

1.1. SGML



Índice

Objetivos	3
Introducción.....	4
El origen de los lenguajes de marcas.....	4
Tipos de lenguajes de marcas	5
Marcas de procesamiento	5
Marcas de descripción	5
El estándar SGML	6
Características de los lenguajes de marcas.....	7
Beneficios de los lenguajes de marcas	9
SGML.....	10
Estructura.....	10
Estructura natural	10
Estructura lógica.....	11
Estructura física	12
Componentes SGML	12
XML.....	14
Conceptos	14
Objetivos de XML	14
Componentes XML.....	14
Estructura de un XML.....	15
XSL.....	17
Despedida	18
Resumen.....	18

Objetivos

En esta unidad perseguimos los siguientes objetivos:

- Aprender qué son los lenguajes de marcas.
- Conocer los orígenes de SGML.
- Conocer la estructura en lenguajes de marcas a través de su definición.
- Conocer los lenguajes genéricos, en concreto SGML.
- Conocer el uso de DTD para generar tipos de lenguajes.
- Conocer ejemplos de estructuras en los lenguajes que derivan de SGML.

Introducción

El origen de los lenguajes de marcas

Los lenguajes de marcado se remontan al uso de la imprenta y a cómo los autores indicaban, a través de marcas, cómo se debían formatear los textos.

Normalmente **indicaban el estilo de la letra, la disposición de los títulos, los párrafos y demás elementos de formato.**

Las especificaciones o explicaciones se anotaban junto con el texto. Estas marcas especificaban operaciones tipográficas y la configuración que debía realizar el que manejaba la imprenta sobre cada elemento. Por lo tanto, eran instrucciones de formato que se aplicaban de manera global o individual sobre cada párrafo, palabra o bloque de texto.

Una vez que surge la informática se decide que las marcas deben ser estándares y escritas en ASCII, un código de 7 bits que representa los caracteres basado en el alfabeto latino.

● ● ● ●	= Ignorar la corrección y dejarlo como estaba:
.....	en Quijote Cervantes...
=	Quitar el espacio de más: la noticia
=	Añadir un espacio; en este caso
=	Cambiar el orden de letras, palabras o frases: chno
=	Marcado en el margen izquierdo, alinear verticalmente Con las líneas contiguas En párrafos, sangría izquierda a a
=	Marcado en el margen derecho, alinear verticalmente Con las líneas contiguas En párrafos, sangría derecha b b
=	Sangrar línea
=	Alinear a margen izquierdo

Ejemplo de símbolos de marcado de un texto.

¿Quieres saber más?

En el siguiente enlace puedes encontrar un listado con los símbolos más utilizados en la corrección de textos. <https://correctordetextos.com/signos.htm>

Tipos de lenguajes de marcas

Existen distintos tipos de lenguajes de marcado, que se combinan en la práctica.
Marcas de presentación

Este tipo de marcas **son códigos incrustados en el texto del documento** que producen el efecto WYSIWYG (*What You See Is What You Get*, “Lo que ves es lo que obtienes”), utilizados por los sistemas de procesamiento de textos tradicionales.

Dicho marcado se suele incluir en el documento de manera que el usuario no lo ve.

```
[mi_shortcode atributo="valor"]  
[mi_shortcode atributo="valor"]Contenidos[/mi_shortcode]
```

- Este tipo de marcado es útil para maquetar la presentación de un documento para su lectura, pero **es insuficiente para el procesamiento automático de la información**.
- El marcado de presentación resulta **más fácil de elaborar**, sobre todo para cantidades pequeñas de información.

Marcas de procesamiento

Describen las reglas de procesamiento del texto y cómo se imprimirá. **Un ejemplo de lenguajes que usan este sistema son PostScript y LaTeX**.

```
\begin{verbatim}  
Escribes tu código aquí  
\end{verbatim}
```

- Sus símbolos o marcas **indican la clase de operaciones tipográficas** que deben ser aplicadas a cada uno de los elementos del documento electrónico para dar formato al texto.
- Se utilizan **para configurar la apariencia física de los documentos**, tanto en pantalla como impreso.

Marcas de descripción

Es un lenguaje que conforma la estructura de un documento, identifica cada parte de este y el tipo de elementos que constituye cada parte.

A cada marca se le puede asignar un comportamiento determinado para dar forma al documento.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>title</title>
        <link rel="stylesheet" href="style.css">
        <script src="script.js"></script>
    </head>
    <body>
        <header>
            <h1>Título de la WEB</h1>
        </header>
        <nav>
            <a href="http://dominio.com/seccion2.html">IR SECCIÓN 2</a>
            <a href="http://dominio.com/seccion2.html">IR SECCIÓN 3</a>
        </nav>
        <section>
            <article>
                <h2>CONTENIDO PRINCIPAL</h2>
                <p>Este es el contenido principal de mi web</p>
                <div>
                    <p>Aquí tenéis una imagen.</p>
                    
                </div>
            </article>
        </section>
        <aside>
            <h3>Banner de publicidad</h3>
            <a href="http://dominio-externo.com">
                
            </a>
            <h3>Testimonios</h3>
            <p>Me gusta mucho esta página.</p>
        </aside>
        <footer>
            <h4>Avisos legales</h4>
            <a href="http://dominio.com/aviso-legal">Política de cookies</a>
            <h4>Redes sociales</h4>
            <a href="http://facebook.com/mi-pagina-de-facebook">Mi Facebook</a>
        </footer>
    </body>
</html>
```

- Determinan **tanto la estructura lógica del documento electrónico como la descripción de su contenido.**
- Utilizan las **marcas o etiquetas para describir los fragmentos de texto.**
- Una de las virtudes del marcado descriptivo es su **flexibilidad**, ya que los fragmentos de texto se etiquetan tal como son, y no como deben aparecer.
- **Simplifica la tarea de formatear un texto**, debido a que la información está separada del propio contenido.

El estándar SGML

SGML surge de la necesidad de estandarizar los lenguajes de marcado. Los primeros lenguajes de marcado usaban el estándar ASCII y seguían unas reglas previamente establecidas, pero había muchos y de distintos fabricantes. **IBM fue el primero que**, preocupado por la gran cantidad de lenguajes de marcas de la época que hacían imposible el envío de documentos de

un sistema a otro, definió el primer intento de estandarización, que fue el GML (*Generalized Markup Language*).

En 1978 el Instituto Nacional Americano de Normalización (ANSI) diseñó la especificación del procesamiento de textos, creando el **estándar SGML**, que fue retomado en 1986 por la ISO y se transformó en la norma 8879. Las siglas del lenguaje SGML (*Standar Generalized Markup Language*) traducidas a "**Estándar de Lenguaje de Marcado Generalizado**", sirven para especificar las **reglas de etiquetado de documentos**, aunque deja libertad para diseñar las marcas a los distintos creadores o fabricantes.

