacía que se abre y se cierra en ese mismo punto.

En general, los diseñadores web suelen trabajar con HTML. El XHTML es más apreciado por los desarrolladores, que aprecian la regularidad adicional. De cualquier manera, los tres primeros puntos de la anterior lista se consideran una buena práctica y se suelen cumplir siempre.

Por otro lado, hay que tener en cuenta que los navegadores no procesan HTML y XHTML exactamente igual. En concreto, en caso de errores para HTML el navegador intentará mostrar el mayor contenido posible. Pero si el documento se sirve como XHTML (con un tipo MIME XML), cualquier error de sintaxis causa que no se muestre el documento.

Para más información sobre HTML y XHTML, consulta la sección correspondiente en la especificación (inglés).

3 Estructura de un documento HTML

La estructura de una página HTML ha de ser coherente con la que hemos visto en el tema anterior para cualquier documento XML. Por ello tendrá un prólogo y un ejemplar.

- **Prólogo:** Todo documento HTML ha de tener una declaración del tipo de documento donde se le indica al navegador el tipo de documento que se va a iniciar y la versión de HTML utilizada para la codificación del mismo y, además, le permite interpretarlo correctamente.

Para la versión HTML 4.0, hay tres prólogos distintos que definen tres tipos de documentos HTML:

- *HTML 4.0 Strict.* Es la DTD utilizada por defecto con HTML 4.0. En estos documentos no se permite el uso de los elementos declarados deprecated en otras versiones o Recomendaciones HTML. La declaración del tipo de documento correspondiente es:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

- *HTML 4.0 Transitional.* Permite el uso de todos los elementos que permite el HTML 4.0 Strict, además de los elementos deprecated. La declaración del tipo de documento correspondiente es:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
```

- *HTML 4.0 Frameset.* Es una variante de HTML 4.0 Transitional para documentos que usan frames. En estos documentos el elemento body hay que reemplazarlo por un elemento frameset. La declaración del tipo de documento correspondiente es:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">
```

En el caso de HTML5 solo existe una declaración de tipo de documento:

```
<!DOCTYPE html>
```

Ejemplar: En un documento HTML está delimitado por las etiquetas `<html>` y `</html>`. El ejemplar puede, a su vez dividirse en dos partes:

- *La cabecera*, delimitada por las etiquetas `<head>` y `</head>`. Contiene la información sobre el título de la página, el autor, palabras clave, etc. Dentro de esta sección es obligatorio definir el título del documento, para ello se usan las etiquetas `<title>` `</title>`. Esta información no se presentará en la ventana del navegador, salvo el título, que aparecerá en la barra de título de la parte superior.
- *El cuerpo*, contiene la información que se va a presentar en la pantalla. Está limitado por las etiquetas `<body>` y `</body>`, salvo en los documentos de tipo HTML 4.0 Frameset donde éstas se sustituyen por `<frameset>` y `</frameset>`.

4 Identificación de etiquetas y atributos HTML

Un documento **HTML** está formado por etiquetas y atributos.

Al igual que en **XML** las etiquetas pueden ser de apertura, <etiqueta>, o de cierre, </etiqueta>. Una de las diferencias con **XML** es que la cantidad de etiquetas de **HTML** está limitada a aquellas que están definidas por el lenguaje.

Aunque **HTML** define una gran cantidad de etiquetas, estas no son suficientes para crear páginas complejas ya que la definición completa de ciertos elementos, como las imágenes y los enlaces, requiere información adicional. Como no es posible crear una etiqueta por cada elemento diferente, se añade la información adicional a las etiquetas mediante los atributos, dando lugar a los elementos.

Para cada uno de los atributos hay definido un conjunto de valores que se le puede asignar. Si el valor de un atributo no es válido, el navegador lo ignora.

Cada una de las etiquetas **HTML** define los atributos que puede utilizar, aunque algunos de ellos son comunes a muchas etiquetas.

5 Clasificación de los atributos comunes según su funcionalidad

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

Atributo	Descripción
name = "texto"	Permite asignar el nombre "texto" a un objeto HTML
title = "texto"	Asigna un título a un elemento HTML, mejorando así la accesibilidad. Dicho título es mostrado por los navegadores cuando el usuario pasa el ratón por encima del elemento. Es especialmente útil con los elementos: a, link, img, object, abbr y acronym
id = "texto"	Permite identificar al elemento HTML sobre el que se aplica de forma única mediante el identificador "texto". Sólo es útil cuando se trabaja con CSS y con Javascript. No pueden empezar por números y sólo puede contener letras, números, guiones medios y/o guiones bajos.
style = "texto"	Permite aplicar al elemento HTML el estilo "texto" directamente.
class = "texto"	Permite aplicar al elemento HTML el estilo "texto" definido en las CSS. No pueden empezar por números y sólo puede contener letras, números, guiones medios y/o guiones bajos.

- **Atributos para internacionalización:** Los utilizan las páginas que muestran sus contenidos en varios idiomas o aquellas que quieren indicar de forma explícita el idioma de sus contenidos

Atributo	Descripción																
dir	Indica la dirección del texto por lo que sólo puede tomar dos valores: ltr (left to right) de izquierda a derecha. Es el valor por defecto. rtl (right to left) de derecha a izquierda.																
lang = "codigo"	Especifica el idioma del elemento mediante un código predefinido. Los posibles valores de este atributo se encuentran en el documento RFC 1766 , algunos de los valores posibles son: <table border="1"><thead><tr><th>Código</th><th>Idioma</th><th>Código</th><th>Idioma</th></tr></thead><tbody><tr><td>en</td><td>Inglés (Gran Bretaña)</td><td>es</td><td>Español</td></tr><tr><td>en-US</td><td>Inglés americano</td><td>fr</td><td>Francés</td></tr><tr><td>ja</td><td>Japones</td><td>fr-CA</td><td>Francés de Canadá</td></tr></tbody></table>	Código	Idioma	Código	Idioma	en	Inglés (Gran Bretaña)	es	Español	en-US	Inglés americano	fr	Francés	ja	Japones	fr-CA	Francés de Canadá
Código	Idioma	Código	Idioma														
en	Inglés (Gran Bretaña)	es	Español														
en-US	Inglés americano	fr	Francés														
ja	Japones	fr-CA	Francés de Canadá														
xml:lang = "codigo"	Especifica el idioma del elemento mediante un código definido según la recomendación RFC 1766 .																

En las páginas **XHTML**, el atributo **xml:lang** tiene más prioridad que **lang** y es obligatorio incluirlo siempre que se incluye el atributo **lang**.

- **Atributos de eventos y atributos para los elementos que pueden obtener el foco:** Sólo se utilizan en las páginas web dinámicas creadas con JavaScript. Como no es nuestro objetivo no lo vamos a contemplar.

2.B. Elementos HTML

Sitio: Aula Virtual CIERD (CIDEAD)
Curso: Lenguaje de marcas y sistemas de gestión de información
Libro: 2.B. Elementos HTML
Imprimido por: MANUEL JOSE GONZALEZ CALVO
Día: jueves, 21 de mayo de 2020, 23:04

Tabla de contenidos

- 1 Introducción - Elementos HTML
- 2 Elementos de la estructura básica de un documento
- 3 Elementos de la sección de cabecera
- 4 Encabezados y párrafos
 - 4.1 Saltos de línea y espacios en blanco
 - 4.2 Comentarios
- 5 Semántica a nivel de texto
- 6 Elementos de listas
- 7 Elementos de tablas
- 8 Elementos de formularios
 - 8.1 Declaración de formularios - Etiqueta form
 - 8.2 Campos de formulario - input
 - 8.3 Campos de formulario - Área de texto
 - 8.4 Campos de formularios - Lista desplegable
 - 8.5 Campos de formulario - checkbox
 - 8.6 Botones de formulario
 - 8.7 Otros campos de formulario
 - 8.8 Campos de formulario con formato de fecha
 - 8.9 Campos de formulario - Rangos
 - 8.10 Organización de formularios
 - 8.11 Un ejemplo de formulario
- 9 Multimedia
- 10 Secciones y etiquetas semánticas
- 11 Elemento iframe
- 12 Validación HTML

1 Introducción - Elementos HTML

Un elemento HTML está formado por:

- Una etiqueta de apertura.
- Cero o más atributos.
- Opcionalmente, un texto, encerrado por la etiqueta. No todas las etiquetas pueden encerrar texto.
- Una etiqueta de cierre. Para algunos elementos no hay etiqueta de cierre o es opcional.

Según el modo en que ocupan el espacio disponible en la página, los elementos pueden ser de dos tipos:

- **Elementos en línea.** Sólo ocupan el espacio necesario para mostrar sus contenidos. Su contenido puede ser texto u otros elementos en línea.
- **Elementos de bloque.** Los elementos 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 hasta allí. Su contenido puede ser texto, elementos en línea u otros elementos de bloque.

El siguiente ejemplo muestra la diferencia entre ambos comportamientos

```
<!DOCTYPE html>

<html>
  <head>
    <title>Ejemplo de la diferencia entre los elementos en línea
      y los elementos de bloque
    </title>
  </head>
  <body>
    <h1>Los encabezados son elementos de bloque.</h1>
    <p>Y los párrafos también.</p>
    <a href="#">Los enlaces son elementos de línea</a>
    <p>Incluso si esta definido dentro de un párrafo, <strong>un texto resaltado</strong> sigue siendo un
elemento en línea.
    </p>
  </body>
</html>
```

Al publicarlo en un navegador quedaría así:

Los encabezados son elementos de bloque.

Y los párrafos también.

[Los enlaces son elementos de línea](#)

Incluso si esta definido dentro de un párrafo, **un texto resaltado** sigue siendo un elemento en línea.

2 Elementos de la estructura básica de un documento

Los elementos que definen la estructura básica de un documento son:

<html> - Define el inicio (y final con "</html>") de un documento html. Contiene a los elementos:

<head> - Define el inicio (y final con "</head>") de la cabecera del documento.

<body> - Define el inicio (y final con "</body>") del cuerpo del documento. Permite definir formatos que se aplican a los elementos de la página de manera global, como por ejemplo el color de fondo del texto o la anchura de los márgenes.

Un ejemplo de un documento HTML básico que utiliza estos elementos es:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo de la estructura de un documento HTML</title>
  </head>
  <body>
    Aquí es donde se coloca la información que se quiere visualizar en el navegador.
  </body>
</html>
```



Al publicarlo en un navegador, por ejemplo en el Firefox, tendríamos:



3 Elementos de la sección de cabecera

Se clasifican en dos tipos:

- **Elementos contenedores:**

Elemento	Descripción
title	Título del documento. Elemento obligatorio.
script	Script incrustado. Su contenido ha de ir situado entre las marcas de comentarios ya que no ha de ser interpretado.
style	Estilo aplicado al documento utilizando CSS. Su contenido ha de ir situado entre las marcas de comentarios ya que no ha de ser interpretado.

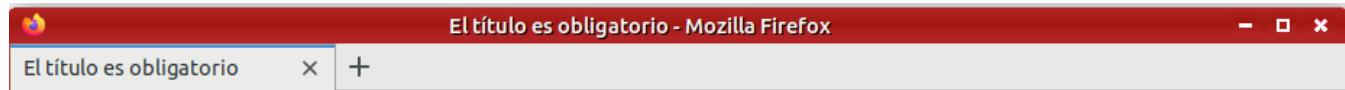
- **Elementos no contenedores:**

Elemento	Descripción
base	URI base del documento
isindex	Prompt de entrada de datos. (Eliminado de los estándares web aunque todavía algún navegador lo soporta)
link	Enlaces a documentos externos de librerías
meta	Metadatos sobre la página, como la codificación de caracteres, descripción o autores.

Ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>El título es obligatorio</title>
  </head>
  <body>
    ...
  </body>
</html>
```

El título aparece en la barra del navegador y como nombre de la pestaña. El ejemplo anterior se vería así:



Para que los acentos se vean bien hay que guardar el ejemplo con la codificación indicada en el documento, UTF-8.

4 Encabezados y párrafos

Para agrupar el texto en párrafos se usa el elemento `<p>`. Es un elemento de bloque.

`<p>Texto del párrafo </p>`

Para los encabezados, en HTML se definen 6 elementos:

`<h1>,<h2>,<h3>, <h4>, <h5>, <h6>`

Cuanto menor es el número, mayor es la importancia del encabezado. El texto marcado debe servir como encabezado a la sección en la que aparece. Se pueden utilizar para organizar jerárquicamente el contenido de la página.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Párrafos y encabezados</title>
  </head>
  <body>
    <h1>Equipos</h1>
    <p>Primer párrafo</p>
    <p>Segundo párrafo</p>
    <h2>Recién ascendidos</h2>
    <p>...</p>
    <h1>Jugadores</h1>
    <p>...</p>
  </body>
</html>
```

Se vería así:

Equipos

Primer párrafo

Segundo párrafo

Recién ascendidos

...

Jugadores

...

Más ejemplos

4.1 Saltos de línea y espacios en blanco

Al procesar el HTML los navegadores ignoran los saltos de línea y, si encuentran varios espacios consecutivos, los colapsan en uno.

Si es necesario introducir un salto de línea se utiliza el elemento
. Para introducir múltiples espacios, se puede utilizar la entidad , aunque también hay otras opciones con etiquetas HTML o propiedades CSS.

Ejemplo

Esta página

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='UTF-8'>
    <title>Saltos de línea</title>
  </head>
  <body>
    Esto
    se
    ve
    en
    una      línea sin espacios de más
    <p>Aquí hay &nbsp;&nbsp;&nbsp;&nbsp;espacios</p>
    Esto <br> introduce un salto de línea
  </body>
</html>
```

se verá así:

Esto se ve en una línea sin espacios de más

Aquí hay espacios

Esto
introduce un salto de línea

4.2 Comentarios

Se pueden introducir comentario en los ficheros HTML así:

```
<!-- comentario -->
```

También pueden ser multilínea:

```
<!-- comentario  
de varias  
líneas -->
```

Los comentarios no son procesados por los navegadores, sirven para documentar el código.

5 Semántica a nivel de texto

Algunos elementos útiles son:

- <a>, para definir hipervínculos.
- , para representar que el texto marcado es importante.
- , para indicar énfasis.
-
, introduce un salto de línea.
- <small>, para comentarios accesorios, como lo que suelen aparecer en letra pequeña.

Elemento <a>

Este elemento hace que el texto encerrado entre las etiquetas sea un hipervínculo.

El atributo más importante es **href**, que indica la URL del vínculo, A veces se usa **href="#"** para referirse a la misma página en la que aparece el vínculo.

El atributo **target** permite elegir donde se abrirá el vínculo. Los valores más usados son:

- **target = "_blank"**, para que se abrá en una ventana/pestaña nueva.
- **target = "_self"**, para que se abrá en la misma. Es el valor por defecto.

Ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Semántica a nivel de texto</title>
  </head>
  <body>
    Texto marcado como <strong>importante</strong>.
    <br>
    Texto con <em>énfasis</em>
    <br>
    Texto marcado <small> con el elemento small </small>
    <br>
    Pulsa <a href="https://www.w3.org/">aquí</a> para ir a la página del W3C.
  </body>
</html>
```

El resultado sería:

Texto marcado como **importante**.
Texto con **énfasis**
Texto marcado con el elemento **small**
Pulsa [aquí](https://www.w3.org/) para ir a la página del W3C.

Aquí se puede consultar la lista completa de elementos para esta categoría. No todos tienen una representación visual en el navegador.

6 Elementos de listas

Hay tres tipos de listas: ordenadas, desordenadas y listas de definición.

Elemento	Descripción
ul	Delimita los elementos que forman una lista desordenada
ol	Delimita los elementos que forman una lista ordenada. Tiene varios atributos.
dl	Delimita los elementos que forman una lista de definición
li	Cada uno de los elementos de una lista ordenada o desordenada.
dt	Cada uno de los términos que se definen de una lista de definición.
dd	Cada una de las definiciones de una lista de definición.

Un ejemplo de un documento HTML que muestra la forma de utilizar estos elementos es:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Listas</title>
  </head>
  <body>
    <h1>Ejemplo de lista desordenada: Módulos de 1º de ASIR</h1>
    <ul>
      <li>Fundamentos de Hardware</li>
      <li>Gestión de Bases de Datos</li>
    </ul>
    <h1>Ejemplos de listas ordenadas: Módulos de 1º de ASIR</h1>
    Comenzando en 1
    <ol>
      <li>Fundamentos de Hardware</li>
      <li>Gestión de Bases de Datos</li>
    </ol>
    Comenzando en 4
    <ol start = "4">
      <li>Fundamentos de Hardware</li>
      <li>Gestión de Bases de Datos</li>
    </ol>
    <h1>Ejemplo de lista de definición: Módulos de 1º de ASIR</h1>
    <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>
```

Que al publicarse en el navegador quedaría:

Ejemplo de lista desordenada: Módulos de 1º de ASIR

- Fundamentos de Hardware
- Gestión de Bases de Datos

Ejemplos de listas ordenadas: Módulos de 1º de ASIR

Comenzando en 1

1. Fundamentos de Hardware
2. Gestión de Bases de Datos

Comenzando en 4

4. Fundamentos de Hardware
5. Gestión de Bases de Datos

Ejemplo de lista de definición: Módulos de 1º de ASIR

Fundamentos de Hardware

 Componentes físicos de un ordenador

Gestión de Bases de Datos

 Diseño y uso de bases de datos relacionales

7 Elementos de tablas

Las tablas son una forma habitual de presentar información de manera compacta y fácil de entender para el que la lee. Aunque hay muchas posibilidades, en general las tablas muestran una serie de datos (columnas) comunes para varios elementos (filas). Como ejemplos, podemos pensar en un horario, o en una tabla comparativa de varios productos.

Los elementos para definir una tabla son los siguientes (no es necesario usar todos):

Elemento	Descripción
table	Delimita el contenido de una tabla.
tr	Delimita cada una de las líneas de la tabla.
td	Delimita el contenido de cada celda de la tabla.
colgroup	Permite agrupar columnas.
tbody	Permite agrupar líneas de la tabla.
thead	Define la línea cabecera de la tabla.
th	Delimita cada una de las celdas de la cabecera
tfoot	Define la fila pie de la tabla.
caption	Para añadir una leyenda a la tabla

En HTML las tablas están formadas por filas (elemento **<tr>**), y estas a su vez por celdas. Las celdas pueden ser de datos (elemento **<td>**) o de cabecera (elemento **<th>**).

- El elemento **<table>** contiene al resto de elementos.
- Por cada fila de la tabla habrá un elemento **<tr>**. Este elemento contiene una serie de elementos **<td>** o **<th>** (uno por columna).
- Normalmente en la primera fila las celdas son elementos **<th>** y en el resto **<td>**.

Ejemplo de tabla básica

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Tablas</title>
  </head>
  <body>
    <table>
      <caption>Tabla de socios</caption>
      <tr>
        <th>Nombre</th>
        <th>Apellido</th>
        <th>Edad</th>
      </tr>
      <tr>
        <td>Juan</td>
        <td>Puertas</td>
        <td>54</td>
      </tr>
      <tr>
        <td>Eva</td>
        <td>Montes</td>
        <td>44</td>
      </tr>
    </table>
  </body>
</html>
```

En el navegador se verá así:

Tabla de socios

Nombre	Apellido	Edad
Juan	Puertas	54
Eva	Montes	44

En este tutorial puedes ver ejemplos de tablas avanzadas.

NOTA IMPORTANTE: Aunque era habitual hace años, no es correcto usar tablas para la maquetación de la página.

8 Elementos de formularios

Los formularios permiten recoger información que el usuario introduce en el navegador.

Es importante validar estos datos, para detectar los errores en local (en el propio equipo), en caso de haberlos. De este modo se evita sobrecargar la red con datos erróneos y sobrecargar al servidor con tareas innecesarias.

Normalmente se combinan los formularios con código **Javascript**, que ayuda a realizar esas validaciones.

En la siguiente imagen puedes ver un ejemplo de un formulario con una muestra de algunos de los diferentes tipos de campos existentes.

The screenshot shows a web browser window titled "Ejemplo de Formulario". The address bar indicates the page is at "file:///D:/OneDrive/OneDrive - IES Europa/Material/Introducción a la programación con JavaScript/08. Formularios/01. Ejemplos de formularios/index.html". The main content area displays a user registration form:

- Section title:** Bienvenido al registro de usuarios de nuestro foro
- Text input:** Introduzca su nombre de usuario y contraseña
- Text input:** Usuario (empty)
- Text input:** Contraseña (empty)
- Text input:** Repita Contraseña (empty)
- Text input:** Describa a continuación sus intereses: (empty text area)
- Select dropdown:** Por último seleccione el sistema operativo de su ordenador:
A dropdown menu is open, showing options: Linux (selected), Windows, and Mac OS. A cursor arrow points to the "Linux" option.
- Checkboxes and button:** Estoy conforme con la política de privacidad de la página **Registrarme** (button)

Puedes ver el código que genera el formulario de la imagen en este **enlace** (Recuerda que, una vez se ha cargado una página en tu navegador, puedes visualizar el código que la generó. Basta con que pulses con el botón derecho sobre el fondo de la página y seleccionas "Ver código fuente de la página").

8.1 Declaración de formularios - Etiqueta form

La apertura y cierre de un formulario se hace mediante el elemento **form**.

La etiqueta permite especificar una serie de atributos para ajustar sus características. Los principales son:

- **name** – nombre del formulario
- **action** – Acción que se ejecuta cuando se pulsa el botón de enviar formulario
- **enctype** – Formato en el que se enviarán los valores del formulario, depende del contenido se enviarán de una u otra manera.
- **method** – Método de envío de la transmisión de datos, se puede elegir entre dos métodos GET y POST

```
<form name="miFormulario" action="/paginaQueEjecutaLaAccion.php" method="get">
```

La diferencia entre los métodos de envío Get y Post es la siguiente:

GET vs POST

GET	POST
<ul style="list-style-type: none">• Permite pasar valores en ASCII con un límite de 100 caracteres• Los valores de las variables que se transmiten se pueden ver en la URL y van concatenadas por el símbolo de AND (&) por ejemplo: http://www.pagina.com/index.php?variable1=valor1&variable2=valor2&variable3=valor3	<ul style="list-style-type: none">• Permite pasar valores de variables y otros elementos tales como archivos• Carece de restricciones de longitud como el método GET• Las variables y sus valores no son visibles en la URL

8.2 Campos de formulario - input

Dentro de un formulario puede haber varios tipos de controles: campos de texto normales, campos para contraseñas, fechas o botones de radio, entre otros.

El elemento **<input>** (que no tiene etiqueta de cierre) se puede usar para varios tipos de control, según el valor que tome el atributo **type**. Por ejemplo, para **type="text"**, que es el valor por defecto, se obtiene un campo de texto. Con **type = "radio"**, un botón de radio.

Además, de **type**, los atributos más importantes de la etiqueta son:

- name** – nombre del campo
- size** – número de caracteres visibles en el campo. Por defecto 20.
- maxlength** – número máximo de caracteres que el usuario podrá introducir en el campo
- value** – valor por defecto del campo de texto
- placeholder** – valor sugerido, se presenta en color gris y desaparece al hacer foco con el cursor en el campo
- readonly** – El valor del campo no puede ser modificado por el usuario
- autofocus** - Sitúa el cursor del ratón en el campo una vez cargada la página
- required** - No se podrá enviar el formulario hasta que el campo esté cumplimentado

Ejemplo:

```
<input type="text" name="usuario" size="30" maxlength="20" placeholder="Escriba aquí el nombre de usuario" required>
```

Se vería en el navegador así:

Usuario

8.3 Campos de formulario - Área de texto

Permite recoger información abierta del usuario pero permitiendo un mayor número de caracteres. Se especifica mediante la etiqueta <textarea>

Para especificar un valor previo en el campo debe escribirse entre las etiquetas <textarea> de apertura y cierre

Atributos principales

- **name** - nombre del campo
- **rows** – número de filas. Reemplazable por la propiedad CSS height
- **cols** – número de columnas. Reemplazable por la propiedad CSS width

También puede utilizar algunos de los atributos especificados para input:

- **placeholder**
- **readonly**
- **autofocus**
- **maxlength**
- **required**

Por ejemplo, la siguiente línea de código HTML:

```
Describa a continuación sus intereses: <br> <textarea name="area"></textarea>
```

Quedaría en el navegador del siguiente modo:

Describa a continuación sus intereses:

A large, empty rectangular text area box. Above it, the text "Describa a continuación sus intereses:" is displayed in a smaller font. In the bottom right corner of the text area box, there is a small icon consisting of three dots arranged in a triangular pattern.

8.4 Campos de formularios - Lista desplegable

Permite que el usuario seleccione un valor entre diferentes opciones.

Etiquetas:

- <select> para definir la lista
- <option> para cada una de las opciones

Atributos - <select>

name - Nombre del elemento

size – Número de elementos de la lista desplegable que se mostrarán

multiple – Permite seleccionar varias opciones manteniendo pulsada la tecla **Ctrl** al seleccionarla

Atributos - <option>

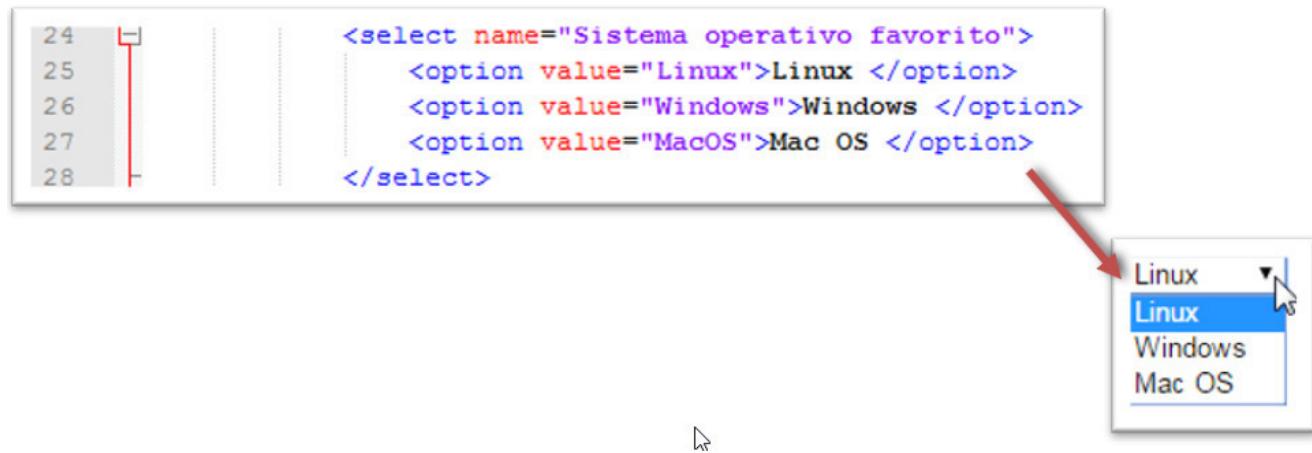
selected – campo que aparece seleccionado por defecto

value – valor que se transmite cuando se envía el formulario

A continuación puedes encontrar un ejemplo de código HTML:

```
Por último seleccione el sistema operativo de su ordenador:  
  
<select name="Sistema operativo favorito">  
    <option value="Linux">Linux </option>  
    <option value="Windows">Windows </option>  
    <option value="MacOS">Mac OS </option>  
</select>
```

En la siguiente imagen puedes observar cómo quedaría en un editor de HTML (en este caso Notepad++) y cuál es el resultado en el navegador.



8.5 Campos de formulario - checkbox

Permite que el usuario indique si está conforme con una o varias opciones.

Etiqueta: <input type="checkbox">

Atributos

- o **name** – obligatorio
- o **checked** – el atributo aparece marcado
- o **value** – valor que se transmitirá al enviar el formulario con ese campo seleccionado

Veamos un ejemplo de código HTML:

```
Estoy conforme con la política de privacidad de la página
```

```
<input type="checkbox" name="conforme" checked>
```

En el navegador se vería así:

Estoy conforme con la política de privacidad de la página

8.6 Botones de formulario

7.6.1. Botón de envío

Permite enviar el formulario a un destinatario, por ejemplo a una página determinada en un servidor web para que procese los datos o una dirección de correo electrónico. Todo dependiendo del valor del atributo **action** de la etiqueta de apertura de formulario.

Etiqueta: <**input type="submit"**>

Atributos

value – permite indicar el texto del botón

7.6.2. Botón de anulación

Borra el contenido de los campos cumplimentados en el formulario

Etiqueta: <**input type="reset"**>

Atributos

value – permite indicar el texto del botón



8.7 Otros campos de formulario

Dependiendo de los valores que queramos recoger podemos utilizar definiciones de campos más apropiadas como las que se ven en la siguiente tabla.

Campo	Etiqueta	Notas
Oculto	<input type="hidden">	
De contraseña	<input type="password">	
Para envío de ficheros	<input type="file">	Requiere formulario definido con post y enctype="multipart/form-data"
Para recoger correo electrónico	<input type="email">	Valida formato automáticamente
Para recoger una URL	<input type="url">	Valida formato automáticamente
Para recoger números enteros en un rango	<input type="number">	Atributos: max – valor máximo min – valor mínimo step – incremento del contador

8.8 Campos de formulario con formato de fecha

Dependiendo de la fecha que queramos recoger y sus características podemos utilizar diferentes variaciones en las etiquetas.

<input type="datetime"> o <input type="datetime-local">

Permite seleccionar día, mes, año y hora

<input type="date">

Permite seleccionar día, mes y año

<input type="month">

Permite seleccionar mes y año

<input type="week">

Permite seleccionar una semana

<input type="time">

Permite seleccionar una hora

Atributos

- **min** – Menor hora seleccionable
- **max** – Mayor hora seleccionable
- **step** – Incremento del contador del campo en segundos

8.9 Campos de formulario - Rangos

Permite seleccionar un valor dentro de un rango.

Etiqueta: <input type="range">

Atributos

- **min** – valor mínimo del rango
- **max** – valor máximo del rango
- **step** – valor del incremento del contador
- **value** – valor inicial del contador

8.10 Organización de formularios

Etiqueta <label>

Permite asociar a cada campo del formulario una etiqueta con su nombre. El texto mostrado entre las etiquetas <label> se muestra y constituye además una ayuda de usabilidad a personas invidentes.