```
<!ELEMENT DOCUMENTO - - (titulo, contenido, autor?) >
<!ELEMENT CONTENIDO - - (capitulo+) >

<!ELEMENT CAPITULO - (subtitulo, parrafo?)

<titulo>Lenguajes de Marcas</titulo>
<contenido>
  <capitulo>
    <subtitulo>Un sistema de transmisión de información</subtitulo>
    <parrafo>Sed ut perspiciatis unde omnis iste natus error sit voluptatem
accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo
inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.
  </parrafo>
  </capitulo>
</contenido>
```

En SGML la forma de definir las etiquetas es la siguiente: se usan "**nombres de elementos**" que se encuentran **delimitados por marcas y que indican el comienzo y final de los elementos** u objetos lógicos.

SGML tiene la característica de ser flexible ya que, por un lado, permite que se definan lenguajes de marcas de forma independiente y, por otro, facilita el intercambio y conservación de documentos y recursos digitales estructurados.

Por lo tanto, SGML puede ser considerado como un **metalenguaje** o *framework* general de descripción de marcado.

Las reglas de comportamiento de estos documentos son definidas en las DTD (descripción de tipo de documento) de SGML. Un ejemplo es el conocido HTML.

Características de los lenguajes de marcas

Las características más llamativas de los lenguajes de marcado son las siguientes.

Información y marcas

Son lenguajes que se **mezclan con los datos y les aportan semántica, formato y estructura**.

Texto plano

Los archivos que contienen lenguajes de marcado son básicamente archivos de texto, por lo que **se pueden editar desde un programa de tratamiento de textos**.

Compacto

Las marcas son escritas en un mismo fichero, por lo que **es fácil acceder a ellas e interpretarlas**.

Procesamiento sencillo

Cualquier intérprete que maneje lenguajes de marcas puede procesarlo, ya que es muy sencillo. Dichas marcas **solo definen datos, estructuras y formato**.

Flexibilidad

Se han podido extender a áreas tales como los gráficos vectoriales, los servicios web o las interfaces de usuario (Android, por ejemplo); esto permite que **en un solo documento se puedan mezclar distintos estándares**.

La estructura de un documento confeccionado con lenguaje de marcado suele dividirse en **cabecera y cuerpo**, y dependiendo del tipo de documento puede subdividirse cada zona en otras.

En concreto un documento HTML tendría:

1. Una parte inicial, donde se indica la versión del estándar HTML. En ella se especifica un DTD sobre el tipo de documento que viene a continuación.

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

Ejemplo DTD para HTML 4.

2. La etiqueta <HTML> nos delimita el resto del documento, que tendrá:

- Una cabecera, delimitada por la etiqueta <HEAD>.
- Un cuerpo del documento, delimitado por la etiqueta <BODY>.

Dentro de estas partes se incluyen los siguientes elementos:

- Metadatos.
- Codificación de caracteres.
- Etiquetas o marcas.
- Elementos.
- Atributos.
- Comentarios.

Beneficios de los lenguajes de marcas

- **Permitir el intercambio de documentos** estructurados entre distintos sistemas.
- **Tener una estructura definida;** los documentos necesitan tenerla y que sea establecida con anterioridad.
- **Posibilitar el almacenamiento de la información** estructurada en el sistema como una base de datos.
- Tener un lenguaje **independiente de la representación.**
- Tener un lenguaje **independiente de la plataforma.**
- Tener un lenguaje **abstraído de la información.**

SGML

Estructura

Cuando hablamos de estructura en los lenguajes de marcas lo asociamos a varios conceptos intrínsecos a dichos lenguajes.

- Los documentos de marcas suelen tener **una estructura definida por su naturaleza** y que caracteriza al documento en cuestión. Por ejemplo, los documentos HTML contienen dos partes diferenciadas, que son el encabezado y el cuerpo.
- La estructura de un XML dependerá de cómo se haya **definido su DTD o Schema**.
- Los lenguajes de marcas establecen como condición indispensable que un documento tenga una **estructura lógica**. Eso implica que un documento contiene partes diferenciadas, pero relacionadas de alguna forma.
- Dentro del propio documento existe una **estructura asociada a la apariencia de los contenidos del mismo**. Esta estructura define los componentes físicos, el posicionamiento y la tipografía.

Estructura natural

Los distintos lenguajes de marcado necesitan ser configurados a través de unas definiciones explícitas. Estas definiciones, que en el caso de HTML, XHTML o XML se construyen a través de ficheros DTD y Schemas, nos indican la estructura natural de cada documento generado y sus estructuras lógicas y físicas.

Estas definiciones indican las partes de las que consta un tipo de fichero de un lenguaje de marcado en particular.