```
<label for="conforme">Acepto el acuerdo de licencia</label>
<input type="checkbox" name="licencia" id="conforme" value="ok">
```

En el ejemplo la etiqueta **label** se asocia al campo tipo **input** mediante el atributo **for** que contiene el valor del identificador (**id**) de la etiqueta **input**. En el navegador se vería:

Acepto el acuerdo de licencia



Etiqueta <fieldset>

Agrupa los campos de formulario que estén entre la etiqueta de apertura y cierre <fieldset> y los rodea con un borde.

Etiqueta <legend>

Escrita inmediatamente a continuación de la etiqueta de apertura <fieldset> agrega un texto relacionado con los campos agrupados

8.11 Un ejemplo de formulario

En este enlace puedes ver un posible código que genera un formulario como el que aparece en la imagen inferior:

Formulario +

https://fpdistancia.educa.madrid.org/pluginfile.php/95215/mod_scorm/content/1/LM

Ejemplo de formulario: Matriculación en el ASIR

Nombre: Apellidos: Sexo: M H

Fecha de nacimiento: dia mes año

Escoge los módulos en los que te matriculas:

<input type="checkbox"/> Fundamentos de Hardware	<input type="checkbox"/> Servicios de Red e Internet
<input type="checkbox"/> Gestión de Bases de Datos	<input type="checkbox"/> Administración de sistemas Operativos
<input type="checkbox"/> Implantación de Sistemas Operativos	<input type="checkbox"/> Implantación de Aplicaciones Web
<input type="checkbox"/> Planificación y Administración de Redes	<input type="checkbox"/> Administración de Sistemas Gestores de Bases de Datos
<input type="checkbox"/> Lenguajes de Marcas y Sistemas de Gestión de Información	<input type="checkbox"/> Seguridad y Alta Disponibilidad
<input type="checkbox"/> Formación y Orientación Laboral	<input type="checkbox"/> Empresa e Iniciativa Emprendedora
<input type="checkbox"/> Proyecto de Administración de Sistemas Informáticos en Red	<input type="checkbox"/> Formación en Centros de Trabajo

Estudios previos:

Pulsa el botón de "enviar" para formalizar la matrícula o el botón de "restablecer" para limpiar el formulario.

Sin embargo verás que dicho código no sigue las reglas de organización de formularios que hemos indicado. ¿Podrías modificarlo? Si es así, entrégalo en **esta actividad voluntaria**.

9 Multimedia

Otros elementos habituales son:

Elemento	Descripción
img	Permite insertar una imagen en una página web.
audio	Para insertar audio
video	Para insertar vídeo
object	Para incrustar contenido multimedia (audio, vídeo, ficheros PDF, applets...)

Elemento

Este elemento se usa para insertar una imagen. No tiene etiqueta de cierre. Los atributos más importante son:

- src. Especifica la ruta de la imagen. Puede ser local, o una URL cualquiera. Los formatos soportados dependen de cada navegador, pero en general se pueden utilizar JPG, PNG y GIF, entre otros.
- alt. Texto alternativo. Se usa si no se puede cargar la imagen o para los lectores de pantalla. Es obligatorio.
- height. Altura. Si no se especifica, se escoge el tamaño original. Si se especifica, se escala. Es de los pocos casos en HTML5 en que un atributo tiene información de representación.
- width. Anchura de la imagen. Como la anterior.

Ejemplo

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='UTF-8'>
    <title>Imágenes</title>
  </head>
  <body>
    <p>Imagen en internet </p>
    <img src='https://i.blogs.es/aa1b9a/luna-100mpx/450_1000.jpg' alt='la luna'>
    <p>Imagen no encontrada</p>
    <img src='http://rutaincorrecta.jpg' alt='la luna'>
    <p>Imagen local</p>
    <img src='index.jpeg' alt='logo'>
  </body>
</html>
```

Se vería así:

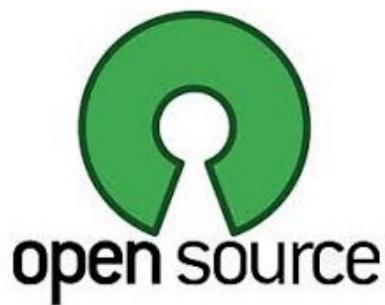
Imagen en internet



Imagen no encontrada

la luna

Imagen local



En el segundo caso, como no encuentra la imagen muestra el texto alternativo.

Para poder probar este ejemplo, descarga este fichero y guárdalo en el mismo directorio que la página con nombre "index.jpeg".

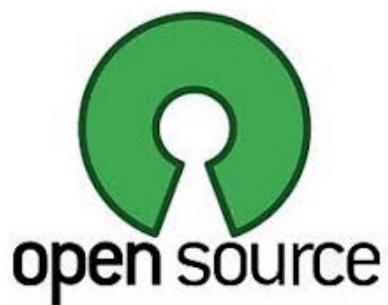
Imagen en internet



Imagen no encontrada

vistas al mar

Imagen local



10 Secciones y etiquetas semánticas

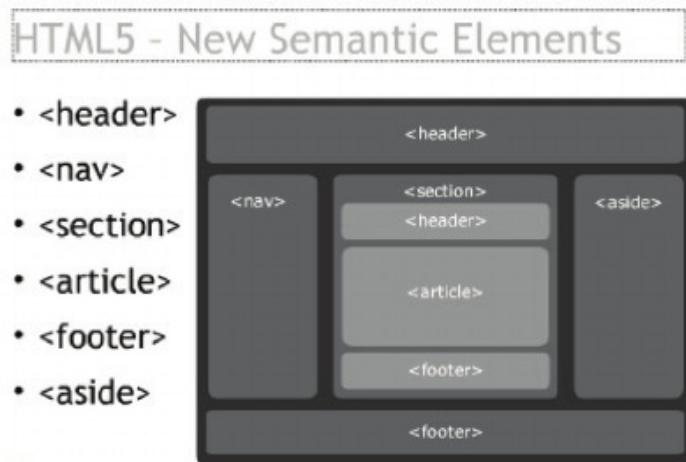
El elemento **<div>** se usa para agrupar otros elementos en secciones, tanto para organizar el contenido como para posicionarlo mediante hojas de estilo.

En HTML5 aparecieron varias etiquetas semánticas para estructurar el contenido de la página, y por tanto solo se debería usar **<div>** cuando no haya una etiqueta más apropiada

Estas etiquetas son:

- **<header>**. Contiene contenido introductorio para la sección de la página en que aparece. Es habitual que contenga los elementos de encabezado, *h1,...,h6*.
- **<aside>**. Se utiliza para contenido parcialmente relacionado con el contenido principal. No tiene por qué mostrarse en un lateral.
- **<footer>**. Contiene información sobre la sección correspondiente, como el autor. No tiene que estar necesariamente en la parte de inferior.
- **<section>**. Una sección genérica dentro del documento.
- **<article>**. Representa un elemento que se puede distribuir de manera independiente o reutilizable.
- **<nav>**. Contiene vínculos, internos o externos. Suele usarse para la barra de navegación.

La siguiente imagen muestra una disposición habitual para estos elementos. Para obtenerla hay que usar CSS.



Lectura recomendada: HTML semántico

11 Elemento iframe

El elemento iframe permite integrar una página web dentro de otra. En ciertos aspectos, sustituye a la especificación de marcos (frameset) de HTML 4, ya obsoleta.

Atributos más importantes de iframe:

- height: Altura, por defecto en píxeles. Si no cabe todo el contenido, se usa una barra de desplazamiento. También se puede fijar con CSS.
- width: Anchura, como el anterior.
- src: La ruta al contenido inicial del iframe.
- name: Nombre, para referirse al iframe desde un vínculo.

A continuación, vamos a ver un ejemplo de uso habitual.

[Ejemplo de listas](#) [Ejemplo de tablas](#)

Aquí está el iframe



El contenido del iframe cambia al seguir los vínculos. Para conseguirlo, en los vínculos hay que poner como valor del atributo target el valor del atributo name del iframe en que queramos que se cargue.

Es decir, para que se carguen en un iframe con

name = "contenido"

se usará

...

El código del ejemplo anterior sería el siguiente:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>iframe</title>
  </head>
  <body>
    <a href = "listas.html" target = "contenido">Ejemplo de listas</a>
    <a href = "tablas.html" target = "contenido">Ejemplo de tablas</a>
    <p>Aquí está el iframe</p>
    <iframe height = "300" width = "400" name = "contenido" src = "listas.html">
  </body>
</html>
```

Inicialmente, el iframe carga "listas.html". Luego cambia entre "listas.html" y "tablas.html" al seguir los vínculos. Estos ficheros son los que se han utilizado en los ejemplos anteriores.

Para probar el ejemplo, tienes que tener los tres ficheros en el mismo directorio y abrir "iframe.html":

Ficheros

12 Validación HTML

Es posible validar si nuestro código **HTML** cumple con la especificación de **HTML5**, para ello existen diferentes páginas en Internet que nos facilitan este servicio:

- **W3C** - <https://validator.w3.org/>
- **Validator.nu** - <https://html5.validator.nu/>

2.C. Hojas de estilo (CSS)

Sitio: Aula Virtual CIERD (CIDEAD)
Curso: Lenguaje de marcas y sistemas de gestión de información
Libro: 2.C. Hojas de estilo (CSS)
Imprimido por: MANUEL JOSE GONZALEZ CALVO
Día: jueves, 21 de mayo de 2020, 23:10

Tabla de contenidos

- 1 Introducción
- 2 Cómo incluir CSS en un documento HTML o XHTML
 - 2.1 Elemento span
- 3 Sintaxis de las reglas de estilo
- 4 Cascada y herencia de estilos
- 5 Selectores
 - 5.1 Selectores de clase
 - 5.2 Selectores de ID
 - 5.3 Selectores descendentes
 - 5.4 Selector hijo
- 6 Propiedades principales
 - 6.1 Propiedades de color y fondo
 - 6.2 Propiedades de fuente
 - 6.3 Propiedades de texto
 - 6.4 Propiedades de listas
 - 6.5 Propiedad display
- 7 Avanzado: Modelo de cajas
 - 7.1 Propiedades de caja
 - 7.2 Unidades de tamaño
 - 7.3 Ejemplos del modelo de cajas
 - 7.4 Posicionamiento
- 8 Validador CSS

1 Introducción

CSS (Cascading Style Sheets) permite a los desarrolladores Web controlar el estilo y el formato de múltiples páginas Web al mismo tiempo.

Antes del uso de **CSS**, los diseñadores de páginas web debían definir el aspecto de cada elemento dentro de las etiquetas **HTML** de la página. El principal problema de esta forma de definir el aspecto de los elementos es que habría que definir el formato de cada uno de los elementos que formen la página, lo cual hace que sea muy difícil de actualizar.

CSS permite separar los contenidos de la página y su aspecto. Para ello se define en una zona reservada el formato de cada uno de los elementos de la web. Cualquier cambio en el estilo marcado para un elemento en la **CSS** afectará a todas las páginas vinculadas a ella en las que aparezca ese elemento. Las hojas de estilo están compuestas por una o más reglas de estilo aplicadas a un documento **HTML** o **XML**.

Al crear una página web, se utiliza en primer lugar el lenguaje **HTML/XHTML** para marcar los contenidos, es decir, para designar la función de cada elemento dentro de la página: párrafo, cabecera, texto destacado, etc. Una vez creados los contenidos, se utiliza el lenguaje **CSS** para definir el formato de cada elemento.

CSS obliga a crear documentos semánticos **HTML/XHTML**, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes.



Las hojas de estilos aparecieron poco después que el lenguaje de etiquetas **SGML**, alrededor del año 1970. Desde la creación de **SGML**, se observó la necesidad de definir un mecanismo que permitiera aplicar estilos a los documentos electrónicos. La guerra de navegadores y la falta de un estándar para la definición de los estilos dificultaban la creación de documentos que tuvieran igual apariencia en distintos navegadores.

En 1995, el **W3C** añadió a su grupo de trabajo de **HTML** el desarrollo y estandarización de **CSS**.

CSS 1, se publicó en 1996, es la primera recomendación oficial.

CSS 2, publicada en 1998, es la segunda recomendación oficial.

CSS 3, comenzó a desarrollarse en 1998, sus primeros borradores se publicaron en junio de 1999. A diferencia de las versiones anteriores la especificación se divide en varios documentos separados llamados módulos. Esto permite que diferentes módulos se encuentren en diferentes estados de desarrollo y no todos pasen a ser recomendaciones oficiales del **W3C**. Un módulo debe primero pasar por la fase de "candidato" antes de ser aprobado.

CSS 4, última versión sobre la que no existe una especificación integrada ya que al igual que **CSS 3** se encuentra dividido en diferentes módulos. A partir de la división en módulos que se hizo en **CSS 3** la evolución de cada uno de ellos es independiente lo que provoca que actualmente la mayoría de módulos estén en especificación de nivel 3 mientras que otros (por ejemplo Selectores) se encuentre en nivel 4.

En el siguiente enlace encontrarás el soporte que cada uno de los navegadores más actuales tienen sobre las etiquetas CSS.

CSS Reference With Browser Support - W3shool

2 Cómo incluir CSS en un documento HTML o XHTML

Hay tres modos distintos de aplicar estilos css a los elementos de una página:

- **Declaración en línea:** se declara el estilo en la misma línea en que se va a aplicar. Esta opción está desaconsejada.
- **Declaración interna:** se declaran los estilos a emplear en la página, en el encabezado de dicha página, mediante la etiqueta `<style>`.
- **Declaración externa:** se declara la hoja de estilo que se va a emplear en la página, en el encabezado de dicha página, mediante la etiqueta `<link>`.



2.1. Declaración en línea

Dentro de la propia etiqueta mediante el atributo `style`.

Deben evitarse para preservar el principio de separación de contenidos y formato.

```
<p style="color: red;"> ... </p>
```

2.2. Declaración Interna

En el encabezado del documento dentro de las etiquetas `<style>`

```
<style type="text/css"> ... </style>
```

Ejemplo

El texto de los elementos `<p>` se mostrará en verde.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='UTF-8'>
    <title>CSS interna</title>
    <style>
      p {color:green}
    </style>
  </head>
  <body>
    <p>Hola</p>
  </body>
</html>
```

2.3. Declaración en archivo externo

En el encabezado mediante la etiqueta `<link>` dentro del elemento `<head>`:

```
<link rel="stylesheet" type="text/css" href="rutaArchivo.css">
```

El archivo de estilos tendrá extensión .css.

El código del archivo de estilos no tendrá etiqueta de declaración de estilo, por ejemplo:

```
h1 {background-color: #blue}
```

Otra forma de usar hojas de estilo externas es mediante la etiqueta `@import`. Es una directiva CSS no html.

```
<style type="text/css">
  @import url("formato1.css");
</style>
```

Ejemplo

El siguiente documento HTML utiliza un archivo CSS externo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='UTF-8'>
    <title>CSS externo</title>
    <link rel="stylesheet" href="parrafo_verde.css">
  </head>
  <body>
    <p>Hola</p>
  </body>
</html>
```

El contenido del fichero CSS es:

```
p {
  color: green;
}
```

El texto de los elementos <p> se mostrará en verde.

Almacena los dos ficheros en la misma carpeta para probar el ejemplo. El fichero CSS se tiene que llamar "parrafo_verde.css".

2.1 Elemento span

Este elemento se utiliza habitualmente para dar estilo a texto no marcado.

Ejemplo

```
<p> Parte de este párrafo <span style="color:red">está en rojo</span>
```

3 Sintaxis de las reglas de estilo

Cada uno de los estilos que componen una hoja de estilos CSS se denomina regla. Cada regla se forma por:

- **Selector:** indica el elemento o elementos HTML a los que se aplica la regla CSS
- **Llave de apertura, {**
- **Declaración:** especifica los estilos que se aplican a los elementos.
 - **Propiedad:** permite modificar el aspecto de un atributo del elemento.
 - **Valor:** indica el nuevo valor del atributo modificado en el elemento.
- **Llave de cierre, }.**

```
p { color: blue; }
```

En este caso **el selector es "p"** (párrafo), **la declaración es "color: blue"** donde encontramos la **propiedad color** y el **valor blue**.

Un archivo CSS puede contener infinitas reglas CSS, cada regla puede contener varios selectores y cada declaración puede estar formada por diferentes declaraciones.

4 Cascada y herencia de estilos

En ocasiones y dependiendo de cómo se haya hecho la definición de estilos es posible que el navegador se encuentre estilos contradictorios. Ante esta situación el navegador aplicará la siguiente precedencia

1. Declaración en línea
2. Declaración interna
3. Declaración externa
4. Propiedades por defecto del navegador

Por otro lado las hojas de estilo también permiten la herencia de propiedades. Si tenemos varios elementos **HTML** anidados los elementos más internos heredan los estilos de los externos en los que están anidados siempre y cuando ellos no los tengan definidos

5 Selectores

A la hora de aplicar estilos a nuestros elementos **HTML** necesitamos un mecanismo que permita identificar sobre cuál o cuáles de estos elementos queremos actuar. Para ello se utilizan los **selectores**, estos permiten identificar a qué elementos de nuestro código html vamos a aplicar el estilo definido. Existe diferentes selectores, a continuación veremos los más importantes.

Selector universal (*)

Sirve para seleccionar todos los elementos de la página.

```
* {margin:10px; padding: 5px};
```

Selectores de etiqueta

En este caso los estilos se aplican solo a la etiqueta.

```
p { text-align: center}
```

En este ejemplo, los párrafos serán alineados al centro.

Si queremos ajustar los mismos estilos a dos etiquetas diferentes podemos ponerlos separados por comas.

```
p, h1, h2 { text-align: center}
```

En este ejemplo, tanto los párrafos como los encabezados de tipo 1 y 2 serán alineados al centro.

5.1 Selectores de clase

En ocasiones no vamos a querer aplicar el mismo formato en todas las etiquetas del mismo tipo. Para ello podemos asignar a las etiquetas a una **clase** determinada, esto requiere que identifiquemos las etiquetas afectadas mediante el atributo **class** de la forma.

```
<p class="parrafoCentrado"> .... </p>
```

El selector se especificaría:

```
p.parrafoCentrado {text-align: center; }
```

De manera que este estilo se aplicaría a todas las etiquetas p que tengan el atributo class a valor parrafoCentrado.

Sin embargo si hacemos la misma definición pero omitimos el identificador de etiqueta:

```
.parrafoCentrado { text-align: center; }
```

Se aplicaría a todas las etiquetas que pertenezcan a la clase “parrafoCentrado” sean del tipo que sean. Por ejemplo a una que fuese:

```
<h1 class="parrafoCentrado">encabezado 1 centrado</h1>
```

Veamos a continuación un código de ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='UTF-8'>
    <title>Ejemplo del uso de clases en hojas de estilo</title>
    <style type="text/css">
      .clase_azul{color:blue}
      p.clase_roja{color:#ff0000; font-style:italic; font-weight:bolder; font-family:courier;}
    </style>
  </head>
  <body>
    <h3 class="clase_azul">Ejemplo del uso de clases en hojas de estilo</h3>
    <p>Cualquier elemento sobre el que apliquemos la clase clase_azul tendrá el texto azul.</p>
    <p class="clase_azul"> Incluso el párrafo.</p>
    <p class="clase_roja">Sobre el párrafo podemos aplicar la clase clase_roja y el texto será rojo, en negrita cursiva y la familia del texto courier.</p>
    <h3 class="clase_roja"> Pero este texto no aparecerá formateado ya que regla de la clase clase_roja solo actúa sobre párrafos.</h3>
  </body>
</html>
```

El resultado en el navegador sería:

Ejemplo del uso de clases en hojas de estilo

Cualquier elemento sobre el que apliquemos la clase clase_azul tendrá el texto azul.

Incluso el párrafo.

Sobre el párrafo podemos aplicar la clase clase_roja y el texto será rojo, en negrita cursiva y la familia del texto courier.

Pero este texto no aparecerá formateado ya que regla de la clase clase_roja solo actúa sobre párrafos.

5.2 Selectores de ID

Permite seleccionar un elemento de la página por medio de su atributo id. El uso de este selector se suele asociar a elementos de estilo que se van a aplicar de manera excepcional una única vez, por ello el valor del atributo id no debe repetirse en dos elementos diferentes de la misma página.

Por ejemplo:

```
<p id="unico">...</p>
```

Se referencia mediante una de las siguientes posibilidades:

```
#unico { color: blue;}
```

```
p#unico {color: blue;}
```

5.3 Selectores descendentes

Permite seleccionar elementos que se encuentran dentro de otros elementos, por ejemplo:

```
p h1 {color: red; }:
```

Se aplicará a todas las etiquetas h1 que estén dentro de bloques p. Además hay que tener en cuenta que:

- No tiene porqué ser descendiente directo.
- El nivel de anidación puede tener varios niveles, por ejemplo:

```
p a b i {text-decoration: underline; }
```

Se aplica a los elementos en cursiva, dentro de etiquetas de negrita, anidados dentro de enlaces que se encuentren en párrafos.

Es importante tener en cuenta que si no tenemos cuidado podemos confundir el selector descendente con la aplicación del mismo selector a distintas etiquetas

```
p a b i {color: blue; }
```

NO ES LO MISMO QUE

```
p,a,b,i {color: blue; }
```

También podemos combinar el selector universal con selectores descendentes, por ejemplo:

```
p * b {color: #0000FF; }
```

Se aplica a todas las etiquetas de tipo que estén anidadas en cualquier otra etiqueta que a su vez esté dentro de una etiqueta de tipo <p>.

Sin embargo no se aplicará a las etiquetas de tipo que estén dentro de una etiqueta de tipo <p> directamente.

5.4 Selector hijo

Es parecido al anterior, pero solo afecta al primer nivel de anidamiento.

Ejemplo

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='UTF-8'>
    <title>Selector hijo</title>
    <style>
      section > p {color:red}
    </style>
  </head>
  <body>
    <p>Párrafo inicial</p>
    <section>
      <p>Párrafo hijo de sección</p>
      <article>
        <p>Párrafo nieto de sección</p>
      </article>
    </section>
  </body>
</html>
```

Se vería así:

Párrafo inicial

Párrafo hijo de sección

Párrafo nieto de sección

6 Propiedades principales

En los siguientes subapartados vamos a ver las propiedades principales que se usan en CSS como son:

- Propiedades de color y fondo.
- Propiedades de fuente.
- Propiedades de texto.
- Propiedades de caja.
- Propiedades de clasificación.

Pasemos a verlos detenidamente.



6.1 Propiedades de color y fondo

Las propiedades de color y fondo son los que enumeramos a continuación:

Elemento	Descripción
color	Indica el color del texto. Lo admiten casi todas las etiquetas de HTML. El valor de este atributo es un color, con su nombre o su valor RGB.
background-color	Indica el color de fondo del elemento. El valor de este atributo es un color, con su nombre o su valor RGB.
background-image	Permite colocar una imagen de fondo del elemento. El valor que toma es el nombre de la imagen con su camino relativo o absoluto
background-repeat	Indica si ha de repetirse la imagen de fondo y, en ese caso, si debe ser horizontal o verticalmente. Los valores que puede tomar son: repeat-x, repeat-y o no-repeat.
background-attachment	Especifica si la imagen ha de permanecer fija o realizar un scroll. Los valores que pueden tomar son: scroll o fixed.
background-position	Es una medida, porcentaje o el posicionamiento vertical u horizontal con los valores establecidos que sirve para posicionar una imagen. Los valores que puede tomar son: porcentaje, tamaño, o [top, center, bottom] [left, center, right]
background	Establece en un solo paso cualquiera de las propiedades de background anteriores. Los valores que puede tomar son: background-color, background-image, background-repeat, background-attachment, background-position.

Dado que no todos los nombres de colores son admitidos en el estándar, es aconsejable utilizar el valor RGB.

Un ejemplo de un documento XHTML en el que se utiliza este método para incluir formatos es:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo de atributos CSS de color y fondo</title>
    <style type="text/css">
      body { background-color: black; color:yellow; }
      p { color: #ffffff; }
    </style>
  </head>
  <body>
    <h3>Ejemplo del uso de atributos de color y fondo</h3>
    <p>El texto de cualquier elemento, salvo el del párrafo que es blanco, es amarillo y el fondo negro.</p>
  </body>
</html>
```

En el navegador se ve del siguiente modo:

Ejemplo del uso de atributos de color y fondo

El texto de cualquier elemento, salvo el del párrafo que es blanco, es amarillo y el fondo negro.

6.2 Propiedades de fuente

En este apartado vamos a ver las distintas propiedades que podemos utilizar referentes a las fuentes de nuestro documento y que son:

Elemento	Descripción
font-size	Indica el tamaño de la fuente. Puede ser un tamaño absoluto, relativo o en porcentaje. Toma valores de unidades de CSS
font-family	Establece la familia a la que pertenece la fuente. Si el nombre de una fuente tiene espacios se utilizan comillas para que se entienda bien. El valor es el nombre de la familia fuente.
font-weight	Define el grosor de los caracteres. Los valores que puede tomar son: normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800 o 900
font-style	Determina si la fuente es normal o cursiva. El estilo oblique es similar al cursiva. Los valores posibles son: normal, italic, oblique.
font-variant	Determina si la fuente es normal o mayúsculas pequeñas. Los valores que puede tomar son: normal o small-caps
line-height	El alto de una línea y por tanto, el espaciado entre líneas. Es una de esas características que no se podían modificar utilizando HTML.
font	Permite establecer todas las propiedades anteriores en el orden que se indica a continuación: font-style, font-variant, font-weight, font-size[line-height], font family. Los valores han de estar separados por espacios. No es obligatorio el uso de todos los valores.

Un ejemplo de un documento XHTML en el que se utiliza este método para incluir formatos es:

```
<!DOCTYPE html>
<html>
<head>
<title>Ejemplo de atributos CSS de fuente</title>
<style type="text/css">
    body { background-color: black; color:yellow; font-family: courier }
    p { color: #ffffff; font:italic 900 12px Verdana; }
</style>
</head>
<body>
<h3>Ejemplo del uso de atributos de fuente</h3>
<p>El texto de cualquier elemento es de la familia Courier y amarillo, salvo el del párrafo que es Verdana, blanco y de tamaño 12 px.</p>
</body>
</html>
```

En el navegador el resultado es:

Ejemplo del uso de atributos de fuente
El texto de cualquier elemento es de la familia Courier y amarillo, salvo el del párrafo que es Verdana, blanco y de tamaño 12 px.