```
<?xml version="1.0"?>
<!DOCTYPE MENSAJE SYSTEM "mensaje.dtd">
<mensaje>
    <remite>
        <nombre>Alfredo Reino</nombre>
        <email>alf@ibium.com</email>
    </remite>
    <destinatario>
        <nombre>Bill Clinton</nombre>
        <email>president@whitehouse.gov</email>
    </destinatario>
    <asunto>Hola Bill</asunto>
    <texto>
        <parrafo>¿Hola qué tal? Hace <enfasis>mucho</enfasis> que no escribes. A
ver si llamas y quedamos para tomar algo.</parrafo>
    </texto>
</mensaje>
```

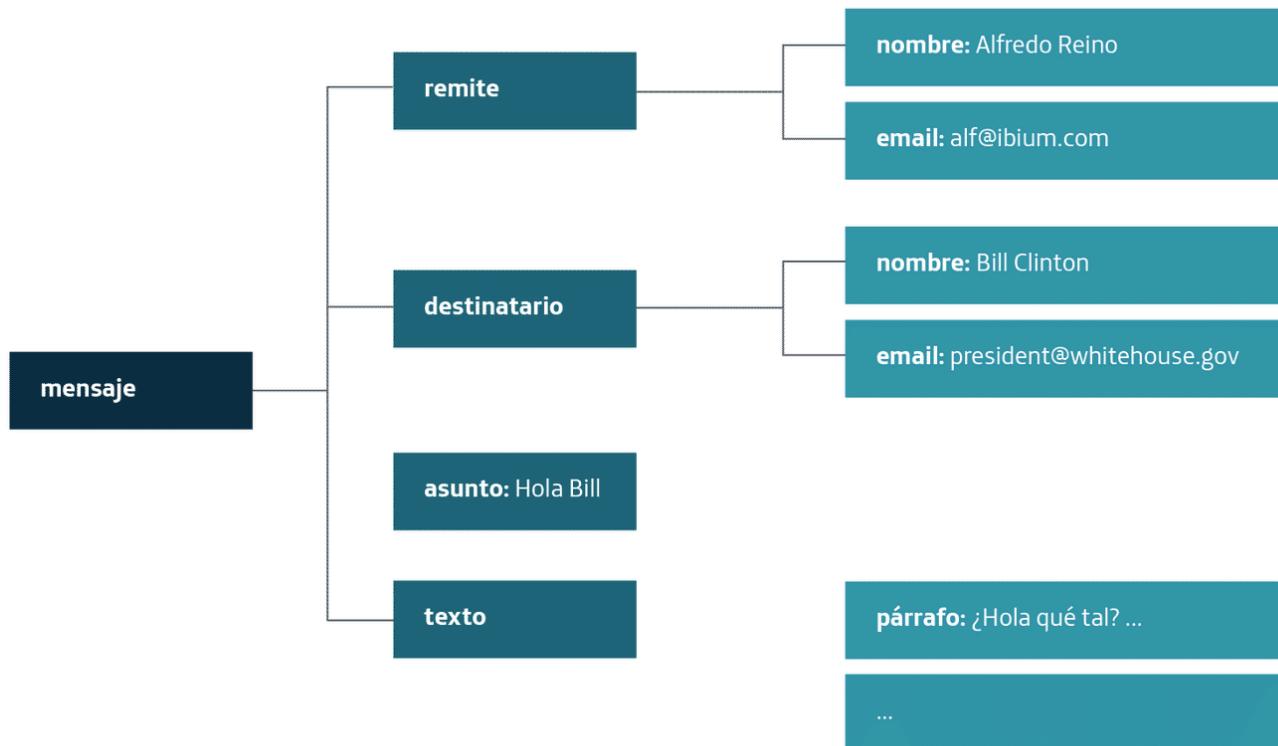
Estructura lógica

La estructura lógica está **formada por las partes que componen el documento y por sus relaciones.**

En el caso de documentos de presentación, la estructura lógica indica el orden de los contenidos y su relación entre ellos.

Por ejemplo, en el caso de una página web podríamos diferenciar entre:

- Una cabecera con información general sobre el contenido de la página, logotipo, título, etc.
- Un menú global de navegación.
- El cuerpo o contenido sustancial de la página.
- El pie de página que hace referencia a información añadida, como el *copyright*.
- Otros elementos, como búsqueda o navegación rápida.



Estructura física

La estructura física indica la **apariencia del documento sobre la pantalla**, incluyendo sus componentes físicos, el posicionamiento de los elementos y la tipografía empleada, estilos, imágenes, etc.

```
<!DOCTYPE html>
<HTML>
<HEAD>
<TITLE>Un Titulo para el Browser de turno </TITLE>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
</HEAD><BODY>
<H1>Esto es el título. </H1>
<P>Esto es un párrafo con información</P>
<BR>Una línea más escribo en <STRONG> negrita </STRONG>!
<H3>Un subtítulo</H3>
<UL>
<LI>Esto es una lista no ordenada.</LI>
<LI>Las listas quedan mejor si tienen varios elementos.</LI>
</UL>
<P>más información</P>
</BODY>
</HTML>
```

Esto es el título.

Esto es un párrafo con información
Una línea más escribo en **negrita** !

Un subtítulo

- Esto es una lista no ordenada.
- Las listas quedan mejor si tienen varios elementos.

más información

Componentes SGML

Los componentes de un documento SGML son:

La declaración

Indica lo que contendrá el documento en cuanto a caracteres, delimitadores y características opcionales de SGML utilizadas, así como la sintaxis.

La DTD

Indica la estructura del documento y de los elementos que lo van a conformar.

La instancia del documento

Es el documento conformado y validado que contiene la información textual y las etiquetas que lo definen, que están declaradas en la DTD.

Ejemplos

A continuación te mostramos algunos ejemplos de un documento SGML. En ellos podemos ver que constan de una primera sección de código con el DTD, que define las reglas según las que se comportarán las etiquetas dentro del contenido.

```
<!ELEMENT DOCUMENTO - - (titulo, contenido, autor?)>
<!ELEMENT CONTENIDO - - (capitulo+)>
<!ELEMENT CAPITULO - (subtitulo, párrafo?)>
```

DTD para definir la estructura del documento

```
<documento>
    <titulo>HIPERTEXTO</titulo>
    <autor> MARIA JESUS LAMARCA </autor>
    <contenido>
        <capitulo>
            <subtitulo>
                EL NUEVO CONCEPTO DE DOCUMENTO EN LA CULTURA DE LA IMAGEN
            </subtitulo>
            <párrafo>
                Doctorado: Fundamentos, Metodología y Aplicaciones de las
                Tecnologías Documentales y Procesamiento de la Información.
            </párrafo>
        </capitulo>
    </contenido>
</documento>
```

Contenido del documento.

XML

Conceptos

XML surge del estándar SGML para definir la gramática de lenguajes específicos en la creación y estructuración de documentos grandes. También es usado para generar bases de datos documentales y para el intercambio de datos entre sistemas.

Su desarrollo comenzó en el año 1996, precisamente en el entorno empresarial, donde el HTML se había quedado corto respecto a las necesidades. En 1998 el W3C recoge la recomendación creando el estándar.

XML es un lenguaje estándar que sigue las recomendaciones de W3C y que se creó para poder estructurar datos. No define las etiquetas ni cómo se utilizan, sino un conjunto de reglas sintácticas para poder crear las etiquetas y los documentos. Por lo tanto, no es un lenguaje sino un meta lenguaje, y en la actualidad es el estándar universal para el intercambio de datos. Por otra parte, es independiente del navegador y cada etiqueta realiza siempre la misma función. Con XML se puede jerarquizar la información y estructurar los datos describiendo los contenidos.

Objetivos de XML

1. Servir para el **intercambio de datos en Internet** con cualquier lenguaje de programación.
2. Facilitar la **construcción de documentos estructurados** y de fácil sintaxis.
3. Generar documentos **legibles para el ser humano**.
4. **Generar de forma rápida estructuras de datos** con un lenguaje normalizado y extensible.
5. Ser **independiente del hardware y del software** que lo maneja.

Componentes XML

XML DTD

Define la estructura y componentes de un tipo de documento XML:

```
<!DOCTYPE nota[  
  <!ELEMENT nota(to, from, heading, body)>  
  <!ELEMENT para (#PCDATA)>  
  <!ELEMENT de (#PCDATA)>  
  <!ELEMENT asunto (#PCDATA)>  
  <!ELEMENT cuerpo (#PCDATA)>  
>]
```

XML Schemas

Define la estructura, igual que un DTD, y permite validar un documento.

```
<xs:element name="nota">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="para" type="xs:string"/>
      <xs:element name="de" type="xs:string"/>
      <xs:element name="asunto" type="xs:string"/>
      <xs:element name="cuerpo" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- **XML Namespace:** especificaciones sobre cómo se conecta una URL con los atributos y tags de un XML.
- **DOM:** llamadas a funciones para manejar ficheros XML y HTML.
- **Xlink:** método para enlaces a un fichero XML.
- **XSL:** lenguaje para generación de hojas de estilo.
- **Xpointer:** definición de una sintaxis para la señalización de partes de un documento.

Elementos

Un sistema XML típico consta de tres tipos de archivos:

- **Datos XML y las etiquetas** que describen el significado y la estructura de los datos.
- **Esquemas XML**, que definen las reglas. Por ejemplo, un esquema podría servir para asegurarse de que los usuarios no pueden escribir texto en un campo de fecha.
- **Transformaciones XML**, que permiten usar datos en programas y archivos. Por ejemplo, una transformación podría agregar datos relativos a las compras a un libro de contabilidad, mientras que otra transformación podría insertar la misma información en un programa contable. Un estándar de W3C, como XSTL, nos ayuda a incorporar datos desde el documento hacia otro HTML.

Estructura de un XML

Cabecera

En esta zona se encuentran las **declaraciones XML**, que son instrucciones que identifican el archivo principal y los demás archivos asociados.

```
<?xml version="1.0" encoding="UTF-7"?>
```

Declaraciones XML

Raíz

Elemento **raíz o nodo principal** es el que contiene al resto de nodos y etiquetas, con sus datos para cada registro. También debe contener los atributos que se relacionan con dicho nodo principal.

Es un elemento imprescindible en todo documento XML para que esté bien formado.

```
<Disco>
  ...
</Disco>
```

Raíz o nodo principal.

Etiquetas

Etiquetas y datos **son los componentes principales** del archivo que cuelgan de cada nodo.

```
<autor>Los Beatles</autor>
<titulo>Let it Be</titulo>
<formato>Wav</formato>
<localizacion>\discos\pop</localizacion>
```

Etiquetas y datos XML.

Atributos

Son otra forma de incluir datos en los nodos del documento, aunque en este caso perdemos la posibilidad de jerarquizar la información.

Los atributos se usan frecuentemente **para identificar un nodo en concreto o transmitir información propia del nodo donde se encuentran**.

```
<autor dni="58585858F" ref="G58">Los Beatles</autor>
<titulo alias="LIT">Let it Be</titulo>
<formato>Wav</formato>
<localizacion>\discos\pop</localizacion>
```