6.3 Propiedades de texto

En el apartado anterior vimos los atributos relacionados con las fuentes y en este vamos a ver los relacionados con el texto en sí y son los siguientes:

Elemento	Descripción
text-decoration	Establece si el texto está subrayado, sobrerayado o tachado. Los valores que puede tomar son: none, underline, overline, line-through o blink
text-align	Indica la alineación del texto. Aunque las hojas de estilo permiten el justificado de texto no funciona en todos los sistemas. Los valores que puede tomar son: left, right, center o justify
text-indent	Determina la tabulación del texto. Los valores que toma son una longitud, en unidades CSS, o un porcentaje de la establecida.
text-transform	Nos permite transformar el texto, haciendo que tenga la primera letra en mayúsculas de todas las palabras, todo en mayúsculas o minúsculas. Los valores que puede tomar son: capitalize, uppercase, lowercase o none
word-spacing	Determina el espaciado entre las palabras. Los valores que puede tomar es un tamaño.
letter-spacing	Determina el espaciado entre letras. Los valores que puede tomar es un tamaño.
vertical-align	Establece la alineación vertical del texto. Sus valores posibles son: baseline, sub, super, top, text-top, middle, bottom, text-bottom o un porcentaje.
line-height	Altura de la línea. Puede establecerse mediante un tamaño o un porcentaje

Un ejemplo de un documento XHTML en el que se utiliza este método para incluir formatos es:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo de atributos CSS de texto</title>
    <style type="text/css">
      h3 { text-decoration:underline; text-align: center; text-transform: capitalize }
      p { text-indent: 50%; }
    </style>
  </head>
  <body>
    <h3>Ejemplo del uso de atributos de texto</h3>
    <p>El texto de del encabezado de tercer nivel está subrayado, centrado y la primera letra de cada palabra es mayúcula.</p>
    <p>El párrafo está tabulado</p>
  </body>
</html>
```

Su aspecto en el navegador es el siguiente:

The screenshot shows a browser window titled "Ejemplo de atributos CSS de texto". The address bar shows the URL: https://fpdistancia.educa.madrid.org/pluginfile.php/95215/mod_scorm/content/1/LMSG102_CONT_R16_Codigo15.html. The page content is titled "Ejemplo Del Uso De Atributos De Texto". It contains two paragraphs. The first paragraph has its first letter capitalized and is underlined. The second paragraph is indented by 50% from the left margin. The browser interface includes standard navigation buttons (back, forward, search) and a toolbar with icons for refresh, stop, and other functions.

El texto de del encabezado de tercer nivel está subrayado, centrado y la primera letra de cada palabra es mayúcula.

El párrafo está tabulado

6.4 Propiedades de listas

Hay cuatro propiedades de estilo para listas.

Elemento	Descripción
<code>list-style-type</code>	Indica cual es el símbolo que se utiliza como marcador en las listas. Valores que puede tomar son: disc, circle, square, decimal, lower-roman, upper-roman, lower-alpha, upper-alpha, none.
<code>list-style-image</code>	Permite utilizar el uso de una imagen como marcador en una lista. El valor que toma es la ruta del fichero imagen
<code>list-style-position</code>	Determinan la posición del marcador en una lista. Puede tomar los valores: outside o inside.
<code>list-style</code>	Permite establecer de una única vez todas las características de una lista. Hay que seguir el orden siguiente: list-style-type, list-style-position y list-style-image.

```
<!DOCTYPE html>
<html>
<head>
<meta charset='UTF-8'>
<title>Estilo para listas</title>
<style>
#flecha { list-style-image: url("flecha.png") }
.circ { list-style-type: circle }
.armenio { list-style-type: armenian }
</style>
</head>
<body>
<p>Lista con imagen</p>
<ul id="flecha">
<li>Patatas</li>
<li>Peras</li>
</ul>
<p>Lista con círculo</p>
<ul class="circ">
<li>Patatas</li>
<li>Peras</li>
</ul>
<p>Alfabeto armenio</p>
<ol id="armenio" reversed>
<li>Peras</li>
<li>Manzanas</li>
</ol>
</body>
</html>
```

Que en el navegador se ve del siguiente modo:

Lista con imagen

-  Patatas
-  Peras

Lista con círculo

- Patatas
- Peras

Alfabeto armenio

- Ա. Peras
- Ա. Manzanas

Para probar el ejemplo, necesita tener la imagen de la flecha en el mismo directorio.

6.5 Propiedad display

Esta propiedad se puede usar para:

- Hacer que un elemento sea de bloque o de línea.
- Ocultarlo o hacerlo visible. Normalmente, a través de JavaScript.

Ejemplo

En este ejemplo, hay varios vínculos, que son elementos de línea. Pero hay una regla CSS que hace que los que tienen clase "especial" sean de bloque.

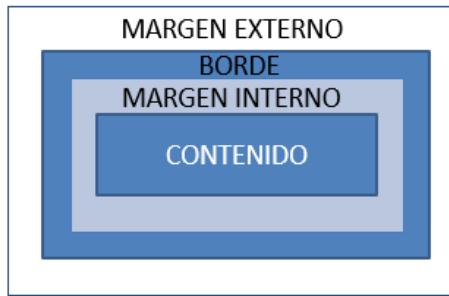
```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de atributos CSS de fuente</title>
    <style>
        .especial { display: block; }
    </style>
</head>
<body>
    <a href="#">Primer vínculo (normal)</a>
    <a href="#">Segundo vínculo (normal)</a>
    <a href="#" class="especial">Tercer vínculo (especial)</a>
    <a href="#" class="especial">Cuarto vínculo (especial)</a>
</body>
</html>
```

Se verá así:

[Primer vínculo \(normal\)](#) [Segundo vínculo \(normal\)](#)
[Tercer vínculo \(especial\)](#)
[Cuarto vínculo \(especial\)](#)

7 Avanzado: Modelo de cajas

La W3C define lo que se denomina "Modelo de caja", que no es más que una zona rectangular como la siguiente que rodea cada uno de los elementos de nuestra página web.



Cada etiqueta **HTML** aplica ese modelo y por lo tanto tiene:

- **Contenido**. Texto de la etiqueta
- **Margen interior**. Distancia desde el contenido al borde del elemento. Propiedad **HTML padding**.
- **Borde**. El borde del elemento. Propiedad **HTML border**.
- **Margen exterior**. Distancia desde el borde del elemento a los elementos adyacentes. Propiedad **HTML margin**.

Cada uno de esos elementos puede definirse mediante propiedades CSS que veremos a continuación.

7.1 Propiedades de caja

A continuación veremos algunas de las propiedades que afectan a cada uno de los elementos del modelo de caja.

Contenido

FORMATO	PROPIEDAD	VALORES
anchura	width	auto Longitud en px o equivalente
altura	height	auto Longitud en px o equivalente

Tanto **width** como **height** especifican las dimensiones del contenido sin tener en cuenta borde y márgenes.

Margen interno

FORMATO	PROPIEDAD	VALORES
Relleno / margen interno	padding padding-top padding-bottom padding-right padding-left	Auto valor de longitud valor de porcentaje

Si utilizamos **padding** con un solo valor se aplica a los cuatro lados, con dos valores el primero se aplica a superior e inferior y el segundo a laterales, con tres valores se aplica el primero al superior, el segundo a los laterales y el tercero al inferior, con cuatro valores se aplica a superior, derecho, inferior, izquierdo.

También es posible fijar el valor de cada una de los cuatro valores independientemente con la propiedad correspondiente.

borde

FORMATO	PROPIEDAD	VALORES
Color del borde	border-color border-top-color border-bottom-color border-right-color border-left-color	Color en alguna de las notaciones permitidas transparent
Grueso del borde	Border-width Border-top-width Border-bottom-width Border-right-width Border-left-width	Valor de longitud thin medium thick
Estilo del borde	border-style border-top-style border-bottom-style border-right-style border-left-style	Solid dashed dotted double ridge Groove inset outset hidden none

Si utilizamos **border-color**, **border-width** o **border-style** con un solo valor se aplican a los cuatro lados, con dos valores el primero se aplica a superior e inferior y el segundo a laterales, con tres valores se aplica el primero al superior, el segundo a los laterales y el tercero al inferior, con cuatro valores se aplica a superior, derecho, inferior, izquierdo.

También es posible fijar el valor de cada una de los cuatro valores independientemente con la propiedad correspondiente.

Margen externo

FORMATO	PROPIEDAD	VALORES
Ancho del margen externo	margin margin-top margin-bottom margin-right margin-left	Auto valor de longitud valor de porcentaje

Si utilizamos **margin** con un solo valor se aplica a los cuatro lados, con dos valores el primero se aplica a superior e inferior y el segundo a laterales, con tres valores se aplica el primero al superior, el segundo a los laterales y el tercero al inferior, con cuatro valores se aplica a superior, derecho, inferior, izquierdo.

También es posible fijar el valor de cada una de los cuatro valores independientemente con la propiedad correspondiente.

7.2 Unidades de tamaño

A la hora de especificar tamaños CSS nos permite usar diferentes tipos de unidades, estas además pueden ser absolutas o relativas:

- **Absolutas.** Cualquier longitud expresada en una de estas unidades siempre se mostrará del mismo tamaño.
- **Relativas.** Dependen del tamaño de otro elemento.



Normalmente se utilizan unidades absolutas para la secciones y la estructura de la página web porque se adaptan mejor a diferentes tamaños de pantalla. Las más habituales son los porcentajes y la unidad **em**.

Para las dimensiones de imágenes y vídeos se suelen utilizar píxeles. Es la única unidad absoluta que se utiliza habitualmente.

Se debe especificar la unidad usada:

- Por ejemplo si queremos especificar un tamaño de 13 píxeles utilizaremos: **13px**

Unidades absolutas	
cm	Centímetros
mm	Milímetros
In	Pulgadas
px *	Píxeles
pt	Puntos
pc	Picas

*: Los píxeles son relativos a la pantalla.

UNIDADES RELATIVAS	
em	Relativa al tamaño del tipo de letra por defecto
porcentajes (%)	Relativos a las dimensiones del elemento contenedor
ex	Relativa al valor de x-height de la fuente actual
ch	Relativa al ancho del cero “0”
rem	Relativa al tamaño de letra del elemento raíz

Ejemplo de porcentajes

En esta página se crean dos secciones. La primera tiene fondo verde y su anchura la mitad que el elemento contenedor, body, que al ser un elemento de bloque ocupa todo la anchura disponible.

Las dos secciones también ocuparían toda la anchura si no fuera por las reglas de estilo.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Porcentajes</title>
    <style>
      #s1 {
        background-color:green;
        width:50%;
      }
      #s2 {
        background-color:blue;
        width:30%;
      }
    </style>
  </head>
  <body>
    <section id = "s1">Contenido sección 1</section>
    <section id = "s2">Contenido sección 2</section>
  </body>
</html>
```

Se vería así:

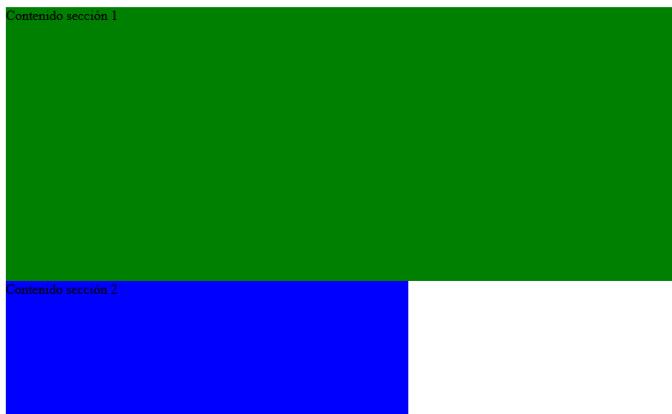
Contenido sección 1
Contenido sección 2

Ejemplo de porcentajes (2).

Con la altura (**height**) ocurre lo mismo, pero hay que tener en cuenta que depende del contenido que haya en la página, a no ser que se fije un valor para el elemento como en este ejemplo. El elemento **<body>** tiene una altura fijada en píxeles. Las secciones tienen la altura y anchura expresadas en porcentajes relativos a los del elemento **<body>**.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Porcentajes</title>
    <style>
      body{
        height:800px;
      }
      #s1{
        background-color:green;
        width:50%;
        height:40%;
      }
      #s2{
        background-color:blue;
        width:30%;
        height:20%;
      }
    </style>
  </head>
  <body>
    <section id = "s1">Contenido sección 1</section>
    <section id = "s2">Contenido sección 2</section>
  </body>
</html>
```

Se vería así:



Ejemplo de unidades em

Para las fuentes es habitual utilizar la unidad em, que hace referencia al tamaño de la fuente actual.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Unidades em</title>
    <style>
      .grande {font-size: 2em;}
      .muyGrande {font-size: 4em}
    </style>
  </head>
  <body>
    <p class="grande">Este es un texto grande</p>
    <p class="muyGrande">Este es un texto muy grande</p>
  </body>
</html>
```

```
</style>
</head>
<body>
  <p>Normal</p>
  <p class = "grande">Grande</p>
  <p class = "muyGrande">Muy grande</p>
</body>
</html>
```

Se vería así:

Normal

Grande

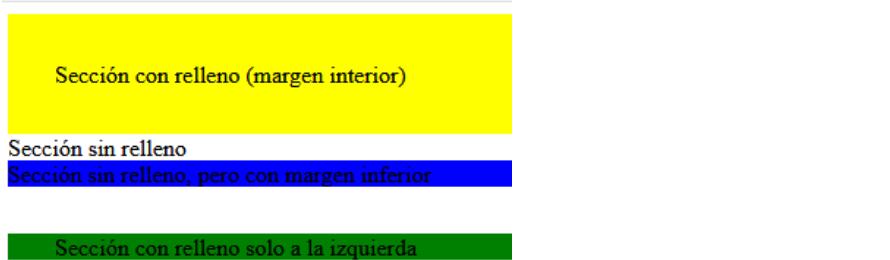
Muy grande

7.3 Ejemplos del modelo de cajas

Ejemplo de relleno y margen

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Relleno y margen</title>
    <style>
      #relleno {
        background-color:yellow;
        padding: 2em; /*se aplica a izquierda, derecha arriba y abajo*/
      }
      #rellenoIzq{
        background-color:green;
        padding-left: 2em; /*se aplica a izquierda, derecha arriba y abajo*/
      }
      #rellenoMargen{
        background-color:blue;
        margin-bottom: 2em; /*se aplica a izquierda, derecha arriba y abajo*/
      }
    </style>
  </head>
  <body>
    <section id = "relleno">Sección con relleno (margen interior)</section>
    <section>Sección sin relleno</section>
    <section id = "rellenoMargen">Sección sin relleno, pero con margen inferior</section>
    <section id = "rellenoIzq">Sección con relleno solo a la izquierda</section>
  </body>
</html>
```

Se verá así:



El espacio en blanco (color de fondo de **<body>**) entre las secciones y azul y verde se debe al margen inferior de la primera.

Ejemplo de border

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Relleno y margen</title>
    <style>
      #borde1 {
        background-color:yellow;
        border-color: green;
        border-style: solid;
      }
      #borde2{
        background-color:pink;
        border-width: 16px;
        border-left-color: green;
        border-top-color: green;
        border-left-style: dashed;
        border-top-style: dashed;
      }
    </style>
  </head>
  <body>
    <section id = "borde1">Sección con borde1</section>
    <br>
    <section id = "borde2">Sección con borde2</section>
  </body>
</html>
```

Se verá así:



7.4 Posicionamiento

En los ejemplos vistos hasta ahora, el navegador representa los elementos en el orden en el que aparecen atendiendo a si son elementos de bloque o de línea (entre otras cosas). Utilizando CSS, es posible modificar el posicionamiento por defecto de los elementos. Las propiedades implicadas son **position** y **float**.

Con la propiedad **float**, los elementos "flotan" hacia la izquierda o derecha. Todos los elementos flotados se van situando uno junto a otro. Si no hay espacio disponible, se pasan a una nueva línea. Se adapta bastante bien a diferentes tamaños de pantalla.

En este ejemplo se utiliza la propiedad **float** para maquetar una página sencilla, junto a las etiquetas semánticas. Los elementos **<nav>** y **<section>** están flotados a la izquierda y se reparten la anchura disponible.

Es necesario fijar la altura porque en la página apenas hay contenido.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Etiquetas semánticas y float</title>
    <style>
      body{
        background-color:pink;
      }
      header{ background-color:blue;}
      nav {
        background-color:red;
        width:10%;
      }
      section {
        background-color:green;
        width:90%;
      }
      footer {
        background-color:yellow;
      }
      nav, section {
        height:500px;
        float: left;
      }
    </style>
  </head>
  <body>
    <header>Encabezado</header>
    <nav>Vínculos</nav>
    <section>Contenido principal del página</section>
    <footer>Página creada por...</footer>
  </body>
</html>
```

Se vería así:

Para saber más

- Los métodos de posicionamiento se describen en detalle en este vínculo (en el capítulo 5).
- Aquí puedes ver más ejemplos de maquetación.

8 Validator CSS

- Validator CSS online

3.A. Aplicación de los lenguajes de marcas a la sindicación de contenidos

Sitio: Aula Virtual CIERD (CIDEAD)

Curso: Lenguaje de marcas y sistemas de gestión de información

Libro: 3.A. Aplicación de los lenguajes de marcas a la sindicación de contenidos

Imprimido por: MANUEL JOSE GONZALEZ CALVO

Día: jueves, 21 de mayo de 2020, 23:13

Tabla de contenidos

- 1 Sindicación de contenidos
- 2 Características
- 3 Ventajas de la sindicación de contenidos
- 4 Ámbito de aplicación
- 5 Tecnologías de creación de canales de contenidos
- 6 Estructura de los canales de contenidos
 - 6.1 RSS
- 7 Validación
- 8 Agregadores

1 Sindicación de contenidos

La redifusión, o sindicación de contenidos, permite a un sitio utilizar los servicios o contenidos ofertados por otro sitio diferente.

Un ejemplo de redifusión podemos encontrarlo en el mercado televisivo. Supongamos una serie de televisión, que es creada por una cadena de televisión. Al principio, sólo la emite esa cadena, en exclusividad. Pero con el paso del tiempo, la vendió a otras cadenas. Estas otras cadenas, al emitir la serie, hacen redifusión.



La **redifusión web** consiste en ofrecer un contenido desde una fuente web, cuyo origen está en otra página web. Se proporciona a los usuarios la actualización del mismo. Los servicios que ofrece el sitio web original, junto con los **metadatos** que tiene asociados en el sitio original, forman los **feed** o **canales de contenidos**. Para leer una fuente, o canal, hay que suscribirse a ella utilizando un **agregador**.

La **redifusión de contenidos web** suele realizarse bajo una **licencia de normas de uso**, o mediante un contrato que regule los derechos de los contenidos.

Las fuentes suelen codificarse en lenguaje **XML** (XML = eXtensible Markup Language, significa Lenguaje de Marcas Extensible), aunque es válido hacerlo en cualquier lenguaje que se pueda transportar mediante el protocolo **HTTP** (**HyperText Transfer Protocol**, significa Protocolo de Transferencia de Hipertexto).

Imagen: Bordo Clase Profesor enseñanza de Mohamed Hassan en Pixabay - Creative Commons CCO

2 Características

Publicación en la web

Publicar en la web puede ser visto como un flujo de información, que va desde un cierto **origen** hasta los usuarios y usuarias que la leerán. Podrán hacerlo a través de su navegador, es decir, accediendo a una página web disponible en Internet.

Supongamos que el flujo de información de una publicación tiene su origen en unos ficheros localizados en un ordenador local, codificados en un documento **HTML**.

Lo que tendremos que hacer, para hacer llegar esa información a los lectores, es subir dichos documentos al directorio adecuado del servidor web que contiene la página.

Sindicación

Podemos hacer que una web se convierta en suministradora (origen) de un canal de información, de modo que esta información pueda ser sindicada. Para lograrlo, en la cabecera de la página web hay que incluir, debajo del elemento `<title>`, un enlace al canal de contenidos.

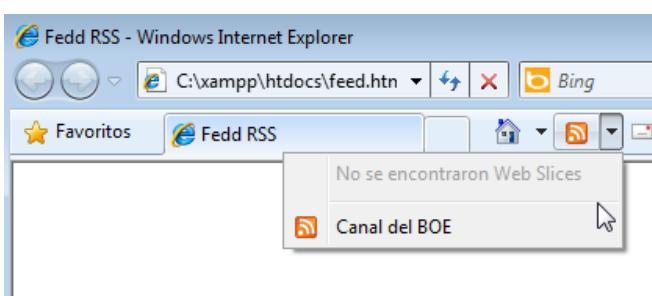
Para lograrlo, hay que usar una de las dos líneas siguientes, dependiendo de que el canal esté hecho con un estándar **RSS** o con uno **Atom**, respectivamente:

- `<link rel="alternate" type="application/rss+xml" title="titulo_que_tendrá_el_enlace" href="http://www.misitio.com/fichero.rss" />`
- `<link rel="alternate" type="application/atom+xml" title="titulo_que_tendrá_el_enlace" href="http://www.misitio.com/fichero.atom" />`

Al vincular un canal de esta manera, el resultado puede ser poco claro y cambiar entre navegadores. Por ejemplo, para el siguiente documento:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Fedd RSS</title>
    <link rel="alternate" type="application/rss+xml" title="Canal del BOE" href="https://www.boe.es/rss/boe.php" />
    <meta charset="UTF-8">
  </head>
  <body>
  </body>
</html>
```

El resultado en Internet Explorer sería el siguiente:



Es decir, se muestra un menú desplegable con las fuentes vinculadas (si la barra correspondiente está activada).

Para que quede más claro, se puede poner un vínculo normal, para el normalmente se utiliza el símbolo de RSS.

Sistemas de Gestión de Contenidos (CMS)

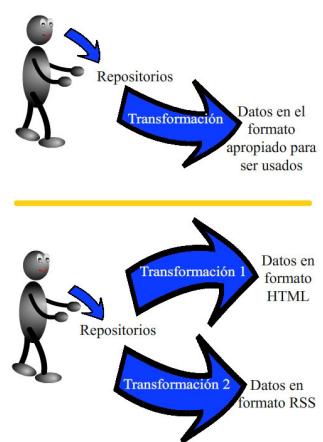
Actualmente es habitual el uso de algún **Sistema de Gestión de Contenidos**.

En este caso el origen de los contenidos es un repositorio y, antes de ser servidos al cliente en el formato adecuado, sufren algún tipo de transformación. La parte superior de la figura muestra la estructura del flujo de la información en este caso. Incluso puede haber más de un repositorio.

Esta transformación puede corresponder a uno de los siguientes casos:

- Documento **XML** -> Transformación **XSLT**-> Documento **XHTML**.
- Base de datos -> script en **Perl** -> Documento **HTML**.
- Texto plano -> Página de servidor activo -> Documento **HTML**.
- Mente del autor -> Bloc de notas -> Documento **HTML**.

Al utilizar un **CMS** de cualquier tipo la transformación puede replicarse. Además de tener más de una entrada de información podríamos tener varias salidas. Por ejemplo, podemos generar tanto ficheros **HTML** como canales **RSS** tal y como se muestra en la parte inferior de la figura.



3 Ventajas de la sindicación de contenidos

¿Cuáles serán las ventajas de utilizar los canales de contenidos de otros propietarios?

- Aumentar el tráfico de nuestro sitio web.
- Ayuda a que los usuarios y usuarias visiten frecuentemente el sitio web.
- Favorece el posicionamiento del sitio en buscadores.
- Ayuda a establecer relaciones entre distintos sitios web dentro de la comunidad.
- Permite a otras personas añadir características a los servicios del sitio web (por ejemplo, notificaciones de actualizaciones mediante mensajes instantáneos), aunque se requiera de tecnologías adicionales.
- Enriquece Internet impulsando la tecnología semántica y fomentando la reutilización.



Imagen: Medios de Comunicación Social de Geralt en Pixabay - Creative Commons CCO

Todos los logotipos de la Imagen son propiedad de sus respectivas empresas bajo licencia Copyright

4 Ámbito de aplicación

La redifusión web no es sólo un fenómeno vinculado a los **weblogs**, aunque ha ayudado mucho a su popularización. Siempre se han sindicado contenidos y se ha compartido todo tipo de información en formato **XML**.

De esta forma podemos ofrecer contenidos propios para que sean mostrados en otras páginas web de forma integrada, lo que aumenta el valor de la página que muestra el contenido y también nos genera más valor, ya que normalmente la redifusión web siempre enlaza con los contenidos originales.

La redifusión de contenidos web puede aplicarse a todo tipo de contenidos, es decir, texto, audio, vídeos e imágenes.

Desde el punto de vista de los suscriptores, la redifusión de contenidos permite, entre otras cosas, la actualización profesional. Mediante la suscripción a sitios relevantes, el usuario o la usuaria puede estar al día en temas relacionados con su profesión, recibiendo las noticias e informaciones en su blog o en su programa agregador de noticias.



5 Tecnologías de creación de canales de contenidos

Los estándares más utilizados se clasifican en dos grupos:

RSS: (Really Simple Syndication) es parte de la familia de los formatos XML, desarrollado para compartir la información que se actualiza con frecuencia entre sitios web. Además se utiliza para:

- Conectar con sistemas de mensajería instantánea.
- Conversión RSS en mensajes de correo electrónico.
- Transformar los enlaces favoritos del navegador en RSS.

Blog abonnieren	
RSS 0.91 feed	
RSS 1.0 feed	
RSS 2.0 feed	
ATOM 0.3 feed	
ATOM 1.0 feed	
RSS 2.0 Kommentare	

Ha sido desarrollado por tres organizaciones diferentes, lo que ha dado lugar a siete formatos diferentes entre sí:

- **RSS 0.90**, es el estándar que creó la empresa Netscape en el año 1999. Se basa en la especificación RDF de metadatos, con la intención de que su proyecto My Netscape estuviese formado por titulares de otras webs.
- **RSS 0.91**, es la versión simplificada de RSS 0.90 que Netscape lanzó posteriormente. El desarrollo de este formato se detuvo por falta de éxito, aunque la empresa UserLand Software decidió usar esta versión para desarrollar blogs.
- **RSS 1.0**, fue creado a partir del estándar el RSS 0.90. Es más estable y permite definir una cantidad mayor de datos que el resto de versiones de RSS.
- **RSS 2.0**, UserLand Software rechazó el estándar RSS 1.0 por considerarlo complejo y continuó el desarrollo del formato RSS 0.91, publicando las versiones 0.92, 0.93 y 0.94. Su sintaxis está incompleta y no cumplen todas las normas de XML. El estándar RSS 2.0 se publicó para subsanar esos problemas.

Atom: fue publicado como un estándar propuesto por el grupo de trabajo **Atom Publishing Format and Protocol** (Formato y protocolo de publicación **Atom**) de la IETF en el **RFC4287**. Se desarrolló como una alternativa a **RSS**, con el fin de evitar la confusión creada por la existencia de estándares similares para la sindicación de contenidos, entre los que existía cierta incompatibilidad. En lugar de sustituir a los estándares existentes, se creó un nuevo estándar que convive con ellos. Se caracteriza por su flexibilidad. Atom permite tener un mayor control sobre la cantidad de información a representar en los agregadores.

6 Estructura de los canales de contenidos

Para construir un canal de contenido, es necesario crear un fichero, con extensión **rss** o **atom**, basado en **XML**. Este fichero se publicará en uno de los directorios del sitio web desde el que se oferta.

Estará formado por los siguientes elementos básicos:

- **Declaración del documento xml y la definición de la codificación empleada** en el documento. Ésta última será preferentemente **UTF-8**.
- **Un canal** en el que se determina el sitio web asociado a la fuente web a la que hace referencia el fichero. Éste, además de su propia definición, estará formado por:
 - **Secciones**, cada una de las cuales es una referencia a la web que contiene uno de los servicios que se van a ofrecer. En un canal pueden incluirse tantas secciones como se quiera, lo que hace que un canal de contenido pueda tener un tamaño enorme si contiene un gran número de enlaces independientes.

No existe ninguna restricción respecto a la cantidad de canales de contenidos que se pueden ofrecer desde un sitio web.