Atributos XML.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-7"?>
<Disco id="F478">
  <autor>Los Beatles</autor>
  <titulo>Let it Be</titulo>
  <formato>Wav</formato>
  <localizacion>\discos\pop</localizacion>
</Disco>
```

XSL

XSLT (eXtensible Stylesheet Language for Transformations) es un lenguaje que permite realizar transformaciones a un XML para obtener otro tipo de documento, como un HTML o un documento de texto plano.

La hoja de estilos XSLT no es más que otro XML con las reglas de transformación del documento. La extensión del archivo es .xsl.

Las ventajas de utilizar documentos XSLT frente a otro tipo de lenguajes de marcas como CSS son:

1. Se puede **cambiar el orden** de los elementos.
2. Se puede **operar con sus valores**.
3. Se pueden **agrupar sus elementos**.



XSLT permite realizar modificaciones a un XML para obtener otro tipo de documento.

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>Notas de departamento</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>De</th>
<th>Para</th>
<th>Asunto</th>
<th>Cuerpo</th>
</tr>
<xsl:for-each select="ListadoNotas/nota">
<tr>
<td><xsl:value-of select="de"/></td>
<td><xsl:value-of select="para"/></td>
<td><xsl:value-of select="asunto"/></td>
<td><xsl:value-of select="cuerpo"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

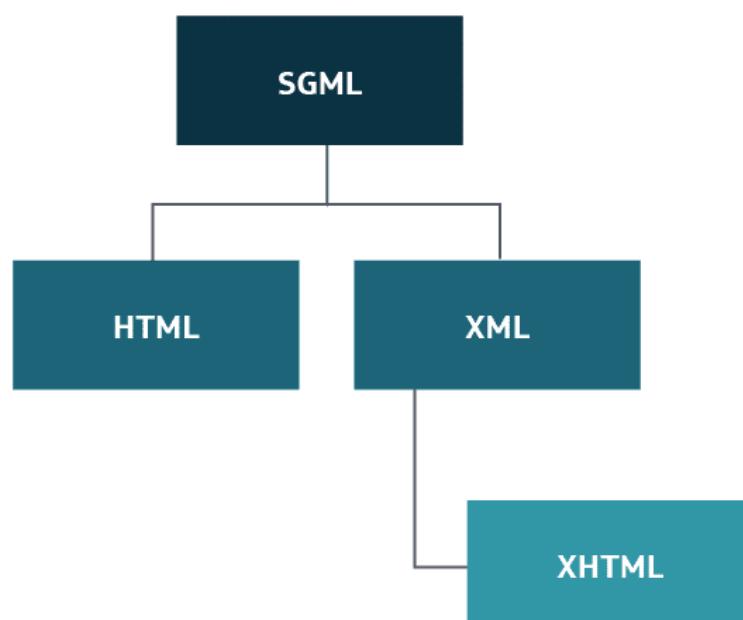
Ejemplo documento XSL.

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

- Los lenguajes de marcas **se emplean para la transmisión de información**, tanto entre máquina y usuario, como entre diferentes sistemas informáticos.
- **El lenguaje SGML fue el más destacado y ha derivado en los lenguajes XML y HTML**, muy extendidos en los sistemas de información actuales, como la web.



- Con un lenguaje de marcas **podemos construir documentos complejos y/o transferir grandes cantidades de información estructurada** mediante etiquetas y atributos.
- En los lenguajes de marcas **se siguen las reglas especificadas por el proveedor de datos**. Estas reglas se definen en los DTD, que son parte fundamental de un documento de marcado, pues indican las normas para el almacenamiento o mostrado de los datos.

1.2. Lenguajes de marcas en entornos web



Índice

Objetivos	3
Introducción.....	4
Metadatos.....	4
Instrucciones de proceso.....	4
Codificación de caracteres	5
Etiquetas o marcas	6
Elementos	6
Atributos.....	7
Comentarios	7
Arquitectura cliente/servidor	8
Aplicaciones web	8
Características	8
Arquitectura hardware cliente/servidor	9
HTTP	10
Tabla de métodos <i>request</i>	11
XML en el intercambio de información.....	13
Ficheros de configuración basados en XML.....	13
Despedida	14
Resumen.....	14

Objetivos

En esta unidad perseguimos los siguientes objetivos:

- Conocer la **estructura y los elementos de los documentos de los lenguajes de marcas.**
- Conocer la **arquitectura cliente/servidor.**

Introducción

Metadatos

Los metadatos se usan para **implementar una forma de etiquetado, catalogación, descripción y clasificación de los recursos existentes**. Suelen ser datos sobre datos.

Berners y Lee, en la definición que dan sobre este concepto, afirman que “los metadatos son información intelible para el ordenador sobre recursos web u otras cosas”.

En algunos casos pueden estar asociados a un esquema de descripción. También pueden aportar información sobre elementos de datos o atributos, información sobre la estructura de los datos, información sobre un aspecto concreto, etc.

En los lenguajes de marcado los **metadatos sirven para complementar la información** contenida en el documento de marcas o para **añadir información** que no va a ser visualizada, pero que le da contexto al documento, e incluso indica cómo se debe interpretar el documento.

En los documentos HTML, por ejemplo, se incluyen metadatos en la etiqueta `<head>`, tales como el título de la página, especificación de caracteres a usar, palabras clave, estilos, etc.

Instrucciones de proceso

Una instrucción de procesamiento o **instrucciones de proceso (PI)** le dan al intérprete, que es el encargado de presentar el documento, instrucciones de **cómo deben ser procesadas las marcas** que se incluyen en él.

Un ejemplo de instrucción de procesamiento sería en el DOM (Document Object Model) y se denominaría Node. **PROCESSING_INSTRUCTION_NODE**, y pueden ser utilizados en Xpath y XQuery con el comando “*processing-instruction()*”.

En SGML una “processing instruction” se encierra entre '<?' y '>'

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

```
<?xmlstylesheet type="text/css" href="style.css"?>
```

Codificación de caracteres

La codificación de caracteres consiste en la conversión de un carácter desde el lenguaje natural a un símbolo que pertenece a otro sistema de representación. Esta codificación se realiza mediante el uso de unas normas y reglas.

En este caso, la codificación serviría para **codificar caracteres** a través del lenguaje de marcas y su contenido de lenguaje escrito en los distintos alfabetos existentes.

Una de las primeras codificaciones fue el código **ASCII**, que con 8 caracteres codificó 128 caracteres del alfabeto latino.

Al confeccionar documentos HTML o XML se indica en el encabezado el juego de caracteres que queremos que use el navegador para decodificar la página o documento web.

La codificación de caracteres más moderna se realiza en **Unicode (UTF-8, UTF-16 o UTF-32)**, que es un estándar definido por la ISO10646.

Por ejemplo, si queremos usar el UTF-8, en una página HTML podríamos incluir en la cabecera de nuestro documento la siguiente directiva:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
```

En XHTML.

```
<meta charset="UTF-8"/>
```

En HTML5.

Caracteres especiales

Cuando se necesitó codificar caracteres especiales, o los caracteres de lenguajes de todo el mundo, y representarlos en HTML, se crearon unas codificaciones que hacían que el navegador o el intérprete del documento los reconociera como caracteres especiales.

Tabla ASCII

Aquí podemos ver una tabla de códigos ASCII para caracteres y caracteres especiales.

<https://ascii.cl/es/codigos-html.htm>

También hubo que codificar de manera especial los caracteres que formaban parte del lenguaje de marcas, como "<" y ">". Para ello se usaron **caracteres de escape**, que sirven para que el intérprete del documento conozca cuándo empieza y acaba la directiva o cuándo comienza el contenido.

Así, se crearon referencias específicas, de manera que esos caracteres pudiesen formar parte del contenido.

CÓDIGO	REPRESENTACIÓN
<	<
>	>
&	&
"	"

Referencias específicas para caracteres especiales.

Etiquetas o marcas

Una *tag* o etiqueta es la marca que usa el lenguaje para delimitar un objeto específico.

Las etiquetas o marcas están definidas por caracteres de escape que indican al software que lee el documento cuándo comienza y acaba el *tag* o etiqueta. **En lenguajes de marcado como HTML existen etiquetas o marcas que usan el carácter "<", el "\\" o el carácter ">"**.

En HTML y en XML casi todas las marcas indican el comienzo de un elemento y finalizan con otra marca parecida, que marca el fin del contenido donde aplicamos esa marca.



Elementos

Un elemento es un componente individual que comienza con una marca de inicio y acaba con una marca de final.

```
<etiqueta atributo1="valor1" atributo2="valor2">contenido</etiqueta>
```

Sintaxis de una etiqueta.

- **Elemento void:** solo contiene una etiqueta de entrada, lleva atributos y puede no contener ningún elemento hijo como texto.
- **Elementos de texto sin procesar (*raw text elements*):** se construyen con una etiqueta, atributos, algún contenido de texto, sin elementos, y con una etiqueta de fin.
- **Elementos normales:** se construyen con una etiqueta, atributos, algún contenido de texto y una etiqueta de fin.

Atributos

Los atributos sirven para **dotar al elemento de propiedades** que tienen que ver con la marca en la que están contenidos.

La forma de representar dichas propiedades es a través de pares nombre-valor, separados por un signo de igual "**=**". Normalmente se escriben después de la etiqueta de comienzo de un elemento.

El valor que va asociado al atributo suele ir encerrado entre comillas dobles o simples.

```
<etiqueta atributo1="valor1" atributo2="valor2">contenido</etiqueta>
```

Existe una **clasificación para los atributos:**

Atributos básicos

Se pueden utilizar prácticamente en todas las etiquetas HTML.

Atributos para internacionalización

Los utilizan las páginas que muestran sus contenidos en varios idiomas.

Atributos de eventos

Solo se utilizan en las páginas web dinámicas creadas con JavaScript.

Atributos de foco

Relacionados principalmente con la accesibilidad de los sitios web.

Comentarios

Los comentarios sirven para **incluir información extra que no es interpretada por el navegador** y que puede aparecer en cualquier parte de un documento. Para definirlos en XHTML se usa la siguiente etiqueta:

```
<!-- Esto es un comentario -->
```

Por razones de compatibilidad con algunos navegadores anteriores a 1995 **los contenidos de estilo y script se mantienen entre delimitadores de comentarios.**

No se permite ningún espacio en blanco entre el delimitador de apertura de declaración de etiqueta ("**<!**") y el delimitador de apertura de comentario ("**--**"), pero sí se permite entre el delimitador de cierre de comentario ("**--**") y el delimitador de cierre de declaración de etiqueta ("**>**").

Un error común es incluir una cadena de guiones ("---") dentro de un comentario.

Arquitectura cliente/servidor

Aplicaciones web

Las **aplicaciones web tienen su naturaleza en Internet**. Esto estructura su funcionamiento siguiendo un modelo **cliente/servidor**, en donde el servidor es multi-plataforma y el cliente, aunque soportado por varias marcas de navegador, es uni-plataforma.

Esto quiere decir que el estándar de la parte cliente, implantado a través de los navegadores, se unifica a través de HTML5, CSS3 y JavaScript, mientras que en el servidor viven distintas plataformas no estandarizadas, como Java, PHP, Python, CGI, .NET, etc.

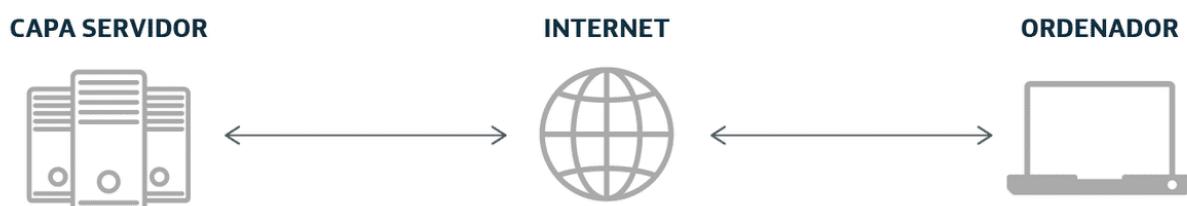
Siempre que vayamos a crear un software para desplegar en Internet deberá tener varios componentes que trabajarán en conjunto para satisfacer las necesidades y los objetivos de dicha aplicación. Tendremos en cuenta:

La interfaz del cliente

Soportará la parte visual e interactiva; en ella el usuario trabaja con contenidos, enlaces, formularios etc., y estos producirán información.

El módulo de servidor

Recibirá esa información y, según la lógica de la aplicación, lanzará procesos en bases de datos u otros módulos y dará una respuesta adecuada al cliente.



En esta arquitectura el cliente tiene la presentación visual y cierta lógica que se ejecuta en la parte de interfaz de usuario y emite peticiones a la parte servidor.

Características

En cuanto a la construcción de estas aplicaciones y su mantenimiento podemos indicar:

Son fáciles de mantener

Como cada aplicación tiene sus propios módulos y se programan abstrayéndose del resto de capas, el mantenimiento de cada una de las capas se realiza de manera independiente, por lo

que se pueden realizar mejoras y correcciones sin que afecte al resto de módulos de las otras capas.

Son fáciles de escalar

Por el mismo motivo, el escalamiento de la aplicación hacia afuera es razonablemente sencillo.

Son reusables

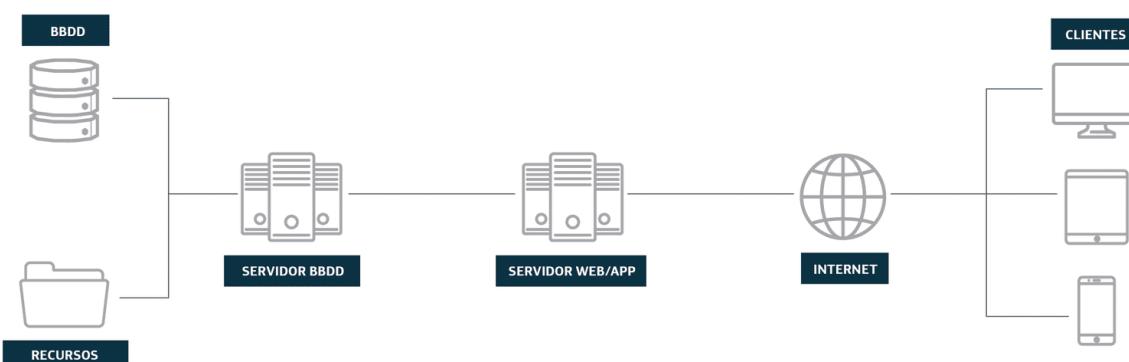
Los módulos se pueden construir para que puedan ser usados por otras aplicaciones o para la generación de nuevos módulos.