6.1 RSS

El documento RSS incluye como primera linea la declaración del documento XML, normalmente:

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

A continuación aparece la etiqueta **<rss>**. Es declaración RSS que indica que es un documento RSS y la versión empleada.

Dentro de ella, aparece un canal (etiqueta **<channel>**), que se encarga de describir el feed RSS propiamente dicho. Tiene tres elementos hijos obligatorios:

- **<title>** - Define el título del canal
- **<link>** - Define el hiperenlace al canal
- **<description>** - Describe el canal

También hay varios elementos opcionales. Algunos de ellos son:

- **<language>** - Define el idioma del canal
- **<category>** - Define una o más categorías a las que pertenece la fuente
- **<copyright>**

Cada canal tiene uno o más artículos o secciones (etiqueta **<item>**), cada uno de los cuales cuenta "una historia" del canal. Tiene tres elementos hijos obligatorios:

- **<title>** - Define el título del artículo
- **<link>** - Define el hiperenlace al artículo
- **<description>** - Describe el artículo

También hay varios elementos opcionales. Algunos de ellos son:

- **<author>** - Define el autor del artículo
- **<category>** - Define una o más categorías a las que pertenece la fuente
- **<guid>** - Define un identificador único para el elemento

Esquema (con un item)

```
<?xml version="1.0" encoding="UTF-8"?>
<rss>
  <channel>
    <title></title>
    <link></link>
    <description></description>
    <item>
      <title></title>
      <link></link>
      <description></description>
    </item>
  </channel>
</rss>
```

Ejemplo

Como ejemplo, veamos el canal RSS del Boletín Oficial del Estado. En la dirección <https://boe.es/rss/boe.php> podemos encontrar la información para el último BOE publicado.

```
-<rss version="2.0">
-<channel>
  <title>BOE - Boletín Oficial del Estado</title>
  <link>http://www.boe.es/diario_boe/</link>
-<description>
  Leyes, disposiciones, actos, textos legales y anuncios publicados en la edición de hoy
</description>
<language>es-es</language>
<pubDate>Sat, 21 Dec 2019 00:00:00 +0100</pubDate>
<lastBuildDate>Sat, 21 Dec 2019 00:00:00 +0100</lastBuildDate>
<webMaster>webmaster@boe.es</webMaster>
-<item>
  <title>Sumario</title>
  <link>http://www.boe.es/boe/dias/2019/12/21/</link>
-<description>
  Sumario del diario núm. 306, correspondiente al sábado 21 de diciembre de 2019. - Referencia: BOE-S-2019-306 - KBytes: 479
</description>
<category>Sumario</category>
<guid isPermaLink="true">http://www.boe.es/boe/dias/2019/12/21/</guid>
<pubDate>Sat, 21 Dec 2019 00:00:00 +0100</pubDate>
</item>
-<item>
  <title>
    Acuerdo entre el Gobierno del Reino de España y la Secretaría de la Convención Marco de las Naciones Unidas sobre el Cambio Climático, el Protocolo de Kioto y el Acuerdo de París, hecho en Madrid y 1 noviembre de 2019.
  </title>
  <link>
    http://www.boe.es/diario_boe/txt.php?id=BOE-A-2019-18359
  </link>
-<description>
  I. Disposiciones generales - MINISTERIO DE ASUNTOS EXTERIORES, UNIÓN EUROPEA Y COOPERACIÓN - Acuerdos internacionales administrativos - Referencia: BOE-A-2019-18359 - KBytes: 292 - Páginas
</description>
<category>
  Disposiciones generales</category>
-<category>
  MINISTERIO DE ASUNTOS EXTERIORES, UNIÓN EUROPEA Y COOPERACIÓN
</category>
<guid isPermaLink="true">
  http://www.boe.es/boe/dias/2019/12/21/pdfs/BOE-A-2019-18359.pdf
</guid>
<pubDate>Sat, 21 Dec 2019 00:00:00 +0100</pubDate>
</item>
-<item>
  <title>
    Resolución de 20 de diciembre de 2019, de la Presidencia del Comisionado para el Mercado de Tabacos, por la que se publican los precios de venta al público de determinadas labores de tabaco en Expe
  </title>
```

Con el navegador se pueden contraer y expandir los elementos para ver el documento más cómodamente.

```
-<rss version="2.0">
-<channel>
  <title>BOE - Boletín Oficial del Estado</title>
  <link>http://www.boe.es/diario_boe/</link>
-<description>
  Leyes, disposiciones, actos, textos legales y anuncios publicados en la edición de hoy
</description>
<language>es-es</language>
<pubDate>Sat, 21 Dec 2019 00:00:00 +0100</pubDate>
<lastBuildDate>Sat, 21 Dec 2019 00:00:00 +0100</lastBuildDate>
<webMaster>webmaster@boe.es</webMaster>
+<item></item>
```

El elemento raíz es **rss**. Tiene un único hijo, **channel**, que define el canal de noticias. Como hijos de este elemento hay varios elementos **item**, uno por cada noticia publicada en el canal.

Para cada **item**, hay un título (**title**), un vínculo (**link**), una descripción (**descripción**), una o más categorías (**category**), un elemento **guid** y una fecha de publicación (**pubDate**).

El primer elemento **item** es el sumario, como se puede ver en el título, la descripción y la categoría.

```
-<item>
  <title>Sumario</title>
  <link>http://www.boe.es/boe/dias/2019/12/21/</link>
-<description>
  Sumario del diario núm. 306, correspondiente al sábado 21 de diciembre de 2019. - Referencia: BOE-S-2019-306 - KBytes: 479
</description>
<category>Sumario</category>
<guid isPermaLink="true">http://www.boe.es/boe/dias/2019/12/21/</guid>
<pubDate>Sat, 21 Dec 2019 00:00:00 +0100</pubDate>
</item>
..
```

En este caso los elementos elemento **guid** y **link** coinciden. Si se accede a esa URL en el navegador, se encuentra, efectivamente, el sumario del BOE para esa fecha.

El resto de elementos **item** se corresponden con los artículos publicados ese día. Tienen dos elementos **category** y los elementos **link** y **guid** no son iguales.

Si nos fijamos en el segundo, se trata de la publicación de un acuerdo internacional.

```
-<item>
-<title>
    Acuerdo entre el Gobierno del Reino de España y la Secretaría de la Convención Marco de las Naciones Unidas sobre Cambio Climático, sobre la ratificación por parte de España del Acuerdo de París, firmado en noviembre de 2019.
</title>
-<link>
    http://www.boe.es/diario_boe/txt.php?id=BOE-A-2019-18359
</link>
-<description>
    I. Disposiciones generales - MINISTERIO DE ASUNTOS EXTERIORES, UNIÓN EUROPEA Y COOPERACIÓN - Acuerdos y convenios
</description>
<category>I. Disposiciones generales</category>
-<category>
    MINISTERIO DE ASUNTOS EXTERIORES, UNIÓN EUROPEA Y COOPERACIÓN
</category>
-<guid isPermaLink="true">
    http://www.boe.es/boe/dias/2019/12/21/pdfs/BOE-A-2019-18359.pdf
</guid>
<pubDate>Sat, 21 Dec 2019 00:00:00 +0100</pubDate>
</item>
```

Tiene dos elementos **category**, uno para indicar que es una disposición general, y otro para indicar que se trata del Ministerio de Asuntos Exteriores.

El elemento **link** contiene un vínculo que lleva a una página con el contenido en HTML. El elemento **guid** lleva a la versión en PDF del mismo contenido.

7 Validación

En internet hay múltiples lugares que dan este servicio.

Congratulations!

Para validar un documento **RSS** con uno de estos validadores, se le da la dirección del fichero donde se encuentra alojado y comprueba que lo pueden encontrar, es decir que la URI es válida, y que no contiene errores.



This is a valid RSS feed.

Una vez validado, suelen ofrecer una imagen del tipo "**XML**" o "**RSS**", de color naranja por lo general, que se puede incluir en la página principal, para enlazar a la dirección del fichero alojado en su dominio. Así, cuando un visitante pulse sobre este pequeño ícono, accederá directamente al contenido actual de la fuente y podrá navegar a través de él a las páginas que más le interesen.

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

Algunos de los validadores que podemos encontrar en Internet son:

- FeedValidator.
- W3C Feed Validation Service mediante URI.
- W3C Feed Validation Service mediante código.
- RSS Advisory Board.
- Googletransitdatafeed.

Imagen: RSS Feed Validado en el validador del W3C - Jorge Castellanos - Creative Commons CCO

Nota: El logotipo RSS fue creado por Stephen Horlander - Diseñador de Mozilla para los marcadores de Firefox

8 Agregadores

¿Qué es un agregador o lector de fuentes?

Es una aplicación de software para suscribirse a fuentes en formatos **RSS** y **Atom**. El agregador avisa al usuario o usuaria de qué webs han incorporado contenido nuevo desde nuestra última lectura y cuál es ese contenido.

En el agregador hay que indicar la dirección web de cada archivo fuente, ya sea en formato **RSS** o **Atom**, para que pueda acceder a sus contenidos, los interprete y los muestre.

Existen varios tipos de agregadores:



- **Los agregadores web (o agregadores en línea)**, son aplicaciones que residen en determinados sitios web y que se ejecutan a través de la propia web. Son recomendables cuando el usuario o la usuaria no accede siempre a Internet desde el mismo ordenador. Es el caso de Feedly, Inoreader o NewsBlur.
- **Los agregadores de escritorio**, son aplicaciones que se instalan en el ordenador del usuario o usuaria. Su uso es aconsejable para quienes accedan a Internet siempre desde el mismo ordenador. Su interfaz gráfica suele ser parecida a la de los programas de cliente de correo electrónico, con un panel donde se agrupan las suscripciones, y otro donde se accede a las entradas individuales para su lectura. Algunos ejemplos serían RSSOwl o .QuiteRSS. También podemos descargarnos por ejemplo la versión de escritorio de Feedly.

Imagen: Logo Feedly - Feedly Inc - Todos los derechos reservados

- **Complementos de navegador**. También hay agregadores disponibles como complementos de navegador web, como por ejemplo Awesome RSS o Livemarks.

4.A. Definición de esquemas y vocabularios XML - DTD

Sitio: Aula Virtual CIERD (CIDEAD)
Curso: Lenguaje de marcas y sistemas de gestión de información
Libro: 4.A. Definición de esquemas y vocabularios XML - DTD
Imprimido por: MANUEL JOSE GONZALEZ CALVO
Día: jueves, 21 de mayo de 2020, 23:14

Tabla de contenidos

- 1 Documento XML. Estructura y sintaxis.
- 2 Declaración de tipo de documento
- 3 Definición de la sintaxis de documentos XML
 - 3.1 Definiciones de tipo de documento, DTD.
 - 3.2 Declaraciones de tipos de elementos terminales
 - 3.3 Declaraciones de tipos de elementos no terminales
 - 3.4 Declaraciones de listas de atributos para los tipos de elementos
 - 3.5 Declaraciones de entidades
 - 3.6 Declaraciones de notación
 - 3.7 Secciones condicionales

1 Documento XML. Estructura y sintaxis.

Hasta ahora hemos trabajado con documentos básicos de XML. En la primera unidad vimos que un documento XML básico estaba formado por un prólogo y un ejemplar. Recordamos que cada una de esas partes tiene el siguiente cometido:

- **Prólogo:** Informa al intérprete encargado de procesar el documento de todos aquellos datos que necesita para realizar su trabajo. Consta de dos partes:
 - **Definición de XML:** Donde se indica la versión de XML que se utiliza, el código de los datos a procesar y la autonomía del documento. Este último dato hasta ahora siempre ha sido "yes" ya que los documentos generados eran independientes.
 - **Declaración del tipo de documento:** Hasta el momento solo hemos dicho que es el nombre del ejemplar precedido de la cadena <!DOCTYPE y separado de ésta por, al menos un espacio.
- **Ejemplar:** Contiene los datos del documento que se quiere procesar. **Es el elemento raíz del documento y ha de ser único.** Está compuesto de elementos estructurados según una estructura de árbol en la que el elemento raíz es el ejemplar y las hojas los elementos terminales, es decir, aquellos que no contienen elementos. Los elementos pueden estar a su vez formados por atributos.

Estos documentos básicos están **incompletos**, ya que solo hemos declarado el tipo de documento que va a ser (qué ejemplar vamos a definir), pero **no hemos definido qué cualidades tiene ese ejemplar**.

Para completarlos y definir en profundidad las cualidades del ejemplar, usaremos o bien DTDs o bien XML Schemas.

2 Declaración de tipo de documento

Ya habíamos visto que permite **al autor definir restricciones y características en el documento**, aunque no habíamos profundizado en las partes que la forman:

- **La declaración del tipo de documento propiamente dicha.**
Comienza con el texto que indica el nombre del tipo, precedido por la cadena " " separado del nombre del tipo por, al menos, un espacio. El nombre del tipo ha de ser idéntico al del ejemplar del documento XML en el que se está trabajando.
- **La definición del tipo de documento.** Permite asociar al documento una definición de tipo **DTD** , la cual se encarga de definir las cualidades del tipo. Es decir, define los tipos de los elementos, atributos y notaciones que se pueden utilizar en el documento así como las restricciones del documento, valores por defecto, etc. Para formalizar todo esto, XML está provisto de ciertas estructuras llamadas **declaraciones de marcado**, las cuales pueden ser internas o externas. Normalmente un documento XML se compone de una mezcla de declaraciones de marcado internas y externas. En este último caso debe expresarse en el documento dónde encontrar las declaraciones, así como indicar en la declaración de XML que el documento no es autónomo. Las diferencias entre estos tipos de declaraciones de marcado dan lugar a dos subconjuntos el interno y el externo, conviene saber que primero se procesa el subconjunto interno y después el externo, lo que permite sobrescribir declaraciones externas compartidas entre varios documentos y ajustar el DTD a un documento específico.
 - **Subconjunto interno:** Contiene las **declaraciones que pertenecen exclusivamente a un documento** y no es posible compartirlas. Se localizan dentro de unos corchetes que siguen a la declaración de tipo del documento.
 - **Subconjunto externo:** Están localizadas en un documento con extensión **dtd** que puede situarse en el mismo directorio que el documento XML. Habitualmente son declaraciones que pueden ser compartidas entre múltiples documentos XML que pertenecen al mismo tipo. En este caso la declaración de documento autónomo ha de ser negativa, ya que es necesario el fichero del subconjunto externo para la correcta interpretación del documento. Con ello el procesado del documento será más lento, ya que antes de procesar el documento el procesador ha de obtener todas las entidades.



- **<!DOCTYPE nombre_ejemplar SYSTEM "URI">**

En este caso, se especifica un URI donde pueden localizarse las declaraciones.

- **<!DOCTYPE nombre_ejemplar PUBLIC "id_publico" "URI">**

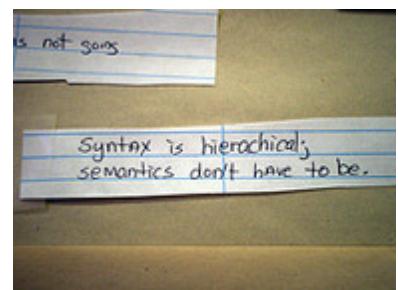
En este caso también se especifica un identificador, que puede ser utilizado por el procesador XML para intentar generar un URI alternativo, posiblemente basado en alguna tabla. **Como se puede observar también es necesario incluir algún URI.**

3 Definición de la sintaxis de documentos XML

Recordamos que en estos documentos las etiquetas de marcado describen la estructura del documento.

Un elemento es un grupo formado por una etiqueta de apertura, otra de cierre y el contenido que hay entre ambas.

En los documentos de lenguajes de marcas, la distribución de los elementos está jerarquizada según una estructura de árbol, lo que implica que es posible anidarlos pero no entrelazarlos.



Hemos visto que en los elementos el orden es importante, ¿lo es también para los atributos? En este caso el orden no es significativo. Lo que hay que tener presente es que no puede haber dos atributos con el mismo nombre.

Sabemos que los atributos no pueden tener nodos que dependan de ellos, por tanto solo pueden corresponder con hojas de la estructura de árbol que jerarquiza los datos. ¿Significa esto que todas las hojas van a ser atributos? Pues no, es cierto que los atributos son hojas, pero las hojas pueden ser atributos o elementos.

En ese caso, ¿qué criterios podemos utilizar para decidir si un dato del documento que se pretende estructurar ha de representarse mediante un elemento o un atributo? Aunque no siempre se respetan, podemos usar los siguientes criterios:

- **El dato será un elemento si cumple alguna de las siguientes condiciones:**

- Contiene subestructuras.
- Es de un tamaño considerable.
- Su valor cambia frecuentemente.
- Su valor va a ser mostrado a un usuario o aplicación.

- **Los casos en los que el dato será un atributo son:**

- El dato es de pequeño tamaño y su valor raramente cambia, aunque hay situaciones en las que este caso puede ser un elemento.
- El dato solo puede tener unos cuantos valores fijos.
- El dato guía el procesamiento XML pero no se va a mostrar.

Los espacios de nombres, o namespaces, ¿qué nos permiten?

- Diferenciar entre los elementos y atributos de distintos vocabularios con diferentes significados que comparten nombre.
- Agrupar todos los elementos y atributos relacionados de una aplicación XML para que el software pueda reconocerlos con facilidad.

¿Cómo se declaran?

xmnl: "URI_namespace"

¿Y si se usa un prefijo que nos informe sobre cuál es el vocabulario al que está asociada esa definición?

xmnl:prefijo="URI_namespace"

En ambos casos URI_namespace es la localización del conjunto del vocabulario del espacio de nombres al que se hace referencia.

3.1 Definiciones de tipo de documento, DTD.

Están formadas por una **relación precisa de qué elementos pueden aparecer en un documento y dónde, así como el contenido y los atributos del mismo**. Garantizan que los datos del documento XML cumplen las restricciones que se les haya impuesto en el DTD, ya que estas últimas permiten:

- Especificar la estructura del documento.
- Reflejar una restricción de integridad referencial mínima utilizando (ID e IDREF).
- Utilizar unos pequeños mecanismos de abstracción comparables a las macros, que son las entidades.
- Incluir documentos externos.

¿Cuáles son los inconvenientes de los DTD?

Los principales son:

- Su sintaxis no es XML.
- No soportan espacios de nombres.
- No definen tipos para los datos. Solo hay un tipo de elementos terminales, que son los datos textuales.
- No permite las secuencias no ordenadas.
- No es posible formar claves a partir de varios atributos o elementos.
- Una vez que se define un DTD no es posible añadir nuevos vocabularios.

Cuando están definidas dentro del documento XML se ubican entre corchetes después del nombre del ejemplar en el elemento <!DOCTYPE> pero, cuando está definido en un fichero externo ¿a qué tipo de fichero corresponde? Definimos el DTD externo en un fichero de texto plano con extensión **dtd**.

3.2 Declaraciones de tipos de elementos terminales

Los **tipos terminales** son aquellos elementos que se corresponden con hojas de la estructura de árbol formada por los datos del documento XML asociado al DTD. La declaración de tipos de elementos está formada por la cadena "`<!ELEMENT`">" separada por, al menos un espacio del nombre del elemento XML que se declara, y seguido de la declaración del contenido que puede tener dicho elemento.

En el caso de elementos terminales, es decir, aquellos que no contienen más elementos, esta declaración de contenido es dada por uno de los siguientes valores:



- **EMPTY:** Indica que el elemento no es contenedor. Por ejemplo, la siguiente definición muestra un elemento A que no contiene nada:

```
<!ELEMENT A EMPTY>
```

- **ANY:** Permite que el contenido del elemento sea cualquier cosa. Un ejemplo de definición de un elemento de este tipo es:

```
<!ELEMENT A ANY>
```

- **(#PCDATA):** Indica que los datos son analizados en busca de etiquetas, resultando que el elemento no puede contener elementos, es decir solo puede contener datos de tipo carácter exceptuando los siguientes: <, &,]], >. Si es de este tipo, el elemento A tendrá una definición como:

```
<!ELEMENT A (#PCDATA)>
```

EJEMPLO

Si se tiene la siguiente estructura en un documento XML

```
<alumno>Olga Velarde Cobo</alumno>
```

Un DTD que podría ajustarse sería:

```
<!ELEMENT alumno (#PCDATA)>
```

3.3 Declaraciones de tipos de elementos no terminales

Una vez que sabemos el modo de definir las hojas de un árbol de datos **veamos cómo definir sus ramas**, es decir los elementos que están formados por otros elementos.

Para definirlos utilizamos referencias a los grupos que los componen tal y como muestra el ejemplo:

```
<!ELEMENT A (B, C)>
```

En este caso se ha definido un elemento A que está formado por un elemento B seguido de un elemento C.



¿Y qué sucede cuando un elemento puede aparecer en el documento varias veces, hay que indicarlo de algún modo? Pues sí, también hay que indicar cuando un elemento puede no aparecer. Para ello usamos los siguientes operadores, que nos permiten definir la cardinalidad de un elemento:

- **Operador opción, ?.** Indica que el elemento no es obligatorio. En el siguiente ejemplo el subelemento trabajo es opcional.

```
<!ELEMENT telefono (trabajo?, casa )>
```

- **Operador uno-o-más, +.** Define un componente presente al menos una vez. En el ejemplo definimos un elemento formado por el nombre de una provincia y otro grupo, que puede aparecer una o varias veces.

```
<!ELEMENT provincia (nombre, (cp, ciudad)+ )>
```

- **Operador cero-o-mas, *.** Define un componente presente cero, una o varias veces. En el ejemplo el grupo (cp, ciudad) puede no aparecer o hacerlo varias veces.

```
<!ELEMENT provincia (nombre, (cp, ciudad)* )>
```

- **Operador de elección, |.** Cuando se utiliza sustituyendo las comas en la declaración de grupos indica que para formar el documento XML hay que elegir entre los elementos separados por este operador. En el ejemplo siguiente, el documento XML tendrá elementos provincia que estarán formados por el elemento nombre y el cp (código postal), o por el elemento nombre y la ciudad.

```
<!ELEMENT provincia (nombre, (cp | ciudad) )>
```

EJEMPLO

Si se tiene la siguiente estructura en un documento XML

```
<alumno>
  <nombre>Olga</nombre>
  <dirección>El Percebe 13</dirección>
</alumno>
```

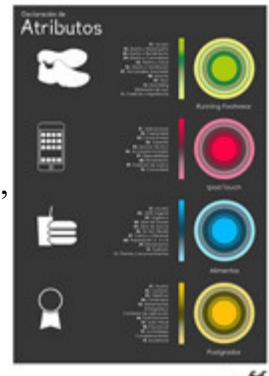
Un DTD que podría ajustarse sería:

```
<!ELEMENT alumno (nombre, apellidos, dirección)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT dirección (#PCDATA))>
```

3.4 Declaraciones de listas de atributos para los tipos de elementos

Ya sabemos cómo declarar elementos, ahora veamos el modo de **declarar los atributos asociados a un elemento**. Para ello utilizamos la cadena `<!ATTLIST` seguida del nombre del elemento asociado al atributo que se declara, luego el nombre de éste último seguido del tipo de atributo y del modificador. Este elemento puede usarse para declarar una lista de atributos asociada a un elemento, o repetirse el número de veces necesario para asociar a dicho elemento esa lista de atributos, pero individualmente.

Al igual que los elementos no todos los atributos son del mismo tipo, los más destacados son:



- Enumeración, es decir, el atributo solo puede tomar uno de los valores determinados dentro de un paréntesis y separados por el operador |.

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

- **CDATA**, se utiliza cuando el atributo es una cadena de texto.
- **ID**, permite declarar un atributo identificador en un elemento. Hay que recordar que este valor ha de ser único en el documento. Además hay que tener en cuenta que los números no son nombres válidos en XML, por tanto no son un identificador legal de XML. Para resolverlo suele incluirse un prefijo en los valores y separarlo con un guión o una letra.
- **IDREF**, permite hacer referencias a identificadores. En este caso el valor del atributo ha de corresponder con el de un identificador de un elemento existente en el documento.
- **NMTOKEN**, permite determinar que el valor de un atributo ha de ser una sola palabra compuesta por los caracteres permitidos por XML.

¿También hemos de declarar si el valor de un atributo es obligatorio o no? Si, para ello se usan los siguientes modificadores:

- **#IMPLIED**, determina que el atributo sobre el que se aplica es opcional.
- **#REQUIRED**, determina que el atributo tiene carácter obligatorio.
- **#FIXED**, permite definir un valor fijo para un atributo independientemente de que ese atributo se defina explícitamente en una instancia del elemento en el documento XML.
- Literal, asigna a un atributo el valor dado por una cadena entre comillas.

EJEMPLO

Si se tiene la siguiente estructura en un documento XML

```
<alumno edad=15>
  <nombre>Olga</nombre>
  <apellidos>Velarde Cobo</apellidos>
  <dirección>El Percebe 13</dirección>
</alumno>
```

Un DTD que podría ajustarse sería:

```
<!ELEMENT alumno (nombre, apellidos, dirección)>
<!ATTLIST alumno edad CDATA #REQUIRED>
<!ELEMENT nombre (#PCDATA)>
```

```
<!ELEMENT apellidos (#PCDATA)>
<!ELEMENT dirección (#PCDATA)>
```

3.5 Declaraciones de entidades

¿Qué sucede si queremos **declarar valores constantes dentro de los documentos?** ¿podemos?

Las entidades nos permiten definir constantes en un documento XML. Cuando se usan dentro del documento XML se limitan por "&" y ";", por ejemplo &entidad;

¿Cómo trabaja el intérprete con ellos? Al procesar el documento XML, el intérprete sustituye la entidad por el valor que se le ha asociado en el DTD.

```
<!ENTITY ber "&this_is_cool">
<!-- I can include other DTDs, too -->
<!ENTITY N SYSTEM "stuff.dtd">
NoteDTD
<!-->
<ENTITY this_is_broken;
<ENTITY that "One is OK: ">
<!-->
<ENTITY complex ">
<!--highlighting keeps stuff
  from working-->
</entity>
I'd like to forward this to a Cheezy XML node.
-->
```

No admiten recursividad, es decir, una entidad no puede hacer referencia a ella misma.

Para definir una entidad en un DTD se usa el elemento `<!ENTITY>`

Las entidades pueden ser de tres tipos:

- **Internas:** Existen cinco entidades predefinidas en el lenguaje, son:
 - `<`: Se corresponde con el signo menor que, <.
 - `>`: Hace referencia al signo mayor que, >.
 - `"`: Son las comillas rectas dobles, ".
 - `'`: Es el apóstrofe o comilla simple, '.
 - `&`: Es el et o ampersand, &.

¿Se puede definir una entidad diferente? ¿Cómo?

Utilizando la siguiente sintaxis:

```
<!ENTITY nombre_entidad "valor de la entidad">
```

Por ejemplo, `<!ENTITY dtd "Definiciones de Tipo de Documento">`

- **Externas:** Permiten establecer una relación entre el documento XML y otro documento a través de la URL de éste último. Un ejemplo de declaración de una entidad externa es:

```
<!ENTITY nombre_entidad SYSTEM "http://localhost/docsxml/fichero_entidad.xml">
```

En este caso el contenido de los ficheros es analizado, por lo que deben seguir la sintaxis XML.

Cuando es necesario incluir ficheros con formatos binarios, es decir ficheros que no se analicen, se utiliza la palabra reservada NDATA en la definición de la entidad y habrá que asociar a dicha entidad una declaración de notación, tal y como muestra el ejemplo del apartado siguiente.

- **De parámetro:** Permite dar nombres a partes de un DTD y hacer referencia a ellas a lo largo del mismo. Son especialmente útiles cuando varios elementos del DTD comparten listas de atributos o especificaciones de contenidos. Se denotan por %entidad;

```
<!ENTITY %direccion "calle, numero?, ciudad, cp">
<!ENTITY alumno (dni, %direccion;)>
<!ENTITY ies (nombre, %direccion;)>
```

- **De parámetro externas:** Permite incluir en un DTD elementos externos, lo que se aplica en dividir la definición DTD en varios documentos.

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

3.6 Declaraciones de notación



Cuando se incluyen ficheros binarios en un fichero, ¿cómo le decimos qué aplicación ha de hacerse cargo de ellos? La respuesta es utilizando notaciones. La sintaxis para declarar **notaciones** es:

```
<!NOTATION nombre SYSTEM aplicacion>
```

Por ejemplo, una notación llamada **gif** donde se indica que se hace referencia a un editor de formatos gif para visualizar imágenes será:

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

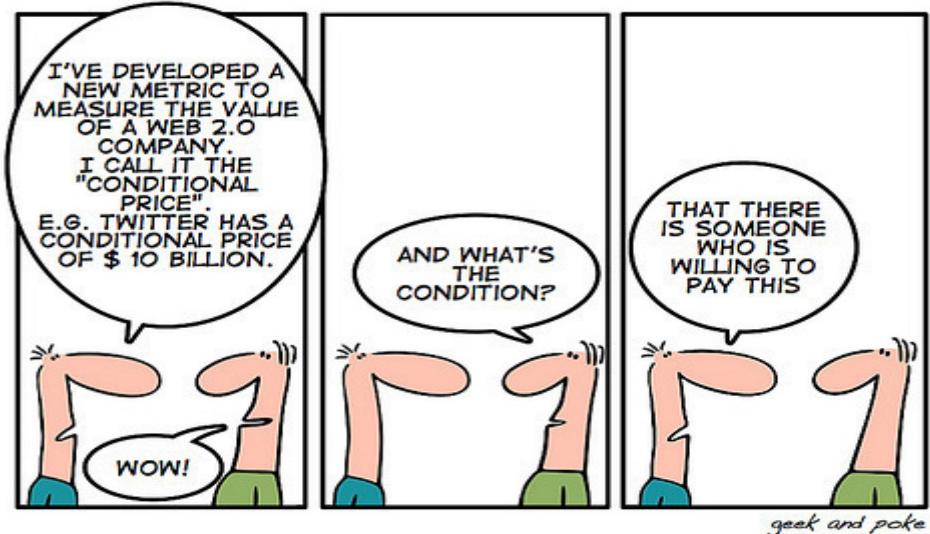
Para asociar una entidad externa no analizada a esta notación basta declarar dicha entidad del siguiente modo:

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

3.7 Secciones condicionales

Permiten incluir o ignorar partes de la declaración de un DTD. Para ello se usan dos tokens:

- **INCLUDE**, permite que se vea esa parte de la declaración del DTD. Su sintaxis es:



```
<![INCLUDE [Declaraciones visibles] ] >
```

Por ejemplo:

```
<![INCLUDE [ <!ELEMENT nombre (#PCDATA)>] ] >
```

- **IGNORE**, permite ocultar esa sección de declaraciones dentro del DTD. La forma de uso es:

```
<![IGNORE [Declaraciones ocultas] ] >
```

Por ejemplo:

```
<![IGNORE [<!ELEMENT clave (#PCDATA)>] ] >
```

4.B. XML Schema

Sitio: Aula Virtual CIERD (CIDEAD)

Curso: Lenguaje de marcas y sistemas de gestión de información

Libro: 4.B. XML Schema

Imprimido por: MANUEL JOSE GONZALEZ CALVO

Día: jueves, 21 de mayo de 2020, 23:15

Tabla de contenidos

- 1 Introducción
- 2 Tipos de datos
 - 2.1 Facetas de los tipos de datos
 - 2.2 Facetas: ejercicios
 - 2.3 Facetas: soluciones a los ejercicios
- 3 Elementos del lenguaje
 - 3.1 Ejemplo de esquema
- 4 Definición de tipos de datos XML Schema
 - 4.1 Definición de tipos de datos: ejercicios
 - 4.2 Definición de tipos de datos: soluciones
- 5 Asociación con documentos XML
 - 5.1 Asociación con documentos: solución
- 6 Documentación del esquema
- 7 Herramientas de creación y validación

1 Introducción

Los **DTD** permiten diseñar un **vocabulario para ficheros XML**, pero, ¿qué sucede cuando los valores de los elementos y atributos de esos ficheros han de corresponder a datos de un tipo determinado, o cumplir determinadas restricciones que no pueden reflejarse en los DTD? Para ello se definen **XML Schemas**.

¿También se definen en ficheros planos? Si, ya que son documentos XML, pero en este caso la extensión de los archivos es xsd, motivo por el cual también se les denomina documentos **XSD**.

Los elementos XML que se utilizan para generar un esquema han de pertenecer al espacio de nombre XML Schema, que es: <http://www.w3.org/2001/XMLSchema>.

El ejemplar de estos ficheros es <**xs:schema**>, contiene declaraciones para todos los elementos y atributos que puedan aparecer en un documento XML asociado válido. Los elementos hijos inmediatos de este ejemplar son <**xs:element**> que nos permiten crear globalmente un elemento. Esto significa que el elemento creado puede ser el ejemplar del documento XML asociado.

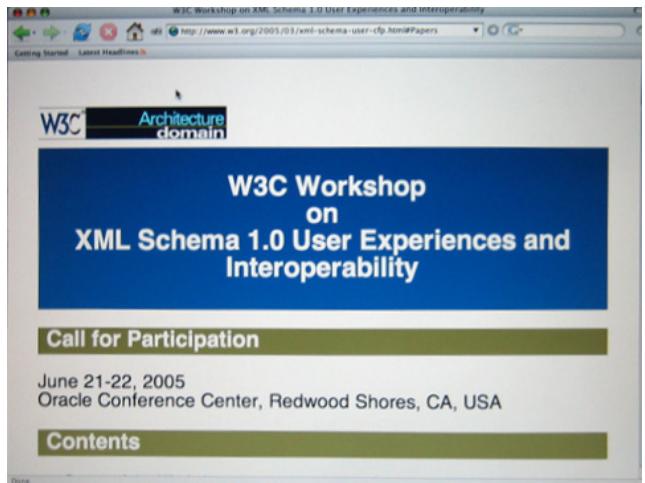
EJEMPLO

Vamos a crear un esquema correspondiente al siguiente documento XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE alumno>
<alumno edad="22">Olga Velarde Cobo</alumno>
```

Un XML schema que podría ajustarse sería:

```
< ?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="alumno" type="xs:string"/>
</xs:schema>
```



2 Tipos de datos

Son los distintos valores que puede tomar el **atributo type** cuando se declara un elemento o un atributo y representan el tipo de dato que tendrá el elemento o atributo asociado a ese **type** en el documento XML.



Algunos de estos valores predefinidos son:

- **string**, se corresponde con una cadena de caracteres UNICODE.
- **boolean**, representa valores lógicos, es decir que solo pueden tomar dos valores, true o false.
- **integer**, número entero positivo o negativo.
- **positiveInteger**, número entero positivo.
- **negativeInteger**, número entero negativo.
- **decimal**, número decimal, por ejemplo, 8,97.
- **dateTime**, representa una fecha y hora absolutas.
- **duration**, representa una duración de tiempo expresado en años, meses, días, horas, minutos segundos. El formato utilizado es: PnYnMnDTnHnMnS. Por ejemplo para representar una duración de 2 años, 4 meses, 3 días, 5 horas, 6 minutos y 10 segundos habría que poner: P2Y4M3DT5H6M7S. Se pueden omitir los valores nulos, luego una duración de 2 años será P2Y. Para indicar una duración negativa se pone un signo – precediendo a la P.
- **time**, hora en el formato hh:mm:ss.
- **date**, fecha en formato CCYY-MM-DD.
- **gYearMonth**, representa un mes de un año determinado mediante el formato CCYY-MM.
- **gYear**, indica un año gregoriano, el formato usado es CCYY.
- **gMonthDay**, representa un día de un mes mediante el formato –MM-DD.
- **gDay**, indica el ordinal del día del mes mediante el formato –DD, es decir el 4º día del mes será –04.
- **gMonth**, representa el mes mediante el formato –MM. Por ejemplo, febrero es –02.
- **anyURI**, representa una URI.
- **language**, representa los identificadores de lenguaje, sus valores están definidos en [RFC 1766](#).
- **ID, IDREF, ENTITY, NOTATION, MTOKEN**. Representan lo mismo que en los DTD's

2.1 Facetas de los tipos de datos

¿Cuáles son las restricciones que podemos aplicar sobre los valores de los datos de un elemento o atributo?

Están definidos por las **facetas**, que solo pueden aplicarse sobre tipos simples utilizando el elemento **xs:restriction**. Se expresan como un elemento dentro de una restricción y se pueden combinar para lograr restringir más el valor del elemento. Son, entre otros:

- **length, minlength, maxlenthg:** Longitud del tipo de datos.
- **enumeration:** Restringe a un determinado conjunto de valores.
- **whitespace:** Define el tratamiento de espacios (preserve/replace, collapse).
- **(max/min)(In/Ex)clusive:** Límites superiores/inferiores del tipo de datos. Cuando son Inclusive el valor que se determine es parte del conjunto de valores válidos para el dato, mientras que cuando se utiliza Exclusive, el valor dado no pertenece al conjunto de valores válidos.
- **totalDigits, fractionDigits:** número de dígitos totales y decimales de un número decimal.
- **pattern:** Permite construir máscaras que han de cumplir los datos de un elemento. La siguiente tabla muestra algunos de los caracteres que tienen un significado especial para la generación de las máscaras.

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.
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.
\t	Tabulación.

2.2 Facetas: ejercicios

- 1. Creación de una cadena de texto con una longitud máxima de 9 caracteres y dos valores posibles.**
- 2. Creación de un elemento en el que se respetan los espacios tal y como se han introducido.**
- 3. Creación de un elemento calificaciones de dos dígitos cuyo valor es un número entero comprendido entre 1 y 10, ambos inclusive.**
- 4. Creación de la máscara de un DNI mediante pattern.**

2.3 Facetas: soluciones a los ejercicios

1. Creación de una cadena de texto con una longitud máxima de 9 caracteres y dos valores posibles.

```
<xs:simpleType name="estado">
  <xs:restriction base="xs:string">
    <xs:maxLength value="9"/>
    <xs:enumeration value="conectado"/>
    <xs:enumeration value="ocupado"/>
  </xs:restriction>
</xs:simpleType>
```

2. Creación de un elemento en el que se respetan los espacios tal y como se han introducido.

```
<xs:simpleType name="nombre">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="preserve"/>
  </xs:restriction>
</xs:simpleType>
```

3. Creación de un elemento calificaciones de dos dígitos cuyo valor es un número entero comprendido entre 1 y 10, ambos inclusive.

```
<xs:simpleType name="calificaciones">
  <xs:restriction base="xs:integer">
    <xs:totalDigits value="2"/>
    <xs:minExclusive value="0"/>
    <xs:maxInclusive value="10"/>
  </xs:restriction>
</xs:simpleType>
```

4. Creación de la máscara de un DNI mediante pattern.

```
<xs:simpleType name="dni">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [A-Z]"/>
  </xs:restriction>
</xs:simpleType>
```

3 Elementos del lenguaje

A continuación se listan los más usados:

- Esquema, **xs:schema**, contiene la definición del esquema.
- Tipos complejos, **xs:complexType**, define tipos complejos.
- Tipos simples, **xs:simpleType**, permite definir un tipo simple restringiendo sus valores.
- Restricciones, **xs:restriction**, permite establecer una restricción sobre un elemento de tipo base.
- Agrupaciones, **xs:group**, permite nombrar agrupaciones de elementos y de atributos para hacer referencia a ellas.
- Secuencias, **xs:sequence**, permite construir elementos complejos mediante la enumeración de los que les forman.
- Alternativa, **xs:choice**, representa alternativas, hay que tener en cuenta que es una o-exclusiva.
- Contenido mixto, definido dando valor true al atributo mixed del elemento **xs:complexType**, permite mezclar texto con elementos.
- Secuencias no ordenadas, **xs:all**, representa a todos los elementos en cualquier orden.

3.1 Ejemplo de esquema

Ejemplo de esquema correspondiente a un documento XML para estructurar la información personal sobre los alumnos de un centro educativo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xss:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- elemento raíz -->
    <xss:element name="alumnos" type="datosAlum"/>
    <!-- Definicion del tipo datosAlum -->
    <xss:complexType name="datosAlum">
        <xss:sequence>
            <xss:element name="alumno" type="datos" minOccurs="1" maxOccurs="unbounded"/>
        </xss:sequence>
    </xss:complexType>

    <!-- Definicion del tipo datos -->
    <xss:complexType name="datos">
        <xss:sequence>
            <xss:element name="nombre" type="xs:string" minOccurs="1" maxOccurs="1"/>
            <xss:element name="apellidos" type="xs:string" minOccurs="1" maxOccurs="1"/>
            <xss:element name="direccion" type="datosDireccion" minOccurs="1" maxOccurs="1"/>
            <xss:element name="contactar" type="datosContactar" minOccurs="1" maxOccurs="1"/>
        </xss:sequence>
        <!-- Atributos del elemento usuario -->
        <xss:attribute name="id" type="xs:string"/>
    </xss:complexType>
    <xss:complexType name="datosDireccion">
        <xss:sequence>
            <xss:element name="domicilio" type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xss:element name="codigo_postal" minOccurs="0" maxOccurs="1" >
                <xss:complexType>
                    <xss:attribute name="cp" type="xsd:string"/>
                </xss:complexType>
            </xss:element>
            <xss:element name="localidad" type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xss:element name="provincia" type="xs:string" minOccurs="0" maxOccurs="1"/>
        </xss:sequence>
    </xss:complexType>
    <xss:complexType name="datosContactar">
        <xss:sequence>
            <xss:element name="telf_casa" type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xss:element name="telf_movil" type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xss:element name="telf_trabajo" type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xss:element name="email" minOccurs="0" maxOccurs="unbounded" >
                <xss:complexType>
                    <xss:attribute name="href" type="xs:string"/>
                </xss:complexType>
            </xss:element>
        </xss:sequence>
    </xss:complexType>
</xss:schema>
]
```

4 Definición de tipos de datos XML Schema

En los DTD se diferencia entre los elementos terminales y los no terminales ¿en este caso también?

Si, este lenguaje **permite trabajar tanto con datos simples como con estructuras de datos complejos**, es decir, compuestos por el anidamiento de otros datos simples o compuestos.

- **Tipos de datos simples.** Estos datos se suelen definir para hacer una restricción sobre un tipo de datos XDS ya definido y establece el rango de valores que puede tomar. También se pueden crear tipos de datos simples basados en listas de valores utilizando el atributo derivedBy de simpleType.
- **Tipos de datos compuestos.** El elemento xsd:complexType permite definir estructuras complejas de datos. Su contenido son las declaraciones de elementos y atributos, o referencias a elementos y atributos declarados de forma global. Para determinar el orden en que estos elementos aparecen en el documento XML se utiliza el propio elemento.



4.1 Definición de tipos de datos: ejercicios

- 1. Creación de un elemento simple de nombre edad que representa la edad de un alumno de la ESO, por tanto su rango está entre los 12 y los 18 años.**
- 2. Creación de una lista con los días de la semana en letras.**
- 3. Creación de un elemento compuesto de nombre alumno, formado por los elementos nombre, apellidos, web personal.**

4.2 Definición de tipos de datos: soluciones

="">1. Creación de un elemento simple de nombre edad que representa la edad de un alumno de la ESO, por tanto su rango está entre los 12 y los 18 años.

```
<xmlelement name="edad">
<xssimpleType>
<xstriction base="xs:positiveInteger">
<xsmiInclusive value="12"/>
<xsmiInclusive value="18"/>
</xstriction>
</xssimpleType>
</xmlelement>
```

2. Creación de una lista con los días de la semana en letras.

```
<xssimpleType name="dia_semana" base="xs:string" derivedBy="list">
<diasemana>Lunes Martes Miercoles Jueves Viernes Sabado Domingo</diasemana>
</xssimpleType>
```

3. Creación de un elemento compuesto de nombre alumno, formado por los elementos nombre, apellidos, web personal.

```
<xsccomplexType name="alumno">
<xssecuence>
<xselement name="nombre" type="xs:string" minOccurs="1" maxOccurs="1"/>
<xselement name="apellidos" type="xs:string" minOccurs="1" maxOccurs="1"/>
<xselement name="web" type="xs:string" minOccurs="0" maxOccurs="5">
<xsccomplexType>
<xseattribute name="href" type="xs:string"/>
</xsccomplexType>
</xselement>
<xssecuence>
</xsccomplexType>
```

5 Asociación con documentos XML



Una vez que tenemos creado el **fichero XSD** ¿cómo lo **asociamos a un fichero XML**?

El modo de asociar un esquema a un documento XML es un espacio de nombres al ejemplar del documento, donde se indica la ruta de localización de los ficheros esquema mediante su URI, precedida del prefijo "**xsi:**".

EJERCICIO

Dado el ejemplo de esquema de la sección 3.1. construye un documento XML que cumpla las especificaciones definidas en el archivo XML Schema.

5.1 Asociación con documentos: solución

EJERCICIO

Dado el ejemplo de esquema de la sección 3.1. construye un documento XML que cumpla las especificaciones definidas en el archivo XML Schema.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<alumnos xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:SchemaLocation="file:/D:/ubicación/del archivo/alumnos.xsd">
    <alumno>
        <nOMBRE>Jose Ramón</nOMBRE>
        <apellidos>García González</apellidos>
        <direccion>
            <domicilio>El Pez, 12</domicilio>
            <codigo_postal>85620</codigo_postal>
            <localidad>Suances</localidad>
            <provincia>Cantabria</provincia>
        </direccion>
        <contactar>
            <telf._casa>985623165</telf._casa>
            <telf._movil>611233544</telf._movil>
            <telf._trabajo>965847536</telf._trabajo>
            <email>pepito@educadistancia.com</email>
        </contactar>
    </alumno>
    <alumno>
        <nOMBRE>Carlos</nOMBRE>
        <apellidos>López Pérez</apellidos>
        <direccion>
            <domicilio>El Cangrejo, 25</domicilio>
            <codigo_postal>86290</codigo_postal>
            <localidad>Santillana</localidad>
            <provincia>Cantabria</provincia>
        </direccion>
        <contactar>
            <telf._casa>931132565</telf._casa>
            <telf._movil>623863544</telf._movil>
            <telf._trabajo>984657536</telf._trabajo>
            <email>carlos@educadistancia.com</email>
        </contactar>
    </alumno>
</alumnos>
```

6 Documentación del esquema

Una vez que hemos visto como crear un esquema vamos a ver el modo de incorporar cierta documentación (quién es el autor, limitaciones de derechos de autor, utilidad del esquema, etc.) al mismo.

Podemos pensar que un método para añadir esta información es utilizar comentarios. El problema es que los analizadores no garantizan que los comentarios no se modifiquen al procesar los documentos y por tanto, que los datos añadidos no se pierdan en algún proceso de transformación del documento.

En lugar de usar los comentarios, XML Schema tiene definido un elemento **xs:annotation** que permite **guardar información adicional**. Este elemento a su vez puede contener una combinación de otros dos que son:

- **xs:documentation**, además de contener elementos de esquema puede contener elementos XML bien estructurados.
También permite determinar el idioma del documento mediante el atributo `xml:lang`.
- **xs:appinfo**, se diferencia muy poco del elemento anterior, aunque lo que se pretendió inicialmente era que `xs:documentation` fuese legible para los usuarios y que `xs:appinfo` guardase información para los programas de software.
También es usado para generar una ayuda contextual para cada elemento declarado en el esquema.



EJEMPLO

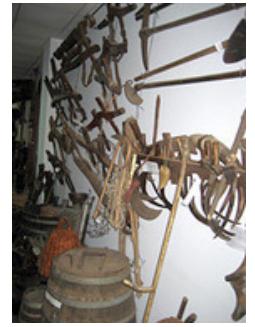
Ejemplo de documentación de un esquema.

```
<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema>
  <xs:annotation>
    <xs:documentation xml:lang ="es-es">
      Materiales para formación e-Learning
      <modulo>Lenguajes de marcas y sistemas de gestión de información.</modulo>
      <fecha_creación> 2011</fecha_creación>
      <autor> Nuky La Bruji</autor>
    </xs:documentation>
  </xs:annotation>
  <xs:element name="lmsgi" type="xs:string">
    <xs:annotation>
      <xs:appinfo>
        <texto_de_ayuda>Se debe de introducir el nombre completo del tema</texto_de_ayuda>
      </xs:appinfo>
    </xs:annotation>
  </xs:element>
</xs:schema>
```

7 Herramientas de creación y validación

Igual que hasta ahora, para crear y validar los documentos XML y los esquemas basta con un editor de texto plano y un navegador. ¿Pero no hay ninguna herramienta que nos facilite el trabajo? Pues sí, existen aplicaciones que permiten al usuario visualizar, validar y editar documentos en el lenguaje XML. Algunos de estos productos son:

- Editix XML Editor (Versiones open source y comercial).
- Microsoft Core XML Services (MSXML) (Gratis).
- XMLFox (freeware).
- Altova XML Spy Edición Estándar (comercial).
- Editor XML xmlBlueprint. (comercial)
- Stylus Studio 2001 (comercial).
- Oxygen XML Editor (comercial).
- Exchanger XML Editor (comercial)
- XML copy editor (open source).



5.A. Conversión y adaptación de documentos XML

Sitio: Aula Virtual CIERD (CIDEAD)
Curso: Lenguaje de marcas y sistemas de gestión de información
Libro: 5.A. Conversión y adaptación de documentos XML
Imprimido por: MANUEL JOSE GONZALEZ CALVO
Día: jueves, 21 de mayo de 2020, 23:16

Tabla de contenidos

- 1 Introducción
- 2 Estructura básica de una hoja XSLT
- 3 Elementos XSLT
- 4 XPath
 - 4.1 Términos básicos
 - 4.2 Expresiones
 - 4.3 Procesando expresiones
 - 4.4 Ruta de Localización
 - 4.5 Funciones de XPath
 - 4.6 Predicados
 - 4.7 Ejemplo - Ejercicio
 - 4.8 Ejemplo - Soluciones
 - 4.9 Acceso a datos de otro documento XML
- 5 Utilización de plantillas
 - 5.1 Plantillas: ejercicio
 - 5.2 Plantillas: solución
- 6 Procesadores XSLT
- 7 Depuradores XSLT

1 Introducción

Los documentos XML son documentos de texto con etiquetas, que contienen exclusivamente información, sin entrar en detalles de formato.

Por eso, si queremos usar directamente los datos (para leer, imprimir, etc.) es necesario transformar primero el documento XML.

Los navegadores, por ejemplo, interpretan las etiquetas del documento XML, les aplican un formato según lo especificado en las hojas CSS y lo muestran al usuario.

Es posible transformar un documento XML en otro tipo de documento. A esto se le denomina transformación de documentos. Algunas tecnologías que entran en juego en la transformación de documentos son:

- **XSLT**: permite definir el modo de transformar un documento XML en otro.
- **XSL-FO**: permite transformar un documento XML en otro documento de un formato legible e imprimible por una persona, por ejemplo en un documento **PDF**.
- **Xpath**: permite acceder a los diversos componentes de un documento XML.

Hoy en día se usa masivamente **XSLT**, que es ya un estándar aprobado por el **W3C**. Los documentos XSLT se denominan **hojas XSLT**.

XSLT es uno de los lenguajes derivados de **XML**, por tanto las hojas **XSLT** también son documentos **XML** (al igual que sucede con los canales **RSS**, atom o los documentos **XSD**).

¿Qué transformaciones podemos realizar sobre un documento **XML** usando **XSLT**? Podemos generar:

- Otro documento **XML**.
- Un documento **HTML**.
- Un documento de texto.

2 Estructura básica de una hoja XSLT

¿Cuáles son los elementos contenidos en una hoja XSLT?



Básicamente hay tres tipos de elementos:

- **Elementos XSLT**, están precedidos del prefijo xsl:, pertenecen al espacio de nombres xsl, están definidos en el estándar del lenguaje y son interpretados por cualquier procesador XSLT.
- **Elementos LRE**, no pertenecen a XSLT, sino que se repiten en la salida sin más.
- **Elementos de extensión**, al igual que los anteriores, no pertenecen al espacio de nombres xsl. Son manejados por implementaciones concretas del procesador. Estos normalmente no son usados.

Y, ¿cómo indicamos que un documento XML está asociado a una hoja XSLT?

Hay que incluir en el documento XML, después del prólogo, una línea como la que sigue:

```
<?xmlstylesheet type="text/xsl" href="ruta_del_fichero_xsl.xsl"?>
```

3 Elementos XSLT

 Hoja en la que se observan unas tortugas colocadas una encima de otra, en y ordenadas por tamaños estando abajo la más grande.

El elemento raíz de una hoja XSLT es **xsl:stylesheet** o **xsl:transform**, que son prácticamente equivalentes. Sus atributos principales son:

- **version**, cuyo valor puede ser 1.0 o 2.0.
- **xmlns:xsl**, se utiliza para declarar el espacio de nombres xsl. Para XSLT suele ser la dirección:

<http://www.w3.org/1999/XSL/Transform>

A los elementos hijos de estos se les conoce como elementos de nivel superior, son estructuras contenedoras de instrucciones. Dado que si son hijos directos no pueden anidarse, excepto **xsl:variable** y **xsl:param**. Los más destacados son:

- **xsl:attribute**, añade un atributo a un elemento en el árbol de resultados.
- **xsl:choose**, permite decidir que parte de la hoja XSL se va a procesar en función de varios resultados.
- **xsl:decimal-format**, define un patrón que permite convertir en cadenas de texto números en coma flotante.
- **xsl:for-each**, aplican sentencias a cada uno de los nodos del árbol que recibe como argumento.
- **xsl:if**, permite decidir si se va a procesar o no una parte del documento XSL en función de una condición
- **xsl:import**, importa una hoja de estilos XSLT localizada en una URI dada.
- **xsl:key**, define una o varias claves que pueden ser referenciadas desde cualquier lugar del documento.
- **xsl:output**, define el tipo de salida que se generará como resultado.
- **xsl:preserve-space**, especifica cuales son los elementos del documento XML que no tienen espacios en blanco eliminados antes de la transformación.
- **xsl:sort** permite aplicar un template a una serie de nodos ordenándolos alfabético numéricamente.
- **xsl:strip-space**, especifica cuáles son los elementos del documento XML que tienen espacios en blanco eliminados antes de la transformación.
- **xsl:template**, es el bloque fundamental de una hoja XSLT, por lo que veremos su descripción en el apartado siguiente.
- **xsl:value-of**, calcula el valor de una expresión XPath dada y lo inserta en el árbol de resultados del documento de salida.
- **xsl:variable**, asigna un valor a una etiqueta para usarlo cómodamente.

4 XPath

XPath es un estándar (diferente de XML) aprobado por el W3C, que permite navegar entre los elementos y atributos de un documento XML.

Para hacerlo, se basa en las relaciones de parentesco entre los nodos del documento.

Inicialmente se creó para utilizarlo con XLST, pero en la actualidad se utiliza también con XML Schema, Xquery, Xlink, Xpointer, Xforms, etc.

Expresiones de camino

XPath se usa definiendo expresiones de camino, para seleccionar nodos o conjuntos de nodos en un documento XML.

Esas expresiones se parecen mucho a las expresiones de camino (path) que se suelen usar en los sistemas de ficheros.

Estas expresiones se aplican a un documento XML, asumiendo que su estructura interna es la de un árbol. Al aplicar una expresión, se obtiene un conjunto de nodos (que puede ser vacío).

En el siguiente código XML de ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<matriculas>
  <alumno>
    <nombre>Pedro</nombre>
    <apellido1>López</apellido1>
    <apellido2>Ortega</apellido2>
    <DNI pais="es">López</DNI>
  </alumno>
</matriculas>
```

Al evaluar la expresión "/matriculas/alumno/nombre" podríamos obtener los nombres de los alumnos matriculados.

4.1 Términos básicos

 Dos circunferencias de distinto color y tamaño formadas por círculos concéntricos que pueden usarse como bases para soportar cualquier objeto.