Disponibilidad

Al tener una estructura modular, la sustitución o puesta en marcha de partes de la aplicación es muy rápida, lo que deriva en su disponibilidad.

Arquitectura hardware cliente/servidor

Como hemos visto, un software para Internet debe tener varios componentes que trabajan en conjunto.



Base de datos

Tablas con los datos de la aplicación.

Gestor BBDD

El gestor de la base de datos es la aplicación encargada de procesar y responder las peticiones de información. Los más utilizados son los basados en SQL u Oracle.

HTTP

Servicio del servidor encargado de responder las peticiones del cliente. Los más extendidos son el servidor Apache para sistemas Linux, o IIS (Internet Information Service) para servidores Windows.

Este servicio devuelve los archivos procesados en el servidor al cliente, como los códigos HTML, CSS, etc.

Internet

La red.

Archivos

Archivos de imágenes, PDF, o librerías que la aplicación puede necesitar externamente.

Clients

Los clientes envían las peticiones mediante URL e interpretan los archivos devueltos.

Normalmente son archivos HTML junto a otros como las hojas de estilo CSS, códigos JavaScript, imágenes, etc.

HTTP

Cuando el cliente hace una **petición** al servidor, este le devuelve el mensaje con la información solicitada. Para esta comunicación, en la mayoría de los casos se usa el **protocolo HTTP** a través de *request-response*.

Veamos el proceso.

1. **Un cliente HTTP inicia una solicitud** mediante el establecimiento de una conexión de Protocolo de Control de Transmisión (TCP) a un puerto determinado en un servidor (normalmente el puerto 80).
2. **Hay un servidor HTTP escuchando en ese puerto** en espera de un mensaje de petición de un cliente.
3. Al recibir la petición **el servidor envía una línea de estado, como "HTTP/1.1 200 OK"**, y un mensaje de su cuenta.



El contenido que se envía en un mensaje HTTP es:

- Una línea de petición.
- Encabezado *request*.
- Una línea vacía.
- Un cuerpo del mensaje opcional.

La línea de solicitud y los encabezados de todos deben terminar con `<CR> <LF>`, es decir, un carácter de retorno de carro seguido de un carácter de salto de línea.

La línea vacía debe consistir solo en `<CR> <LF>` y ningún otro espacio en blanco.

```
GET /index.html HTTP/1.1 Host: www.example.com
```

Ejemplo de petición.

Contenido de un mensaje *response*:

- Una línea de estado.
- Encabezado *request*.
- Una línea vacía.
- Un cuerpo del mensaje opcional.

La línea de estado y los encabezados de todos deben terminar con `<CR> <LF>`, es decir, un carácter de retorno de carro seguido de un carácter de salto de línea. La línea vacía debe consistir solo de `<CR> <LF>` y ningún otro espacio en blanco.

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
ETag: "3f80f-1b6-3e1cb03b"
Content-Type: text/html; charset=UTF-8
Content-Length: 131
Connection: close
<html>
<head> <title>An Example Page</title> </head>
<body> Hello World, this is a very simple HTML document. </body>
</html>
```

Ejemplo de un contenido mensaje.

Tabla de métodos *request*

HTTP tiene un conjunto de métodos para realizar las peticiones e indicar la acción que se desea realizar para un recurso determinado. Estos métodos implementan una semántica diferente, pero características similares: *Get, Head, Post, Put, Delete, Trace, Options, Connect*.

Veamos cada uno de ellos:

GET

Solicita una **representación del recurso especificado**.

Por seguridad no debería ser usado por aplicaciones que causen efectos, ya que transmite información a través de la URI agregando parámetros a la URL.

HEAD

Solicita una **respuesta idéntica a la que correspondería a una petición GET**, pero sin el cuerpo de la respuesta.

Esto es útil para la recuperación de meta-information escrita en los encabezados de respuesta, sin tener que transportar todo el contenido.

POST

Solicita que se sometan los datos a ser procesados para el recurso identificado. **Los datos se incluirán en el cuerpo de la petición.**

Esto puede derivar en la creación de un nuevo recurso, en las actualizaciones de los recursos existentes o en ambas cosas.

PUT

Sube, carga o realiza un upload de un recurso especificado (archivo). Es el camino más eficiente para subir archivos a un servidor, porque en POST utiliza un mensaje multiparte y el mensaje es decodificado por el servidor.

DELETE

Elimina el recurso especificado.

TRACE

Este método solicita al servidor que **envíe de vuelta un mensaje de respuesta**, en la sección del cuerpo de entidad, con todos los datos que reciba del mensaje de solicitud.

Se utiliza con fines de comprobación y diagnóstico.

OPTIONS

Devuelve los **métodos HTTP que el servidor soporta para una URL específica.**

Puede usarse para comprobar la funcionalidad de un servidor web mediante petición, en lugar de un recurso específico.

CONNECT

Se utiliza para saber si se **tiene acceso a un host**, y la petición no necesariamente llega al servidor.

Este método se utiliza principalmente para saber si un *proxy* nos da acceso a un *host* bajo condiciones especiales, como por ejemplo "corrientes" de datos bidireccionales encriptadas (como lo requiere SSL).

XML en el intercambio de información

El estándar **XForms** se basa en dividir el formulario en tres partes:

1. El modelo de XForms.
2. Los datos.
3. La interfaz de usuario.

Por lo tanto, se separan claramente los datos de la visualización, lo que permite una gran flexibilidad de opciones en la presentación.

En los archivos **HTML o XHTML se definirán, en la cabecera, el tipo de XForms** a utilizar y en el cuerpo, los controles que se utilicen.

El fichero aparte XML describe el formulario que permite modificar un gran número de opciones de los controles implementados en el HTML. Cuando el formulario se debe enviar, el procesador de XForms es el encargado de recoger los datos y enviarlos en forma de archivos XML. Esta separación hace que los formularios sean reutilizables, ya que no están ligados al documento HTML. Además, al definirse en XML, los formularios no solo son aplicables a HTML o XHTML, sino a cualquier lenguaje de marcado.

Ficheros de configuración basados en XML

En muchas aplicaciones, ya sea web o de escritorio, se utilizan ficheros XML para configurar los elementos personalizables de las aplicaciones.

Un ejemplo de esto es el fichero de configuración de un servidor web mediante la tecnología J2EE. En este archivo de configuración (*web.xml*) se describe el contenido de despliegue de la aplicación web, las páginas web, *sevlets*, páginas *jsp*, etc.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <servlet>
    <servlet-name>HolaMundoServlet</servlet-name>
    <servlet-class>org.prueba.servlet.HolaMundoServlet</servlet-class>
  </servlet>
```