Nodos

Un documento XML es tratado en XPath como un árbol de nodos. Hay 7 tipos de nodos: elemento, atributo, texto, espacio de nombres, instrucción de proceso, comentario y documento. El elemento más alto del árbol es el nodo raíz. Explicamos más sobre algunos de ellos:

- **Nodo raíz**, es el nodo que contiene al ejemplar del fichero XML.
 - ¡Cuidado! No lo confundamos con el elemento raíz del documento, ya que éste último está por debajo de él.
- **Nodos elemento**, son cada uno de los elementos del documento XML.
 - Tienen un elemento padre.
 - El padre del elemento raíz, es decir del ejemplar, es el nodo raíz del documento.
 - Pueden tener identificadores únicos, lo que permite referenciarlos de forma mucho más directa. Para ello es necesario que el atributo esté definido en un **DTD** o un fichero **XSD** asociado.
- **Nodos texto**, son aquellos caracteres del documento que no están marcados con ninguna etiqueta.
 - No tienen hijos.
- **Nodos atributo**, son los atributos de un elemento.
 - Se consideran etiquetas añadidas al nodo elemento.
 - No se consideran hijos de ese elemento.
 - Aquellos atributos que tengan un valor asignado en el esquema asociado, se tratarán como si ese valor se le hubiese dado al escribir el documento XML.
 - Para las definiciones de los espacios de nombre y para aquellos atributos que se han definido con la propiedad **#IMPLIED** en su **DTD** no se crean nodos.

En el ejemplo de antes:

```
<?xml version="1.0" encoding="UTF-8"?>
<matriculas>
  <alumno>
    <nombre>Pedro</nombre>
    <apellido1>López</apellido1>
    <apellido2>Ortega</apellido2>
    <DNI pais="es">López</DNI>
  </alumno>
</matriculas>
  <matrículas> es el nodo elemento raíz
  <nombre>Pedro</nombre> es un nodo elemento
  pais="es" es un nodo atributo
```

Ítems

Los ítems pueden ser nodos o valores atómicos.

En el ejemplo, valores atómicos serían "Pedro", o "es"

Relaciones entre nodos

Según como si sitúen los nodos dentro del árbol, se habla de las relaciones entre ellos. Así se dice que hay relaciones padre-hijo, hermano, ascendentes y descendentes.

En ejemplo anterior, algunas de estas relaciones son:

- <nombre> es hijo de <alumno>.
- <nombre> y <DNI> son hermanos.
- <matriculas> es un ascendiente de <nombre>
- <apellido1> es descendiente de <matriculas>

4.2 Expresiones

Como hemos dicho, XPath usa expresiones, que  Caricaturas de caras con distintas expresiones. serán evaluadas.

¿Y cuáles son los resultados que da la evaluación de una expresión Xpath?

Pues podemos obtener cuatro tipos de resultados diferentes:

- **Un conjunto de nodos (node-set)**
 - No está ordenado.
 - Se considera que todos los elementos de un conjunto de nodos son hermanos, independientemente de lo que fuesen originalmente.
 - Aunque los hijos de los nodos que forman un conjunto de nodos son accesibles, los subárboles de un nodo no se consideran elementos del conjunto.
- **Un valor booleano.**
- **Un número.**
- **Una cadena.**

¿Qué elementos podemos utilizar en una expresión XPath?

Podemos utilizar :

- Agrupaciones: “()”, “{ }”, “[]”.
- Elemento actual, elemento padre.
- Atributos: “@”.
- Elementos, “*”.
- Separadores, “::”.
- Comas, “,”.
- El nombre de un elemento.
- Tipo de nodo, que puede ser:
 - comment.
 - text.
 - processing instruction.
 - node.
- Operadores: and, or, mod, div, *, /, //, |, +, -, =, !=, <, >, <=, >=.
- Nombres de función.
- Denominación de ejes: ancestor, ancestor-or-self-attribute, child, descendant, descendant-or-self, following, following-sibling, namespace, parent, preceding, preceding-sibling, self.
- Literales, se ponen entre comillas dobles o simples. Pueden anidarse alternando el tipo de comillas.
- Números.
- Referencias a variables, para lo que se utiliza la sintaxis: **\$nombreVariable**

Para ver cómo se emplean cada uno de ellos, puedes encontrar muy buenos ejemplos aquí.

4.3 Procesando expresiones

Cuando estamos procesando un documento XML de entrada, podemos hablar de los siguientes conceptos:

- **Nodo actual**, es aquél en el que se encuentra el procesador.
- **Nodo contexto**, cada expresión está formada por subexpresiones que se van evaluando antes de resolver la siguiente. El conjunto de nodos obtenido tras evaluar una expresión y que se utiliza para evaluar la siguiente es el nuevo contexto.
- **Tamaño del contexto**, es el número de nodos que se están evaluando en un momento dado en una expresión Xpath.

4.4 Ruta de Localización

La **ruta de localización de un nodo** es la ruta que hay que seguir en un árbol de datos para localizar ese nodo. Las rutas de localización se evalúan y esa evaluación **siempre devuelve un conjunto de nodos**, aunque puede estar vacío.

¿Cómo podemos crear una ruta de localización? Mediante la unión de varios pasos de localización.  Autopista.

Las rutas de localización básicas son:

- **La ruta de localización del nodo raíz del documento.** Es la barra diagonal “/”. Se trata de una ruta absoluta, porque siempre significa lo mismo, independientemente de la posición del procesador en el documento de entrada al aplicar la regla.
- **Localización de un elemento,** selecciona todos los hijos del nodo de contexto con el nombre especificado. Los elementos a los que se refiera dependerán de la localización del nodo de contexto, por lo que es una ruta relativa. En el caso de que el nodo contexto no tenga ningún nodo hijo con esa denominación, el valor de la ruta de localización será un elemento vacío.
- **Localización de atributos,** para referirnos a ellos en XPath, se utiliza el símbolo “@” seguido del nombre del atributo deseado.
- **Localización de espacios de nombres,** no se tratan explícitamente.
- **Localización de comentarios.**
- **Localización de nodos de texto.**
- **Localización de instrucciones de procesamiento.**

¿Cómo podemos comparar distintos elementos y tipos de nodo simultáneamente? Utilizando alguno de los tres comodines mostrados a continuación:

- **Asterisco “*”:** compara cualquier nodo de elemento, independientemente de su nombre. No compara ni atributos ni comentarios, nodos de texto o instrucciones de procesamiento.
- **Node():** compara, además de los tipos de elementos, el nodo raíz, los nodos de texto, los de instrucción de procesamiento, los nodos de espacio de nombre, los de atributos y los de comentarios.
- **“@*”:** compara los nodos de atributo.

4.5 Funciones de XPath

 Primer plano de la punta de un portaminas encima de una hoja donde se ve escrita la función matemática $f(z)$.

XPath también nos proporciona funciones.

Podemos emplearlas, unidas a las rutas de localización, para hacer cosas sobre los conjuntos de elementos que devuelven los predicados (que veremos enseguida).

Entre las funciones más importantes que podemos utilizar en Xpath destacan:

- **boolean()**, al aplicarla sobre un conjunto de nodos devuelve true si no es vacío.
- **not()**, al aplicarla sobre un predicado devuelve true si el predicado es falso , y falso si el predicado es true.
- **true()**, devuelve el valor true.
- **false()**, devuelve el valor false.
- **count()**, devuelve el número de nodos que forman un conjunto de nodos.
- **name()**, devuelve un nombre de un nodo.
- **local-name()**, devuelve el nombre del nodo actual o del primer nodo de un conjunto de nodos.
- **namespace-uri()**, devuelve el URI del nodo actual o del primer nodo de un conjunto dado.
- **position()**, devuelve la posición de un nodo en su contexto comenzando en 1. Por ejemplo, para seleccionar los dos primeros elementos de tipo elemento de un fichero XML pondremos: **//elemento[position()<=2]**
- **last()**, Devuelve el último elemento del conjunto dado.
- **normalize-space()**, permite normalizar los espacios de una cadena de texto, es decir, si se introduce una cadena donde hay varios espacios consecutivos, esta función lo sustituye por uno solo.
- **string()**, es una función que convierte un objeto en una cadena. Los valores numéricos se convierten en la cadena que los representa teniendo en cuenta que los positivos pierden el signo. Los valores booleanos se convierten en la cadena que representa su valor, esto es “true” o “false”.
- **concat()**, devuelve dos cadenas de texto concatenadas. El ejemplo siguiente devuelve “Xpath permite obtener datos de un fichero XML”.

```
concat('XPath', 'permite obtener datos de un fichero XML')
```

- **string-length()**, devuelve la cantidad de caracteres que forman una cadena de caracteres.
- **sum()**, devuelve la suma de los valores numéricos de cada nodo en un conjunto de nodos determinado.

4.6 Predicados

Un **predicado** es una expresión booleana que añade un nivel de verificación al paso de localización.



Camisa de publicidad del grupo musical “The Predicates”.

En estas expresiones podemos incorporar funciones XPath.

¿Qué ventajas nos da el uso de predicados?

Mediante las rutas de localización se pueden seleccionar varios nodos a la vez, pero el uso de predicados permite seleccionar un nodo que cumple ciertas características.

Los predicados se incluyen dentro de una ruta de localización utilizando los corchetes, por ejemplo:

```
/receta/ingredientes/ingrediente[@codigo="1"]/nombre
```

En este caso se está indicando al intérprete que escoja, dentro de un fichero XML de recetas, el nombre del ingrediente cuyo código tiene el valor “1”.

Veamos que es lo que podemos incluir en un predicado.

- **Ejes**, permiten seleccionar el subárbol dentro del nodo contexto que cumple un patrón. Pueden ser o no de contenido.
 - **child**, es el eje utilizado por defecto. Su forma habitual es la barra, “/”, aunque también puede ponerse: /child::
 - **attribute**, permite seleccionar los atributos que deseemos. Es el único eje que veremos que no es de contenido.
 - **descendant**, permite seleccionar todos los nodos que descienden del conjunto de nodos contextos. Se corresponde con la doble barra, //, aunque se puede usar: descendant::
 - **self**, se refiere al nodo contexto y se corresponde con el punto ”.”.
 - **parent**, selecciona los nodos padre, para referirnos a él usamos los dos puntos, “..”.
 - **ancestor**, devuelve todos los nodos de los que el nodo contexto es descendiente.
- Nodos test, permiten restringir lo que devuelve una expresión Xpath. Podemos agruparlos en función de los ejes a los que se puede aplicar.
 - Aplicable a cualquier eje:
 - “*”, solo devuelve elementos, atributos o espacios de nombres pero no permite obtener nodos de texto, o comentarios de cualquier tipo.
 - **node()**, devuelve los nodos de todos los tipos.
 - Solo aplicables a ejes de contenido:
 - **text()**, devuelve cualquier nodo de tipo texto.
 - **comment()**, devuelve cualquier nodo de tipo comentario.
 - **processing-instruction()**, devuelven cualquier tipo de instrucción de proceso.

Varios predicados pueden unirse mediante los operadores lógicos **and**, **or** o **not**.

4.7 Ejemplo - Ejercicio

Dado el siguiente fichero XML

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>

<!DOCTYPE agenda>
<agenda>
<propietario>
<identificadores>
<nOMBRE>Alma</nOMBRE>
<apellidos>López Terán</apellidos>
</identificadores>
<direccion>
<calle>El Percebe 13, 6F</calle>
<localidad>Torrelavega</localidad>
<cp>39300</cp>
</direccion>
<telefonos>
<movil>970898765</movil>
<casa>942124567</casa>
<trabajo>628983456</trabajo>
</telefonos>
</propietario>
<contactos>
<persona id="p01">
<identificadores>
<nOMBRE>Inés</nOMBRE>
<apellidos>López Pérez</apellidos>
</identificadores>
<direccion>
<calle>El Ranchito 24, 6B</calle>
<localidad>Santander</localidad>
<cp>39006</cp>
</direccion>
<telefonos>
<movil>970123123</movil>
</telefonos>
</persona>
<persona id="p02">
<identificadores>
<nOMBRE>Roberto</nOMBRE>
<apellidos>Gutiérrez Gómez</apellidos>
</identificadores>
<direccion>
<calle>El Marranito 4, 2F</calle>
<localidad>Santander</localidad>
<cp>39004</cp>
</direccion>
<telefonos>
<movil>970987456</movil>
<casa>942333323</casa>
```

```
</telefonos>
</persona>
<persona id="p03">
<identificadores>
<nOMBRE>Juan</nOMBRE>
<apellidos>Sánchez Martínez</apellidos>
</identificadores>
<direccion>
<calle>El Cangrejo 10, sn</calle>
<localidad>Torrelavega</localidad>
<cp>39300</cp>
</direccion>
<telefonos>
<movil>997564343</movil>
<casa>942987974</casa>
<trabajo>677899234</trabajo>
</telefonos>
</persona>
</contactos>
</agenda>
```

Construir las sentencias Xpath que permitan obtener los siguientes datos:

1. Nombre del propietario de la agenda.
2. Teléfono de casa del propietario.
3. Nombres y apellidos de los contactos de la agenda.
4. Nombre e identificador de cada contacto.
5. Datos del contacto con identificador "p02".
6. Identificadores de los contactos que tienen móvil.

Te facilitamos el fichero XML en el siguiente enlace para tu comodidad en la resolución del ejercicio:

Fichero XML

4.8 Ejemplo - Soluciones

- Nombre del propietario de la agenda.

```
/agenda/propietario/identificadores/nombre
```

- Teléfono de casa del propietario.

```
/agenda/propietario/telefonos/casa
```

- Nombres y apellidos de los contactos de la agenda.

```
//contactos/persona/identificadores/nombre | //contactos/persona/identificadores/apellidos
```

- Nombre e identificador de cada contacto.

```
//contactos/persona/identificadores/nombre | //contactos/persona/@id
```

- Datos del contacto con identificador "p02".

```
//contactos/persona[@id="p02"]/*/*
```

- Identificadores de los contactos que tienen teléfono en casa.

```
//contactos/persona/telefonos/casa/../../@id
```

4.9 Acceso a datos de otro documento XML

 Cartel promocional de la exposición 'elemets', en el que se ven imágenes que simbolizan la madera, el fuego la tierra el metal y el agua.

Además de acceder a los datos del fichero XML con el que se está trabajando directamente, es útil poder acceder a los datos de otros ficheros.

Para ello utilizaremos la función **document()**, pero dicha función **NO** es del lenguaje XPath, sino que pertenece a XSLT.

Esta función puede admitir dos argumentos diferentes:

document(URI)

En este caso la función devuelve el elemento raíz del documento XML que se localiza en el URI especificado.

document(nodo)

En esta ocasión lo que devuelve la función, es el conjunto de nodos cuya raíz es el nodo dado.

5 Utilización de plantillas

El elemento **xml:template**



El elemento **xsl:template** permite controlar el formato de salida que se aplica a ciertos datos de entrada.

Tiene un atributo denominado **match**, que se utiliza para seleccionar los nodos del árbol de entrada (conforme a XPath) a los que se va a aplicar la plantilla.

Para especificar el formato de salida, se emplean sentencias XHTML (el contenido del elemento template ha de ser XHTML bien estructurado).

Para especificar el punto de la salida donde queremos que se apliquen las plantillas, se emplea el elemento **xsl:apply-templates**.

Tiene un atributo **select**, que se utiliza para seleccionar los hijos del nodo de entrada (conforme a XPath) a los que se va a aplicar la plantilla.

Podemos usar ese atributo para especificar el orden en que van a ser procesados los nodos hijo. Si no lo hacemos de este modo, se aplicarán en el orden es el que el interprete utiliza al leer el documento XML, es decir, de arriba abajo.

Ejemplo de uso

Usando el *código* XML visto en el apartado 5.A 4.7, podríamos usar el siguiente código:

```
<xsl:template match="/">
<html>
<body>
<h2>Agenda de </h2>
<xsl:apply-templates/>
</body>
</html>
</xsl:template>

<xsl:template match="propietario/identificadores">
<h3>
<xsl:apply-templates select="apellidos"/>
,
<xsl:apply-templates select="nombre"/>
</h3>
</xsl:template>
```

Generaríamos una salida formateada como la siguiente:

Agenda de

López Terán , Alma

El Percebe 13, 6F Torrelavega 39300 970898765 942124567 628983456 Inés López Pérez El Ranchito 24, 6B Santander 39006
970123123 Roberto Gutiérrez Gómez El Marranito 4, 2F Santander 39004 970987456 942333323 Juan Sánchez Martínez El
Cangrejo 10, sn Torrelavega 39300 997564343 942987974 677899234

Con la primera plantilla creamos el documento html y el body y ponemos el título principal.

Con la segunda, seleccionamos al propietario de la agenda y lo formateamos como encabezado 3.

5.1 Plantillas: ejercicio

Dada la siguiente plantilla XSLT (disponible en formato fichero en este enlace):

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">

<xsl:template match="identificadores">
<xsl:value-of select="nombre"/>,
<xsl:value-of select="apellidos"/>
</xsl:template>

<xsl:template match="persona">
<xsl:apply-templates select="identificadores"/></xsl:apply-templates>
</xsl:template>
</xsl:stylesheet>
```

Que se desea aplicar sobre el siguiente fichero (puedes descargarlo haciendo clic aquí):

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<!DOCTYPE agenda>
<?xml-stylesheet type="text/xsl" href=".//LMSGI_CONT_Ejemplo02.xsl"?>
<agenda>
<persona id="p01">
<identificadores>
<nombre>Inés</nombre>
<apellidos>López Pérez</apellidos>
</identificadores>
<direccion>
<calle>El Ranchito 24, 6B</calle>
<localidad>Santander</localidad>
<cp>39006</cp>
</direccion>

<telefonos>
<movil>970123123</movil>
</telefonos>
</persona>

<persona id="p02">
<identificadores>
<nombre>Roberto</nombre>
<apellidos>Gutiérrez Gómez</apellidos>
</identificadores>

<direccion>
<calle>El Marranito 4, 2F</calle>
<localidad>Santander</localidad>
<cp>39004</cp>
</direccion>
```

```
<telefonos>
<movil>970987456</movil>
<casa>942333323</casa>
</telefonos>
</persona>
</agenda>
```

Indicar cuál sería el **resultado de aplicar la transformación XSLT sobre el fichero XML**.

5.2 Plantillas: solución

La salida que obtenemos es:

```
<?xml version="1.0" encoding="utf-8"?>  
[  
Inés,  
López Pérez  
  
Roberto,  
Gutiérrez Gómez  
  
Juan,  
Sánchez Martínez  
]  
[  
]  
]
```

6 Procesadores XSLT

Un procesador **XSLT** es un software que lee un documento **XSLT** y otro Imagen de un procesador XML, y crea un documento de salida aplicando las instrucciones de la hoja de estilos **XSLT** a la información del documento **XML**.

Pueden estar integrados dentro de un explorador Web, en un servidor web, o puede ser un programa que se ejecuta desde la línea de comandos.

Existen diferentes modos de realizar la transformación **XSLT**

- Mediante el procesador **MSXML**, (servicios principales de Microsoft XML).
- Usando un procesador **XSLTPROC**, por ejemplo xsltproc desde línea de comandos.
- Invocando a la biblioteca de transformación desde un programa.
- Realizando un enlace entre la hoja **XSLT** y el documento **XML**, en este caso hay que añadir, en el fichero **XML** entre el la definición de la versión **XML** y la definición del tipo de documento, la línea:

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

El fichero puede verse directamente desde cualquier navegador que soporte **XSLT**, aunque tiene la desventaja de que queda ligado a esa vista.

La mayoría de editores **XML** permiten escoger el interprete que debe encargarse de procesar un documento **XSLT**.

7 Depuradores XSLT

Los depuradores son elementos de software que permiten seguir la generación de un documento, a partir de los datos de un documento **XML** aplicándoles una hoja de estilo **XSLT**.

Los depuradores **XSLT** nos facilitan la localización y corrección de errores dejándonos ejecutar el código paso a paso, salir, ejecutar el código hasta el cursor, ejecutar hasta el final, pausar y detener.

Algunos editores de **XML** (sobre todo los comerciales), incluyen un depurador que permite visualizar simultáneamente la plantilla que se está ejecutando y sobre qué datos del documento **XML** actúan y cuál es la salida que genera dicha orden.

De este modo es más fácil averiguar cuáles son las plantillas que dan lugar al formato que deseamos en la salida del documento.

6.A. Almacenamiento de la información

Sitio: Aula Virtual CIERD (CIDEAD)
Curso: Lenguaje de marcas y sistemas de gestión de información
Libro: 6.A. Almacenamiento de la información
Imprimido por: MANUEL JOSE GONZALEZ CALVO
Día: jueves, 21 de mayo de 2020, 23:23

Tabla de contenidos

1 Utilización de XML para el almacenamiento de la información

1.1 Ámbitos de aplicación

2 Sistemas de almacenamiento de la información

3 XML y BD Relacionales

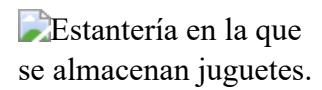
3.1 De DB Relacional a XML

4 XML y BD Orientadas a Objetos

5 BD XML Nativas

1 Utilización de XML para el almacenamiento de la información

Como ya hemos visto, **XML**, es un estándar potente y de amplia aceptación para **guardar y comunicar información acerca de objetos**. Permite la codificación de información, separada de la forma en la que se debe presentar al usuario. Cuando se desea encontrar un fragmento específico de información en los contenidos de un nodo o atributo **XML**, es necesario procesar completamente todo el archivo **XML**.



Estantería en la que se almacenan juguetes.

Podemos pensar en XML como en una base de datos. Visto así, una **base de datos XML** se puede ver como una colección de **documentos XML**. Cada documento XML representa un registro de la base de datos; es un archivo en el sistema de archivos y contiene una cadena válida **XML**.

La estructura de un **documento XML** suele seguir un mismo **esquema XML**, aunque no es necesario que sea así. Este es uno de los beneficios de las bases de datos **XML**, ya que cada archivo se puede configurar de forma estructurada por lo que es independiente, pero fácilmente accesible.

La principal ventaja de usar bases de datos XML es que proporcionan una gran flexibilidad (gracias a tener colecciones de documentos con un esquema independiente), lo que conlleva facilidad a la hora de crear aplicaciones que usen a la Base de Datos XML.

Esta flexibilidad es un gran valor, especialmente en los últimos años, en los que se ha visto la necesidad de contar con estándares para el intercambio de información. Estos estándares ayudan a que las organizaciones puedan compartir su información de una manera más cómoda, automática y eficiente.

1.1 Ámbitos de aplicación

Los documentos y los requerimientos de almacenamiento de datos XML pueden ser agrupados en dos categorías generales:

- **Sistemas centrados en los datos.** Cuando los documentos XML tienen una estructura bien definida y contienen datos que pueden ser actualizados y usados de diversos modos. Es apropiada para ítems como contenidos de periódicos, artículos, publicidad, facturas, órdenes de compra... y algunos documentos menos estructurados.
- **Sistemas centrados en los documentos.** Cuando los documentos tienden a ser más impredecibles en tamaño y contenido. Presentan más tipos de datos, de tamaño más variable, con reglas flexibles para campos opcionales y para el propio contenido.

 Ordenador portátil con un listado de archivos de una base de datos.

Los sistemas de almacenamiento XML deben acomodarse eficientemente con ambos tipos de requerimientos de datos, dado que XML está siendo usado en sistemas que administran ambos tipos de datos.

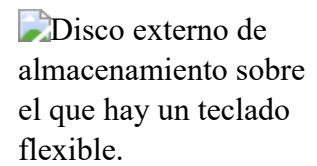
La mayoría de los productos se enfocan en servir uno de esos formatos de datos mejor que el otro. Las bases de datos relacionales tradicionales son mejores para tratar con requerimientos centrados en los datos, mientras que los sistemas de administración de contenido y de documentos, suelen ser mejores para almacenar datos centrados en el documento.

Los sistemas de bases de datos deben ser capaces de exponer los datos relacionales como un documento XML y almacenar un documento XML recibido como datos relacionales para transferir, obtener y almacenar los datos.

2 Sistemas de almacenamiento de la información

En lo que concierne a las bases de datos, **XML** permite integrar sistemas de información hasta ahora separados:

- **Sistemas de información basados en documentos** (ficheros), tienen estructura irregular, utilizan tipos de datos relativamente simples y dan gran importancia al orden.
- **Sistemas de información estructurados** (bases de datos relacionales), son relativamente planos, utilizan tipos de datos relativamente complejos y dan poca importancia al orden.



Podemos establecer las siguientes semejanzas entre una base de datos y un fichero **XML** con su esquema asociado:

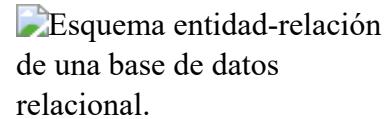
- La tecnología **XML** usa uno o más documentos para almacenar la información.
- Define esquemas sobre la información.
- Tiene lenguajes de consulta específicos para recuperar la información requerida.
- Dispone de **APIs (SAX, DOM)**.

Pero aparecen muchas más cosas que lo diferencian. Debido a que no es una base de datos, la tecnología **XML** carece, entre otras cosas, tanto de almacenamiento y actualización eficientes como de índices, seguridad, transacciones, integridad de datos, acceso concurrente, disparadores, etc.; que son algunas de las características habituales en las bases de datos. Por tanto es imposible pensar que **XML** se vaya a utilizar para las tareas transaccionales de una organización para las cuales sigue estando sobradamente más justificado utilizar una base de datos.

Un ejemplo de esto es **JABX (Java Architecture for XML Binding)** del que puedes obtener más información en el siguiente enlace.

3 XML y BD Relacionales

Las Bases de Datos Relacionales se basan en las relaciones (tablas bidimensionales), como único medio para representar los datos del mundo real. Están asociadas al lenguaje estándar **SQL**.



Se han creado complejas teorías y patrones para encajar objetos o estructuras jerarquizadas en bases de datos relacionales. Existen numerosos **middlewares** encargados de la transferencia de información entre estructuras **XML** y bases de datos relacionales.

Las Bases de Datos Relacionales suponen una posibilidad para el almacenamiento de datos **XML**. Sin embargo, no están bien preparadas para almacenar estructuras de tipo jerárquico como son los documentos **XML**, algunas de las causas son:

- Las bases de datos relacionales tienen una estructura regular frente al carácter heterogéneo de los documentos **XML**.
- Los documentos **XML** suelen contener muchos niveles de anidamiento mientras que los datos relacionales son planos.
- Los documentos **XML** tienen un orden intrínseco mientras que los datos relacionales son no ordenados.
- Los datos relacionales son generalmente densos (cada columna tiene un valor), mientras que los datos **XML** son dispersos, es decir, pueden representar la carencia de información mediante la ausencia del elemento.

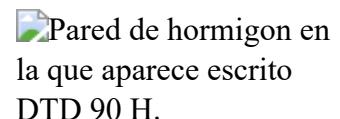
Algunas de las razones para usar los tipos de Bases de Datos Relacionales y los productos de bases de datos existentes para almacenar **XML**, aún cuando no sea de forma nativa son:

- Las bases de datos relacionales y orientadas a objetos son bien conocidas, mientras que las bases de datos **XML** nativas son nuevas.
- Como resultado de la familiaridad con las bases de datos relacionales y orientadas a objetos, los usuarios se inclinan a ellas especialmente por el rendimiento.

3.1 De DB Relacional a XML

El proceso de traducción puede ser descompuesto en los siguientes pasos básicos:

- **Crear el esquema XML** con un elemento para cada tabla y los atributos correspondientes para cada columna no clave. Las columnas que no permiten valores nulos pueden ser marcadas como requeridas, mientras que aquellas que permiten valores nulos pueden ser marcadas como opcionales en el esquema **XML**. Las columnas pueden ser también anidadas como elementos, pero pueden surgir problemas cuando el mismo nombre de columna es usado en más de una tabla. Por ello, lo más simple es transformar las columnas como atributos **XML**, donde las colisiones de nombre en el esquema **XML** no son un problema.
- **Crear las claves primarias en el esquema XML.** Una solución podría ser agregar un atributo para la columna clave primaria, con un ID agregado al nombre de la columna. Este atributo podría necesitar ser definido en el esquema **XML** como de tipo ID. Pueden surgir problemas de colisión al crear claves primarias en el esquema **XML**, ya que a diferencia de las bases de datos relacionales, donde las claves primarias necesitan ser únicas sólo dentro de una tabla, un atributo ID dentro de un documento **XML** debe ser único a través de todo el documento. Para resolverlo se puede agregar el nombre del elemento (nombre de la tabla), al valor de la clave primaria (valor del atributo). Esto asegura que el valor es único a través del documento **XML**.
- **Establecer las relaciones de clave migrada.** Esto se puede lograr mediante el anidamiento de elementos bajo el elemento padre, un ID de esquema **XML** puede ser usado para apuntar a una estructura **XML** correspondiente conteniendo un **IDREF**.



Pueden existir muchas variaciones de esquemas **XML** para representar la misma base de datos relacional.

4 XML y BD Orientadas a Objetos

Las **bases de datos orientadas a objetos** (DBOO) soportan un modelo Colección de diversos de objetos puro, en el sentido de que no están basados en extensiones de otros modelos más clásicos como el relacional:

- Están influenciados por los lenguajes de programación orientados a objetos.
- Pueden verse como un intento de añadir la funcionalidad de un SGBD a un lenguaje de programación.
- Son una alternativa para el almacenamiento y gestión de documentos XML.

Colección de diversos objetos de cocina, se puede observar en primer plano: dos espumaderas, una tijera y un abrelatas, en segundo plano podemos ver un sacacorchos.

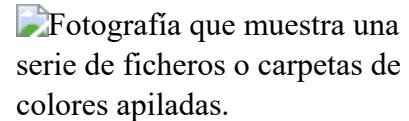
Componentes del estándar de Orientación a Objetos:

- **Modelo de Objetos.** Está concebido para proporcionar un modelo de objetos estándar para las bases de datos orientadas a objetos. Es el modelo en el que se basan los demás componentes.
- **Lenguajes de Especificación de Objetos (ODL).** Para definir los objetos.
- **Lenguaje de Consulta de Objetos (OQL).** Para realizar consultas contra los objetos.
- **'Bindings' para C++, Java y Smalltalk.** Definen un Lenguaje de Manipulación de Objetos (OML) que extiende el lenguaje de programación para soportar objetos persistentes. Además incluyen soporte para OQL, navegación y transacciones.

Una vez transformado el documento XML en objetos, éstos son gestionados directamente por el SGBDOO. Dicha información se consulta acudiendo al lenguaje de consulta OQL. Los mecanismos de indexación, optimización, procesamiento de consultas, etc. son los del propio SGBDOO, y por lo general, no son específicos para el modelo XML.

5 BD XML Nativas

Las Bases de Datos XML Nativas son bases de datos (y como tales soportan transacciones, acceso multi-usuario, lenguajes de consulta, etc) diseñadas especialmente para almacenar documentos XML.



Las BD XML Nativas se caracterizan principalmente por:

- **Almacenamiento de documentos en colecciones.** Las colecciones juegan en las bases de datos nativas el papel de las tablas en las BD relacionales.
- **Validación de los documentos.**
- **Consultas.** La mayoría de las BD XML Nativas soportan uno o más lenguajes de consulta. Uno de los más populares es XQuery.
- **Indexación XML.** Se ha de permitir la creación de índices que aceleren las consultas realizadas.
- **Creación de identificadores únicos.** A cada documento XML se le asocia un identificador único.
- **Actualizaciones y Borrados.**

Según el tipo de almacenamiento utilizado pueden dividirse en dos grupos:

- **Almacenamiento Basado en Texto.** Almacena el documento XML entero en forma de texto y proporciona alguna funcionalidad de base de datos para acceder a él. Hay dos posibilidades:
 - **Posibilidad 1:** Almacenar el documento como un **BLOB** en una base de datos relacional, mediante un fichero, y proporcionar algunos índices sobre el documento que aceleren el acceso a la información.
 - **Posibilidad 2:** Almacenar el documento en un almacén adecuado con índices, soporte para transacciones, etc.
- **Almacenamiento Basado en el Modelo.** Almacena un modelo binario del documento (por ejemplo, DOM) en un almacén existente o bien específico.
 - **Posibilidad 1:** Traducir el DOM a tablas relacionales como Elementos, Atributos, Entidades, etc.
 - **Posibilidad 2:** Traducir el DOM a objetos en una **BDOO**.
 - **Posibilidad 3:** Utilizar un almacén creado especialmente para esta finalidad.

6.B. XQuery

Sitio: Aula Virtual CIERD (CIDEAD)
Curso: Lenguaje de marcas y sistemas de gestión de información
Libro: 6.B. XQuery
Imprimido por: MANUEL JOSE GONZALEZ CALVO
Día: jueves, 21 de mayo de 2020, 23:24

Tabla de contenidos

- 1 Introducción
- 2 Aplicaciones
- 3 Modelo de datos
- 4 Expresiones
- 5 Claúsulas
- 6 Ejemplos XQuery
- 7 Funciones
- 8 Ejemplo: definición y llamada a una función
- 9 Operadores

1 Introducción

XQuery es un lenguaje diseñado para escribir consultas sobre colecciones de datos expresadas en XML. Puede aplicarse tanto a archivos XML, como a bases de datos relacionales con funciones de conversión de registros a XML. Su principal función es extraer información de un conjunto de datos organizados como un árbol de etiquetas XML. En este sentido **XQuery** es independiente del origen de los datos.

Permite la construcción de expresiones complejas combinando expresiones simples de una manera muy flexible.

De manera general podemos decir que **XQuery es a XML lo mismo que SQL es a las bases de datos relacionales**. Al igual que éste último, XQuery es un lenguaje funcional.

Los **requerimientos técnicos** más importantes de **XQuery** se detallan a continuación:

- Debe ser un **lenguaje declarativo**.
- Debe ser **independiente del protocolo de acceso** a la colección de datos. Esto significa que una consulta en XQuery, debe funcionar igual al consultar un archivo local, que al consultar un servidor de bases de datos, o que al consultar un archivo XML en un servidor web.
- Las consultas y los resultados deben respetar el **modelo de datos XML**.
- Las consultas y los resultados deben ofrecer **soporte para los namespaces**.
- Debe soportar **XML-Schemas y DTDs** y también debe ser capaz de trabajar sin ellos.
- Ha de ser **independiente de la estructura** del documento, esto es, funcionar sin conocerla.
- Debe soportar **tipos simples**, como enteros y cadenas, y **tipos complejos**, como un nodo compuesto.
- Las consultas deben soportar **cuantificadores universales** (para todo) y existenciales (existe).
- Las consultas deben soportar **operaciones sobre jerarquías de nodos** y secuencias de nodos.
- Debe ser posible **combinar información de múltiples fuentes** en una consulta.
- Las consultas deben ser capaces de **manipular los datos independientemente del origen** de estos.
- **El lenguaje de consulta debe ser independiente de la sintaxis**, esto es, pueden existir varias sintaxis distintas para expresar una misma consulta en XQuery.

 **Escenario que tiene un atril forrado de carteles en los que puede leerse "Society of Query", en castellano significa "La sociedad de la consulta".**

2 Aplicaciones

Una vez que hemos visto la definición del lenguaje y sus principales requerimientos, queda pensar, ¿para qué se utiliza?

Sus principales aplicaciones se resumen en tres:

- **Recuperar información a partir de conjuntos de datos XML.**
- **Transformar unas estructuras de datos XML** en otras estructuras que organizan la información de forma diferente.
- **Ofrecer una alternativa a XSLT** para realizar transformaciones de datos en XML a otro tipo de representaciones, como HTML o PDF.

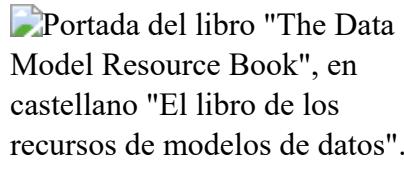
¿Y cuáles son los motores XQuery de código abierto más relevantes y sus características principales?

- **BaseX**: proyecto open-source, con interfaz gráfica y disponible para Linux, Windows y Mac.
- **Qexo**: escrito en Java y con licencia GPL que se distribuye integrado dentro del paquete Kawa.
- **Saxon**: escrito en Java y distribuido en múltiples paquetes, algunos open source y otros bajo licencia comercial.

 Pantalla de ordenador en la que aparecen iconos de las diversas aplicaciones que hay instaladas.

3 Modelo de datos

Aunque **XQuery** y **SQL** puedan considerarse similares, el modelo de datos sobre el que se sustenta **XQuery** es muy distinto del modelo de datos relacional sobre el que sustenta **SQL**, ya que **XML** incluye conceptos como jerarquía y orden de los datos que no están presentes en el modelo relacional.



Por ejemplo, a diferencia de **SQL**, en **XQuery** el orden en que se encuentren los datos es importante, ya que no es lo mismo buscar una etiqueta “****” dentro de una etiqueta “**<A>**” que todas las etiquetas “****” del documento (que pueden estar anidadas dentro de una etiqueta “**<A>**” o no).

La entrada y la salida de una consulta **XQuery** se define en términos de un modelo de datos. Dicho modelo de datos de la consulta proporciona una representación abstracta de uno o más documentos **XML** (o fragmentos de documentos).

Las principales características de este modelo de datos son:

- Se basa en la **definición de secuencia**, como una colección ordenada de cero o más ítems. Éstas pueden ser heterogéneas, es decir pueden contener varios tipos de nodos y valores atómicos. Sin embargo, una secuencia nunca puede ser un ítem de otra secuencia.
- **Orden del documento**: corresponde al orden en que los nodos aparecerían si la jerarquía de nodos fuese representada en formato **XML**, (si el primer carácter de un nodo ocurre antes que el primer carácter de otro nodo, lo precederá también en el orden del documento).
- Contempla un **valor especial llamado “error value”** que es el resultado de evaluar una expresión que contiene un error.

4 Expresiones

Una consulta en XQuery es una expresión que lee una secuencia de datos en XML, y devuelve como resultado otra secuencia de datos en XML.

El valor de una expresión es una secuencia heterogénea de nodos y valores atómicos.

La mayoría de las expresiones están compuestas por la combinación de expresiones más simples unidas mediante operadores y palabras reservadas.

Ya hemos visto que **Xpath** es un lenguaje declarativo para la localización de nodos y fragmentos de información en árboles **XML**.

Puesto que XQuery ha sido construido sobre la base de Xpath y realiza la selección de información y la iteración a través del conjunto de datos basándose en dicho lenguaje, **toda expresión XPath también es una consulta Xquery válida.**

Los **comentarios** en XQuery están limitados entre caras sonrientes, es decir: (: Esto es un comentario XQuery 😊).

En un documento XQuery **los caracteres { } delimitan las expresiones que son evaluadas** para crear un documento nuevo.

XQuery admite expresiones condicionales del tipo **if-then-else** con la misma semántica que tienen en los lenguajes de programación habituales.

Las consultas XQuery pueden estar formadas por hasta cinco tipos de cláusulas diferentes, siguen la norma **FLWOR** (que se pronuncia "flower"). Estas cláusulas son los bloques principales del XQuery, equivalen a las cláusulas **select, from, where, group by, having, order by y limit** de SQL.

En una sentencia **FLWOR al menos ha de existir una cláusula FOR o una LET**, el resto, si existen, han de respetar escrupulosamente el orden dado por el nombre, FLWOR.

Con estas sentencias se consigue buena parte de la funcionalidad que diferencia a XQuery de XPath. Entre otras cosas permite construir el documento que será la salida de la sentencia.

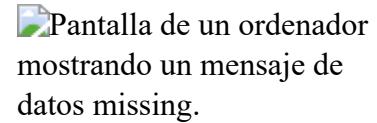
Una consulta **XQuery** está formada por dos partes:

- **Prólogo:** Lugar donde se declaran los espacios de nombres, de funciones, variables, etc.
- **Expresión:** Consulta propiamente dicha.

 Primer plano de un niño con expresión traviesa y sacando la lengua.

5 Claúsulas

Hemos visto el modo de crear sentencias FLWOR, vamos ahora a estudiar aisladamente cada una de las cláusulas que pueden formar estas sentencias.



- **FOR:** asocia una o más variables con cada nodo que encuentre en la colección de datos. Si en la consulta aparece más de una cláusula FOR (o más de una variable en una cláusula FOR), el resultado es el producto cartesiano de dichas variables.
- **LET:** vincula las variables al resultado de una expresión. Si esta cláusula aparece en una sentencia en la que ya hay al menos una cláusula FOR, los valores de la variable vinculada por la cláusula LET se añaden a cada una de las tuplas generadas por la cláusula FOR.
- **WHERE:** filtra tuplas producidas por las cláusulas FOR y LET, quedando solo aquellas que cumplen con la condición.
- **ORDER BY:** ordena las tuplas generadas por FOR y LET después de que han sido filtradas por la cláusula WHERE. Por defecto el orden es ascendente, pero se puede usar el modificador **descending** para cambiar el sentido del orden.
- **RETURN:** construye el resultado de la expresión FLWOR para una tupla dada.

6 Ejemplos XQuery

Vamos a ver algunos ejemplos de XQuery utilizando el fichero libros.xml. En este fichero:

- El elemento raíz es **biblioteca**. Contiene un elemento **libros**.
- Dentro de **libros**, hay varios elementos **libro**.
- Los elementos **libro** tienen atributos **publicacion** y **edicion** (opcional). También tienen elementos **titulo**, **autor** (puede haber más de uno), **editorial**, **paginas** y un elemento opcional para indicar si hay edición electrónica, **edicionElectronica**.

1. Título y editorial de todos los libros

Para devolver varios campos, los envolvemos en un elemento

```
for $x in doc("libros.xml")/biblioteca/libros/libro
return <libro>{$x/titulo, $x/editorial}</libro>
```

```
<libro>
  <titulo>Learning XML</titulo>
  <editorial>O'Reilly</editorial>
</libro>
<libro>
  <titulo>XML Imprescindible</titulo>
  <editorial>O'Reilly</editorial>
</libro>
<libro>
  <titulo>XML Schema</titulo>
  <editorial>O'Reilly</editorial>
</libro>
<libro>
  <titulo>XPath Essentials</titulo>
  <editorial>Wiley</editorial>
</libro>
<libro>
  <titulo>Beginning XSLT 2.0: From Novice to Professional</titulo>
  <editorial>Apress</editorial>
</libro>
<libro>
  <titulo>XQuery</titulo>
  <editorial>O'Reilly</editorial>
</libro>
```

2. El título (sin etiquetas) de todos los libros de menos de 100 páginas.

Para hacer comparaciones con números, lo mejor es convertir los datos con la función **number** para evitar problemas de tipo de dato o que los compare como cadenas.

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where number($x/paginas) < 100
return data($x/titulo)
```

3. El número de libros de menos de 100 páginas.

Utilizamos la función **count()**

```
for $x in doc("libros.xml")/biblioteca/libros
let $y := $x/libro[number(paginas) < 100]
return count($y)
```

1

4. Una lista HTML con el título de los libros de la editorial “O'Reilly” ordenados por título.

Podemos mezclar etiquetas HTML y XQuery y obtener HTML como resultado de una consulta.

```
<ul>
{
for $x in doc("libros.xml")/biblioteca/libros/libro
where $x/editorial = "O'Reilly"
order by $x/titulo
return <li>{data($x/titulo)}</li>
}
</ul>
```

```
<ul>
<li>Learning XML</li>
<li>XML Imprescindible</li>
<li>XML Schema</li>
<li>XQuery</li>
</ul>
```

5. Título y editorial de los libros de 2002

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where $x[@publicacion=2002]
return <libro>{$x/titulo, $x/editorial}</libro>
```

```
<libro>
<titulo>XML Schema</titulo>
<editorial>O'Reilly</editorial>
</libro>
<libro>
<titulo>XPath Essentials</titulo>
<editorial>Wiley</editorial>
</libro>
```

6. Título y editorial de los libros con más de un autor.

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where count($x/autor)>1
return <libro>{$x/titulo, $x/editorial}</libro>
```

```
<libro>
    <titulo>XML Imprescindible</titulo>
    <editorial>O'Reilly</editorial>
</libro>
```

7. Título y editorial de los libros que tienen versión electrónica.

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where $x/versionElectronica
return <libro>{$x/titulo, $x/editorial}</libro>
```

```
<libro>
    <titulo>Learning XML</titulo>
    <editorial>O'Reilly</editorial>
</libro>
<libro>
    <titulo>XPath Essentials</titulo>
    <editorial>Wiley</editorial>
</libro>
```

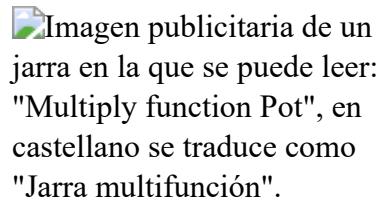
8. Título de los libros que no tienen versión electrónica.

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where not($x/versionElectronica)
return $x/titulo
```

```
<titulo>XML Imprescindible</titulo>
<titulo>XML Schema</titulo>
<titulo>Beginning XSLT 2.0: From Novice to Professional</titulo>
<titulo>XQuery</titulo>
```

7 Funciones

Ahora que conocemos las cláusulas que sustentan el lenguaje **XQuery**, vamos a ocuparnos de las **funciones que soporta**. Estas son funciones matemáticas, de cadenas, para el tratamiento de expresiones regulares, comparaciones de fechas y horas, manipulación de nodos XML, manipulación de secuencias, comprobación y conversión de tipos y lógica booleana. Además permite definir funciones propias y funciones dependientes del entorno de ejecución del motor XQuery. Las funciones más importantes se muestran a continuación:



- **Funciones numéricas**
 - **floor()**, que devuelve el valor numérico inferior más próximo al dado.
 - **ceiling()**, que devuelve el valor numérico superior más próximo al dado.
 - **round()**, que redondea el valor dado al más próximo.
 - **count()**, determina el número de ítems en una colección.
 - **min()** o **max()**, devuelven respectivamente el mínimo y el máximo de los valores de los nodos dados.
 - **avg()**, calcula el valor medio de los valores dados.
 - **sum()**, calcula la suma total de una cantidad de ítems dados.
- **Funciones de cadenas de texto**
 - **concat()**, devuelve una cadena construida por la unión de dos cadenas dadas.
 - **string-length()**, devuelve la cantidad de caracteres que forman una cadena.
 - **startswith()**, **ends-with()**, determinan si una cadena dada comienza o termina, respectivamente, con otra cadena dada.
 - **upper-case()**, **lower-case()**, devuelve la cadena dada en mayúsculas o minúsculas respectivamente.
- **Funciones de uso general**
 - **empty()**, devuelve “**true**” cuando la secuencia dada no contiene ningún elemento.
 - **exists()**, devuelve “**true**” cuando una secuencia contiene, al menos, un elemento.
 - **distinct-values()**, extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados.
 - **data()**, devuelve el valor de los elementos que recibe como argumentos, es decir, sin etiquetas.
- **Cuantificadores existenciales:**
 - **some**, **every**, permiten definir consultas que devuelven algún, o todos los elementos, que verifiquen la condición dada.

Ejemplo de la función data

Esta consulta devuelve todos los títulos, incluyendo las etiquetas

```
for $x in doc("libros.xml")/biblioteca/libros/libro/titulo  
return $x
```

El resultado con el fichero de ejemplo sería

```
<titulo>Learning XML</titulo>
<titulo>XML Imprescindible</titulo>
<titulo>XML Schema</titulo>
<titulo>XPath Essentials</titulo>
<titulo>Beginning XSLT 2.0: From Novice to Professional</titulo>
<titulo>XQuery</titulo>
```

Utilizando la función data

```
for $x in doc("libros.xml")/biblioteca/libros/libro/titulo
return data($x)
```

se obtiene

```
Learning XML
XML Imprescindible
XML Schema
XPath Essentials
Beginning XSLT 2.0: From Novice to Professional
XQuery
```

Además de estas funciones que están definidas en el lenguaje, XQuery permite al desarrollador construir sus propias funciones:

```
declare nombre_funcion($param1 as tipo_dato1, $param2 as tipo_dato2,... $paramN as tipo_datoN)
as tipo_dato_devuelto
{
...CÓDIGO DE LA FUNCIÓN...
}
```

8 Ejemplo: definición y llamada a una función

En el siguiente ejemplo puedes ver la definición y llamada a una función escrita por el usuario que nos calcula el precio de un libro una vez que se le ha aplicado el descuento.

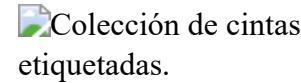
```
declare function minPrice($p as xs:decimal?, $d as xs:decimal?) as xs:decimal?
{
    let $disc := ($p * $d) div 100
    return ($p - $disc)
}
```

Un ejemplo de cómo invocar a la función desde la consulta es:

```
<minPrice>{minPrice($libros/precio,$libros/descuento)}</minPrice>
```

9 Operadores

Veamos ahora algunos de los operadores más importantes agrupados según su funcionalidad:



- **Comparación de valores:** Comparan dos valores escalares y produce un error si alguno de los operandos es una secuencia de longitud mayor de 1. Estos operadores son:
 - **eq**, igual.
 - **ne**, no igual.
 - **lt**, menor que.
 - **le**, menor o igual que.
 - **gt**, mayor que.
 - **ge**, mayor o igual que.
- **Comparación generales:** Permiten comparar operandos que sean secuencias.
 - **=**, igual.
 - **!=**, distinto.
 - **>**, mayor que.
 - **>=**, mayor o igual que.
 - **<**, menor que.
 - **<=**, menor o igual que.
- **Comparación de nodos:** Comparan la identidad de dos nodos.
 - **is**, devuelve true si las dos variables que actúan de operandos están ligadas al mismo nodo.
 - **is not**, devuelve true si las dos variables no están ligadas al mismo nodo.
- **Comparación de órdenes de los nodos:** <<, compara la posición de dos nodos. Devuelve “**true**” si el nodo ligado al primer operando ocurre primero en el orden del documento que el nodo ligado al segundo.
- **Lógicos:** **and** y **or** Se emplean para combinar condiciones lógicas dentro de un predicado.
- **Secuencias de nodos:** Devuelven secuencias de nodos en el orden del documento y eliminan duplicados de las secuencias resultado.
 - **Unión**, devuelve una secuencia que contiene todos los nodos que aparecen en alguno de los dos operandos que recibe.
 - **Intersect**, devuelve una secuencia que contiene todos los nodos que aparecen en los dos operandos que recibe.
 - **Except**, devuelve una secuencia que contiene todos los nodos que aparecen en el primer operando que recibe y que no aparecen en el segundo.
- **Aritméticos:** **+**, **-**, *****, **div** y **mod**, devuelven respectivamente la suma, diferencia, producto, cociente y resto de operar dos números dados.

7.A. Sistemas de gestión empresarial

1 ERP y CRM

ERP (Sistemas de Gestión Empresarial)

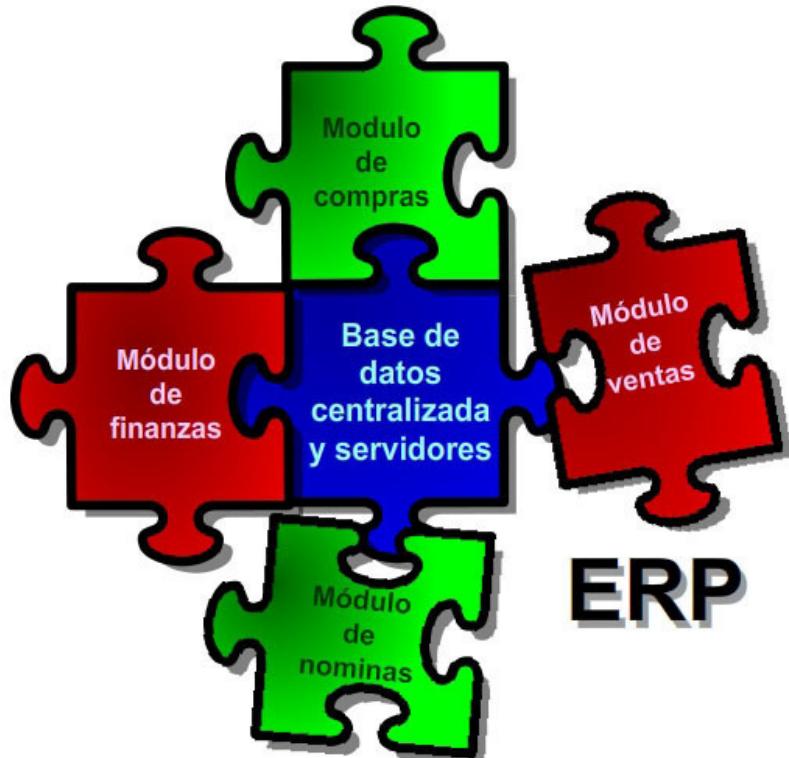
Los ERP son sistemas de gestión de información que se caracterizan por estar compuestos por varios módulos. Cada uno de estos módulos proporciona una unidad de gestión y funcionalidad diferente y específica, por ejemplo: producción, ventas, compras, pedidos, nóminas, etc.

Los objetivos principales de los ERP son:

- **Optimizar** los procesos empresariales.
- **Acceder** a la información confiable y precisa.
- **Permitir** compartir información entre los componentes de la organización.
- **Eliminar** los datos y operaciones innecesarias.

Las características que distinguen a un ERP de cualquier otro software empresarial, es que deben de ser sistemas integrales, modulares y adaptables. Un ERP se caracteriza por:

- Ser un programa con **acceso a una base de datos**.
- Sus **componentes interactúan** entre sí.
- Los **datos** deben ser consistentes, completos.



En ocasiones son sistemas complejos y difíciles de implantar, debido a que necesitan un desarrollo personalizado para cada empresa a partir del paquete inicial. Estas adaptaciones suelen encargarse a las consultorías.

La consultoría en materia de ERP puede ser de dos tipos:

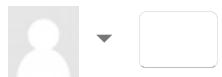
- **Consultoría de negocios.** Estudia los procesos de negocio de la compañía, y evalúa su correspondencia con los procesos del sistema ERP para poder personalizarlo, y de este modo ajustarlo a las necesidades de la organización.
- **Consultoría técnica.** Conlleva el estudio de los recursos tecnológicos existentes, en ocasiones implica la programación del sistema, obtener determinados informes.

En la actualidad, y debido a la amplia implantación de las intranets en las empresas, la mayoría de los sistemas ERP tienen interfaz web, lo que aporta la ventaja de permitir el acceso al ERP a través del navegador web.

CRM (Gestión de la Relación con el Cliente)

Un CRM (Customer Relationship Management) es un sistema de gestión de información que se centra en gestionar toda la información referida a los clientes de la empresa.

En la actualidad los sistemas ERP suelen tener incluido un módulo CRM y los sistemas CRM suelen tener la posibilidad de añadir módulos de gestión empresarial (ERP).



Lenguaje de marcas y sistemas de gestión de información

Página Principal ► CIDEAD ► Formación Profesional ► Ciclos de Grado Superior ► Desarrollo de Aplicaciones Multiplataforma (DAM) ► 1_DAM ► 30373 ► UT02: Utilización de lenguajes de marcas en entornos web ► 2.1. Examen para LMSGI02.

2.1. Examen para LMSGI02.

Intentos permitidos: 3

Este cuestionario se cerró el sábado, 21 de diciembre de 2019, 23:59

Método de calificación: Calificación más alta

Resumen de sus intentos previos

Intento	Estado	Calificación / 10,00	Revisión
1	Finalizado Enviado: viernes, 20 de diciembre de 2019, 16:45	9,23	Revisión
2	Finalizado Enviado: domingo, 22 de diciembre de 2019, 00:01	0,00	Revisión

**Su calificación final en este cuestionario es
9,23/10,00**

[Volver al curso](#)

ADMINISTRACIÓN

Administración del curso



ACCESIBILIDAD



Desplegar ATbar (barra AT) (¿siempre?)

Usted se ha identificado como MANUEL JOSE GONZALEZ CALVO (Salir)
30373

Transformación de documentos XML

Introducción a XSLT



Jorge Castellanos Vega

Introducción

- **XSL → Lenguaje extensible de hojas de estilo**
 - Familia de lenguajes basados en XML que describen la forma de transformar la información contenida en un documento para que sea representada en un medio
 - Parecido a lo que hace un archivo CSS pero aplicado a XML
 - Puesto que XML no tiene etiquetas predefinidas los navegadores no saben cómo mostrarlo
 - La etiqueta <table> en XML no tiene porque ser el inicio de una declaración de tabla HTML
 - XSL describe cómo deben mostrarse los documentos XML

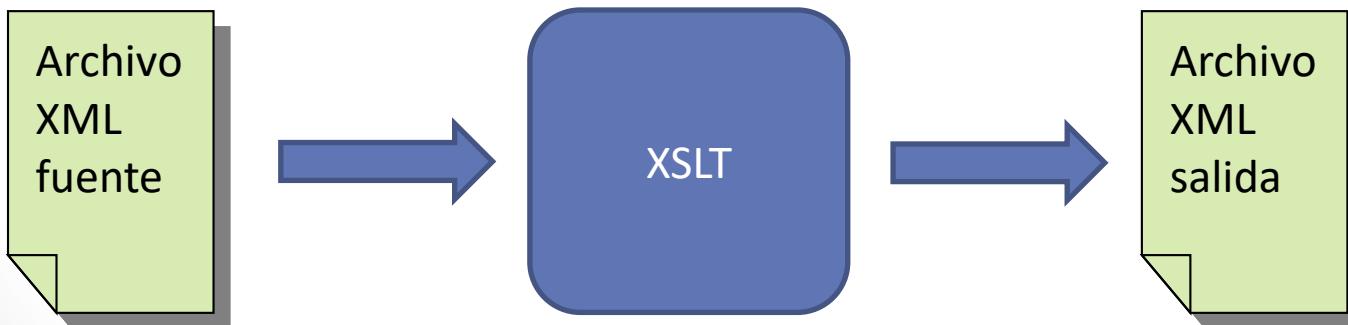
Introducción

- **Lenguajes XSL**
 - XSLT – Lenguaje de transformación de documentos XML
 - XPath – Lenguaje para navegar en documentos XML
 - XSL-FO – Lenguaje para formatear documentos XML (desaparecido en 2013)
 - XQuery – Lenguaje de consultas en documentos XML

XSLT

- **XSL Transformations**

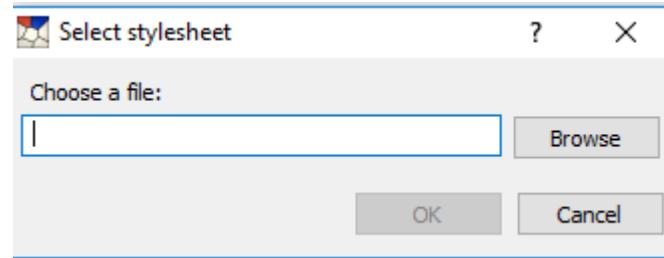
- Parte más importante de XSL
- Permite transformar un documento XML en:
 - Otro documento XML
 - Otro tipo de documento reconocible por un navegador
- Permite añadir o eliminar elementos y atributos en el archivo de salida. También cambiar el orden o decidir qué elementos mostrar y cuáles no.



XSLT

- **Xsl Transformations**

- Utiliza **Xpath** para navegar en documentos XML
- Durante el proceso de transformación XSLT utiliza **Xpath** para definir partes del documento origen que coincidirán con una o más plantillas determinadas.
- Cuando se encuentra una coincidencia XSLT transformará la parte coincidente del documento origen en el documento de destino
- XSLT funciona en la mayoría de navegadores
- En **XML Copy Editor** podemos aplicar una transformación sobre un documento XML pulsando F8 y seleccionando el archivo XSL a aplicar



Ejemplo

- Archivo XML → Canciones.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <MisCancionesPreferidas>
3  <archivo>
4    <canción>Hangar 18</canción>
5    <artista>Megadeth</artista>
6    <disco>Rust in Peace</disco>
7  </archivo>
8  <archivo>
9    <canción>Peace Sells</canción>
10   <artista>Megadeth</artista>
11   <disco>Peace Sells...But Who's Buying</disco>
12 </archivo>
13 <archivo>
14   <canción>Master of Puppets</canción>
15   <artista>Metallica</artista>
16   <disco>Master of Puppets</disco>
17 </archivo>
18 <archivo>
19   <canción>Among The Living</canción>
20   <artista>Anthrax</artista>
```

```
21  <disco>Among The Living</disco>
22  </archivo>...
23  <archivo>
24    <canción>For Whom The Bell Tolls</canción>
25    <artista>Metallica</artista>
26    <disco>Ride The Lightning</disco>
27  </archivo>...
28 </MisCancionesPreferidas>
```

Ejemplo

- Archivo XSL → Canciones.xsl

```
1  <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3 <xsl:template match="/">
4 <html>
5 <body>
6 | <h1>Mis Canciones favoritas</h1>
7 <table style="border:1px solid blue;">
8 <tr style="background-color:lightblue;">
9 | <th style="text-align:left">Título</th>
10 | <th style="text-align:left">Artista</th>
11 | <th style="text-align:left">Disco</th>
12 </tr>
13 <xsl:for-each select="MisCancionesPreferidas/archivo">
14 <tr>
15 | <td><xsl:value-of select="canción"/></td>
16 | <td><xsl:value-of select="artista"/></td>
17 | <td><xsl:value-of select="disco"/></td>
18 </tr>
19 </xsl:for-each>
20 </table>
21 </body>
22 </html>
23 </xsl:template>
24 </xsl:stylesheet>
25
```

Ejemplo

- Resultado de la transformación

Mis Canciones favoritas

Título	Artista	Disco
Hangar 18	Megadeth	Rust in Peace
Peace Sells	Megadeth	Peace Sells...But Who's Buying
Master of Puppets	Metallica	Master of Puppets
Among The Living	Anthrax	Among The Living
For Whom The Bell Tolls	Metallica	Ride The Lightning

→ [Enlace: XSLT Tryit Editor](#)

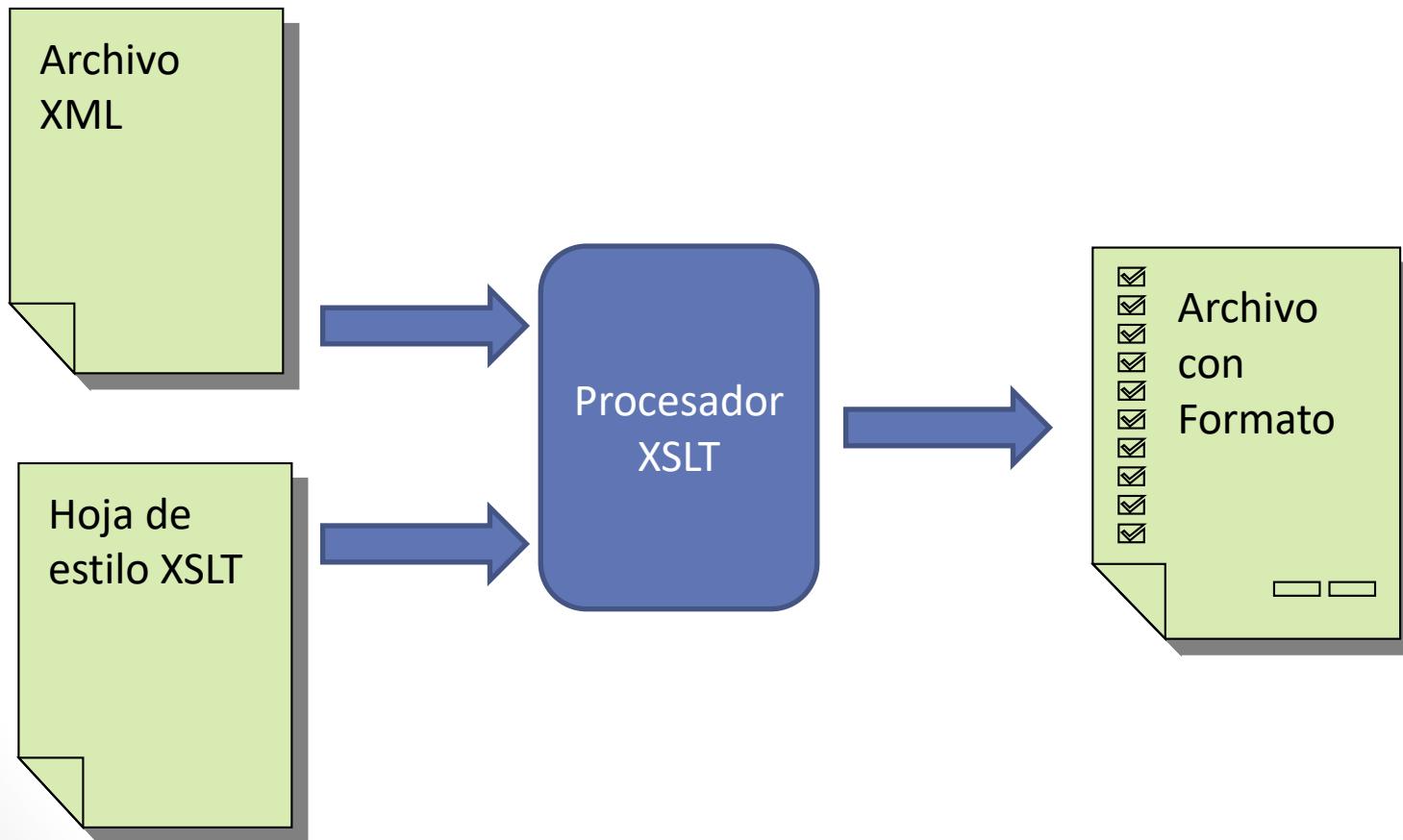
Ejemplo

- Resultado de la transformación

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <html>
3 <body>
4   <h1>Mis Canciones favoritas</h1>
5   <table style="border:1px solid blue;">
6     <tr style="background-color:lightblue;">
7       <th style="text-align:left">Título</th>
8       <th style="text-align:left">Artista</th>
9       <th style="text-align:left">Disco</th>
10      </tr>
11      <tr>
12        <td>Hangar 18</td>
13        <td>Megadeth</td>
14        <td>Rust in Peace</td>
15      </tr>
16      <tr>
17        <td>Peace Sells</td>
18        <td>Megadeth</td>
19        <td>Peace Sells...But Who's Buying</td>
20      </tr>
```

```
21 <tr>
22   <td>Master of Puppets</td>
23   <td>Metallica</td>
24   <td>Master of Puppets</td>
25 </tr>
26 <tr>
27   <td>Among The Living</td>
28   <td>Anthrax</td>
29   <td>Among The Living</td>
30 </tr>
31 <tr>
32   <td>For Whom The Bell Tolls</td>
33   <td>Metallica</td>
34   <td>Ride The Lightning</td>
35 </tr>
36 </table>
37 </body>
38 </html>
```

Esquema de funcionamiento



Declaración

- Podemos declarar el documento de cualquiera de las dos maneras siguientes:

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

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

XSLT - Elementos

- Elemento **plantilla**
 - Definido como **<xsl:template match="ExpresiónXPath">**
 - El atributo match sirve para indicar dónde aplicar la plantilla. Esta se aplicará a los elementos hijos del nodo que coincide con la expresión del atributo
 - P.e: match="/" define el documento entero

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0">
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4  <xsl:template match="/">
5  <html>
6  <body>
7 | <h1>Mis Canciones favoritas</h1>
8  <table style="border: 1px solid blue;">
9  <tr style="background-color:lightblue;">
10 | <th style="text-align:left">Título</th>
11 | <th style="text-align:left">Artista</th>
12 | <th style="text-align:left">Disco</th>
```

XSLT - Elementos

- Elemento **valor de**
 - Formato **<xsl:value-of select="ExpresiónXPath">**
 - Permite extraer el valor de un elemento XML y añadirlo en el archivo de salida

```
<tr>
  <td><xsl:value-of select="MisCancionesPreferidas/archivo/canción"/></td>
  <td><xsl:value-of select="MisCancionesPreferidas/archivo/artista"/></td>
  <td><xsl:value-of select="MisCancionesPreferidas/archivo/disco"/></td>
</tr>
```

- Es posible utilizar expresiones de **Xpath** más complejas para seleccionar elementos

XSLT - Elementos

- Elemento **for each**
 - Definido como **<xsl:for each select="expresión-XPath">**
 - Permite recorrer un conjunto de nodos XML

```
<xsl:for-each select="MisCancionesPreferidas/archivo">
  <tr>
    <td><xsl:value-of select="canción"/></td>
    <td><xsl:value-of select="artista"/></td>
    <td><xsl:value-of select="disco"/></td>
  </tr>
</xsl:for-each>
```

- En el ejemplo obtenemos los valores de los nodos **canción**, **artista** y **disco** en la ruta XPath **MisCancionesPreferidas/archivo**

XSLT - Elementos

- Elemento **for each**
 - Podemos modificar la expresión Xpath para seleccionar ciertos valores:
 - Los operadores de filtrado son:
 - Igual =
 - Distinto !=
 - < menor que
 - > mayor que

```
<xsl:for-each select="MisCancionesPreferidas/archivo[artista='Megadeth']">
  <tr>
    <td><xsl:value-of select="canción"/></td>
    <td><xsl:value-of select="artista"/></td>
    <td><xsl:value-of select="disco"/></td>
  </tr>
</xsl:for-each>
```

- En el ejemplo obtenemos los valores de los nodos **canción**, **artista** y **disco** para el artista indicado

XSLT - Elementos

- Elemento **ordenar**

- Permite ordenar la salida
- Por defecto:
 - Tipo texto
 - Ascendente
 - Upper-first

```
<xsl:sort
    select = string-expression
    lang = { nmtoken }
    data-type = { "text" | "number" | QName }
    order = { "ascending" | "descending" }
    case-order = { "upper-first" | "lower-first" } >
-</xsl:sort>
```

- El atributo select indica el elemento XML sobre el que ordenar
- Simplemente hay que añadir un elemento **<xsl:sort>** dentro de **<xsl:for-each>**

```
<xsl:for-each select="MisCancionesPreferidas/archivo[artista='Megadeth']">
<xsl:sort select="canción"/>
<tr>
<td><xsl:value-of select="canción"/></td>
<td><xsl:value-of select="artista"/></td>
<td><xsl:value-of select="disco"/></td>
</tr>
</xsl:for-each>
```

XSLT - Elementos

- Elemento **alternativa simple**
 - Definido como **<xsl:if test="expresión">**
 - Se añade dentro del elemento **<xsl:for-each>**

```
<xsl:for-each select="MisCancionesPreferidas/archivo">
  <xsl:if test="artista='Megadeth'">.....
    <tr>
      <td><xsl:value-of select="canción"/></td>
      <td><xsl:value-of select="artista"/></td>
      <td><xsl:value-of select="disco"/></td>
    </tr>
  </xsl:if>....
</xsl:for-each>
```

XSLT - Elementos

- Elemento **Alternativa Múltiple**
 - Definido como **<xsl:choose>**

Sintaxis

```
<xsl:choose>
  <xsl:when test="condición1">
    salida para condición 1
  </xsl:when>
  <xsl:when test="condición2">
    salida para condición 2
  </xsl:when>
  .....
  <xsl:otherwise>
    salida para cualquier otro caso
  </xsl:otherwise>
</xsl:choose>
```

Ejemplo

```
<xsl:for-each select="MisCancionesPreferidas/archivo">
  <xsl:choose>
    <xsl:when test="artista='Megadeth'">
      <tr>
        <td>MeGaDeTH</td>
        <td><xsl:value-of select="canción"/></td>
        <td><xsl:value-of select="artista"/></td>
        <td><xsl:value-of select="disco"/></td>
      </tr>
    </xsl:when>
    <xsl:when test="artista='Metallica'">
      <tr>
        <td>MeTaLLiCa</td>
        <td><xsl:value-of select="canción"/></td>
        <td><xsl:value-of select="artista"/></td>
        <td><xsl:value-of select="disco"/></td>
      </tr>
    </xsl:when>
  </xsl:choose>
</xsl:for-each>
```

XSLT - Elementos

- Elemento **aplicar plantilla**
 - Formato **<xsl:apply-templates select="AtributoSelección">**
 - Selecciona todos los nodos coincidentes con los nodos indicados en el atributo **select** y los itera aplicando las plantillas que estén definidas para ellos.

XSLT - Elementos

- Elemento **aplicar plantilla** – ejemplo (1/2)

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="1.0">
3  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4  <xsl:template match="/"><!-- plantilla principal -->
5  <html>
6  <body>
7  ||| <h1>Mis Canciones</h1>
8  <xsl:apply-templates/><!-- llamada para aplicar el resto de plantillas -->
9  </body>
10 </html>
11 </xsl:template>
12
13 <xsl:template match="archivo"><!-- plantilla para el nodo archivo -->
14 <p>
15 <xsl:apply-templates select="canción"/><!-- aplica la plantilla definida para el nodo canción -->
16 <xsl:apply-templates select="artista"/><!-- aplica la plantilla definida para el nodo artista -->
17 <xsl:apply-templates select="disco"/><!-- aplica la plantilla definida para el nodo disco -->
18 </p>
19 </xsl:template>
20
```

XSLT - Elementos

- Elemento **aplicar plantilla** – ejemplo (2/2)

```
21 ⊥<xsl:template match="canción"><!-- plantilla definida para el nodo canción--&gt;
22 ⊥ Canción &lt;span style="color:darkblue"&gt;
23 | &lt;xsl:value-of select=". "/&gt;&lt;/span&gt;
24 | &lt;br /&gt;
25 &lt;/xsl:template&gt;
26
27 ⊥&lt;xsl:template match="artista"&gt;<!-- plantilla definida para el nodo artista--&gt;
28 ⊥ Artista &lt;span style="color:blue"&gt;
29 | &lt;xsl:value-of select=". "/&gt;&lt;/span&gt;
30 | &lt;br /&gt;
31 &lt;/xsl:template&gt;
32
33 ⊥&lt;xsl:template match="disco"&gt;<!-- plantilla definida para el nodo disco--&gt;
34 ⊥ Disco &lt;span style="color:lightblue"&gt;
35 | &lt;xsl:value-of select=". "/&gt;&lt;/span&gt;
36 | &lt;br /&gt;
37 &lt;/xsl:template&gt;
38 &lt;/xsl:stylesheet&gt;
39</pre>
```

XSLT - Elementos

- Elemento aplicar plantilla - resultado

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <html>
3  <body><h1>Mis Canciones</h1>
4  <p>
5  Canción <span style="color:darkblue">Hangar 18</span><br/>
6  Artista <span style="color:blue">Megadeth</span><br/>
7  Disco <span style="color:lightblue">Rust in Peace</span><br/></p>
8  <p>
9  Canción <span style="color:darkblue">Peace Sells</span><br/>
10 Artista <span style="color:blue">Megadeth</span><br/>
11 Disco <span style="color:lightblue">Peace Sells...But Who's Buying</span><br/></p>
12 <p>
13 Canción <span style="color:darkblue">Master of Puppets</span><br/>
14 Artista <span style="color:blue">Metallica</span><br/>
15 Disco <span style="color:lightblue">Master of Puppets</span><br/></p>
16 <p>
17 Canción <span style="color:darkblue">Among The Living</span><br/>
18 Artista <span style="color:blue">Anthrax</span><br/>
19 Disco <span style="color:lightblue">Among The Living</span><br/></p>
20 <p>
21 Canción <span style="color:darkblue">For Whom The Bell Tolls</span><br/>
22 Artista <span style="color:blue">Metallica</span><br/>
23 Disco <span style="color:lightblue">Ride The Lightning</span><br/></p>
24 </body>
25 </html>
26
```

Algunos ejemplos

- Utilizaremos un documento como el siguiente

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <MiBibliotecaMP3>
3  <archivo>
4  <canción>Hangar 18</canción>
5  <artista>Megadeth</artista>
6  <disco>Rust in Peace</disco>
7  <puntuacion>9</puntuacion>
8  </archivo>
9  <archivo>
10 <canción>Peace Sells</canción>
11 <artista>Megadeth</artista>
12 <disco>Peace Sells...But Who's Buying</disco>
13 <puntuacion>9</puntuacion>
14 </archivo>
15 <archivo>
16 <canción>Master of Puppets</canción>
17 <artista>Metallica</artista>
18 <disco>Master of Puppets</disco>
19 <puntuacion>10</puntuacion>
20 </archivo>
21 <archivo>
22 <canción>Among The Living</canción>
23 <artista>Anthrax</artista>
24 <disco>Among The Living</disco>
25 <puntuacion>8</puntuacion>
26 </archivo>
27 <archivo>
28 <canción>For Whom The Bell Tolls</canción>
29 <artista>Metallica</artista>
30 <disco>Ride The Lightning</disco>
31 <puntuacion>8</puntuacion>
32 </archivo>
33 </MiBibliotecaMP3>
```

Algunos ejemplos

- Ejemplo: plantilla principal y bucle

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0">
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4   <xsl:template match="/"><!-- plantilla principal -->
5   <html>
6     <body>
7       <h1>Mis Canciones</h1>
8       <table style="border:1px solid blue;">
9         <tr style="background-color:lightblue;">
10          <th style="text-align:left">Título</th>
11          <th style="text-align:left">Artista</th>
12          <th style="text-align:left">Disco</th>
13          <th style="text-align:left">Puntuación</th>
14        </tr>
15        <xsl:for-each select="MiBibliotecaMP3/archivo"><!-- inicio bucle -->
16          <tr>
17            <td><xsl:value-of select="canción"/></td>
18            <td><xsl:value-of select="artista"/></td>
19            <td><xsl:value-of select="disco"/></td>
20            <td><xsl:value-of select="puntuacion"/></td>
21          </tr>
22        </xsl:for-each><!-- final bucle -->
23      </table>
24    </body>
25  </html>
26 </xsl:template>
27 </xsl:stylesheet>
```

Mis Canciones

Titulo	Artista	Disco	Puntuación
Hangar 18	Megadeth	Rust in Peace	9
Peace Sells	Megadeth	Peace Sells...But Who's Buying	9
Master of Puppets	Metallica	Master of Puppets	10
Among The Living	Anthrax	Among The Living	8
For Whom The Bell Tolls	Metallica	Ride The Lightning	8

Algunos ejemplos

- Ejemplo: Ordenando por puntuación

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0">
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4   <xsl:template match="/"><!-- plantilla principal -->
5   <html>
6   <body>
7     <h1>Mis Canciones</h1>
8     <table style="border: 1px solid blue;">
9       <tr style="background-color:lightblue;">
10      <th style="text-align:left">Título</th>
11      <th style="text-align:left">Artista</th>
12      <th style="text-align:left">Disco</th>
13      <th style="text-align:left">Puntuación</th>
14    </tr>
15    <xsl:for-each select="MiBibliotecaMP3/archivo"><!-- inicio bucle -->
16      <xsl:sort select="puntuacion"/><!-- ordenado por puntuación-->
17    <tr>
18      <td><xsl:value-of select="canción"/></td>
19      <td><xsl:value-of select="artista"/></td>
20      <td><xsl:value-of select="disco"/></td>
21      <td><xsl:value-of select="puntuacion"/></td>
22    </tr>
23  </xsl:for-each><!-- final bucle -->
24  </table>
25  </body>
26  </html>
27 </xsl:template>
28 </xsl:stylesheet>
```

Mis Canciones

Titulo	Artista	Disco	Puntuación
Master of Puppets	Metallica	Master of Puppets	10
Among The Living	Anthrax	Among The Living	8
For Whom The Bell Tolls	Metallica	Ride The Lightning	8
Hangar 18	Megadeth	Rust in Peace	9
Peace Sells	Megadeth	Peace Sells...But Who's Buying	9

Error! Por defecto considera ordenación ascendente y tipo texto por lo que la ordenación no es todo lo correcta que nos gustaría....

Algunos ejemplos

- Ejemplo: Ordenando especificando tipo de datos

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0">
3 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4 <xsl:template match="/"><!-- plantilla principal -->
5 <html>
6 <body>
7 | <h1>Mis Canciones</h1>
8 <table style="border:1px solid blue;">
9 <tr style="background-color:lightblue;">
10 <th style="text-align:left">Título</th>
11 <th style="text-align:left">Artista</th>
12 <th style="text-align:left">Disco</th>
13 <th style="text-align:left">Puntuación</th>
14 </tr>
15 <xsl:for-each select="MiBibliotecaMP3/archivo"><!--
16 | <xsl:sort select="puntuacion" data-type="number"/>
17 <tr>
18 | <td><xsl:value-of select="canción"/></td>
19 | <td><xsl:value-of select="artista"/></td>
20 | <td><xsl:value-of select="disco"/></td>
21 | <td><xsl:value-of select="puntuacion"/></td>
22 </tr>
23 </xsl:for-each><!-- final bucle -->
24 </table>
25 </body>
26 </html>
27 </xsl:template>
28 </xsl:stylesheet>
```

Mis Canciones

Título	Artista	Disco	Puntuación
Among The Living	Anthrax	Among The Living	8
For Whom The Bell Tolls	Metallica	Ride The Lightning	8
Hangar 18	Megadeth	Rust in Peace	9
Peace Sells	Megadeth	Peace Sells...But Who's Buying	9
Master of Puppets	Metallica	Master of Puppets	10

Al indicar tipo de datos numérico la ordenación ahora sí es correcta

Algunos ejemplos

- Ejemplo: Mostramos solo los puntuados por encima de 9

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3 <xsl:template match="/"><!-- plantilla principal -->
4 <html>
5 <body>
6 <h1>Mis Canciones</h1>
7 <table style="border: 1px solid blue;">
8 <tr style="background-color:lightblue;">
9 <th style="text-align:left">Título</th>
10 <th style="text-align:left">Artista</th>
11 <th style="text-align:left">Disco</th>
12 <th style="text-align:left">Puntuación</th>
13 </tr>
14 <xsl:for-each select="MiBibliotecaMP3/archivo"><!-- inicio bucle -->
15 <xsl:if test="puntuacion > 9"><!-- solo muestra puntuados > 9 -->...
16 <tr>
17 <td><xsl:value-of select="canción"/></td>
18 <td><xsl:value-of select="artista"/></td>
19 <td><xsl:value-of select="disco"/></td>
20 <td><xsl:value-of select="puntuacion"/></td>
21 </tr>
22 </xsl:if>...
23 </xsl:for-each><!-- final bucle -->
24 </table>
25 </body>
26 </html>
27 </xsl:template>
28 </xsl:stylesheet>
```

Mis Canciones

Titulo	Artista	Disco	Puntuación
Master of Puppets	Metallica	Master of Puppets	10

Algunos ejemplos

- Alternativa múltiple

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3 <xsl:template match="/"><!-- plantilla principal -->
4 <html>
5 <body>
6 <h1>Mis Canciones</h1>
7 <table style="border: 1px solid blue;">
8 <tr style="background-color:lightblue;">
9 <th style="text-align:left">Título</th>
10 <th style="text-align:left">Artista</th>
11 <th style="text-align:left">Disco</th>
12 <th style="text-align:left">Puntuación</th>
13 </tr>
14 <xsl:for-each select="MiBibliotecaMP3/archivo"><!-- inicio bucle-->
15 <xsl:choose>
16 <xsl:when test="puntuacion = 8"><!-- puntuados con 8, fondo amarillo -->...
17 <r>
18 <td style="background-color:yellow"><xsl:value-of select="canción"/></td>
19 <td style="background-color:yellow"><xsl:value-of select="artista"/></td>
20 <td style="background-color:yellow"><xsl:value-of select="disco"/></td>
21 <td style="background-color:yellow"><xsl:value-of select="puntuacion"/></td>
22 </tr>
23 </xsl:when>
24 <xsl:when test="puntuacion = 9"><!-- puntuados con 9, fondo naranja-->...
25 <r>
26 <td style="background-color:orange"><xsl:value-of select="canción"/></td>
27 <td style="background-color:orange"><xsl:value-of select="artista"/></td>
28 <td style="background-color:orange"><xsl:value-of select="disco"/></td>
29 <td style="background-color:orange"><xsl:value-of select="puntuacion"/></td>
30 </r>
31 </xsl:when>...
32 <xsl:otherwise><!-- el resto, fondo rojo -->...
33 <r>
34 <td style="background-color:red"><xsl:value-of select="canción"/></td>
35 <td style="background-color:red"><xsl:value-of select="artista"/></td>
36 <td style="background-color:red"><xsl:value-of select="disco"/></td>
37 <td style="background-color:red"><xsl:value-of select="puntuacion"/></td>
38 </r>
39 </xsl:otherwise>
40 </xsl:choose>...
41 <xsl:for-each><!-- final bucle-->
42 </table>
43 </body>
44 </html>
45 </xsl:template>
46 </xsl:stylesheet>
```

Mis Canciones

Titulo	Artista	Disco	Puntuación
Hangar 18	Megadeth	Rust in Peace	9
Peace Sells	Megadeth	Peace Sells...But Who's Buying	9
Master of Puppets	Metallica	Master of Puppets	10
Among The Living	Anthrax	Among The Living	8
For Whom The Bell Tolls	Metallica	Ride The Lightning	8

Referencias

- W3School - <https://www.w3schools.com>