Otro ejemplo es el *framework .NET*, en el que se puede configurar el comportamiento de la aplicación así como el motor de ejecución en sí con archivos XML (*web.config*).

Aunque es posible editar estos archivos para configurar la aplicación, contienen librerías especializadas para extraer y modificar los datos de configuración de los archivos XML.

Despedida

Resumen

Has terminado la lección. Repasemos los puntos más importantes que hemos tratado.

- Existen diferentes elementos que conforman un documento realizado con lenguajes de marcas. Son las **etiquetas**, los **elementos**, los **atributos** y los **comentarios**.
- Cada uno de ellos tiene una funcionalidad distinta y su propia sintaxis.
- Las **aplicaciones web y el protocolo de comunicación que utilizan** para la transmisión de datos (HTTP) son herramientas fundamentales de los lenguajes de marcas en los entornos web.

1.3. Gramáticas



Índice

Objetivos	3
Gramáticas	4
Definición	4
XML Schema.....	5
Ventajas de los XML Schema.....	7
Reglas gramáticas XML	7
Despedida	8
Resumen.....	8

Objetivos

- Conocer el significado de la gramática para los lenguajes de marcas.
- Conocer cómo se define una gramática.
- Conocer la gramática de HTML.
- Conocer la gramática de XML.

Gramáticas

Definición

Las gramáticas en los documentos de marcado de tipo electrónico como HTML, XML, etc., vienen conformadas por las **reglas de escritura** de este tipo de documentos más las restricciones e indicaciones de la DTD o XML Schema pertinente.

Todo documento de un lenguaje de marcas tiene en común una gramática que define el marcado permitido en esa clase, el marcado requerido y cómo debe ser utilizado dicho marcado en la instancia del documento.

Según el diccionario de la lengua española, gramática significa:

"Parte de la lingüística que estudia los elementos de una lengua, así como la forma en que estos se organizan y se combinan."

DICCIONARIO DE LA LENGUA ESPAÑOLA

Sin embargo, en relación a los lenguajes de marcas, la gramática permite reconocer si la **representación de marcas de un documento específico es correcta y cumple las reglas** que permiten que este pueda ser interpretado por un programa externo y mostrado de manera correcta.

La gramática para documentos HTML, por ejemplo, es un **conjunto de reglas que indican el orden de los elementos del lenguaje**. Estos elementos del lenguaje se pueden dividir en dos grupos: terminales (las propias palabras de la lengua) y no terminales (todas las demás reglas gramaticales).

En HTML las palabras se corresponden con las etiquetas de marcado incrustadas y el texto en un documento.

El análisis que se realiza para un documento se basa en las reglas de sintaxis por las que se rige el documento, es decir, el lenguaje o el formato en el que está escrito. Todos los formatos que se pueden analizar deben tener una gramática determinista formada por un vocabulario y unas reglas de sintaxis. Esto se denomina **gramática libre de contexto**. Los lenguajes humanos no son de este tipo y, por tanto, no se pueden analizar con técnicas de análisis convencionales.

Reglas gramáticas para HTML

El estándar define esta gramática mediante la **DTD (Definición de Tipo de Documento)** que establece las reglas de formación del lenguaje formal, es decir, qué combinaciones de símbolos elementales son sintácticamente correctas.

En la DTD se identifica la estructura del documento, es decir, aquellos elementos que son necesarios en la elaboración de un documento o un grupo de documentos estructurados de manera similar. Contiene las reglas de dichos elementos: el nombre, su significado, dónde pueden ser utilizados y qué pueden contener.

La entidad **W3C** ha definido los estándares HTML y otros desde el lenguaje SGML a través de las DTD. La construcción de una DTD se revisará en otra lección.

Estas DTD se aproximan al HTML 4 DTD. El W3C recomienda utilizar las versiones autorizadas de estas DTD en sus identificadores de sistema, definidos al validar el contenido. Si es necesario utilizar estas DTD localmente se debe descargar uno de los archivos de esta versión.

Toda DTD debe tener un, y solo un, elemento raíz, también conocido como elemento documento. Este elemento raíz debe coincidir con el nombre que aparece a continuación del DOCTYPE.

Un documento DTD puede contener:

- Declaraciones de elementos.
- Declaraciones de atributos para un elemento.
- Declaraciones de entidades.
- Declaraciones de notaciones.
- Instrucciones de procesamiento.
- Comentarios.
- Referencias a entidades de parámetro.

XML Schema

Para definir la **gramática de los documentos XML se puede usar DTD o XML Schema**, que es una alternativa más potente a las DTD.

XML Schema permite escribir esquemas detallados para documentos XML utilizando la sintaxis estándar de XML. XML Schema describe la estructura de un documento XML. El lenguaje XML Schema también se denomina XML Schema Definition (XSD).

El objetivo de un XML Schema es definir los elementos que permiten construir un documento XML válido, igual que las DTD.

Un XML Schema define:

- Los elementos que pueden aparecer en un documento.
- Atributos que pueden aparecer en un documento.
- Qué elementos son elementos hijos.
- El orden de los elementos hijos.
- El número de elementos hijos.
- Si un elemento debe estar vacío o puede incluir texto.
- Los tipos de datos de sus elementos y atributos.
- Los valores por defecto y fijos para elementos y atributos.

La gramática en los XML Schema está escrita también en XML, por lo que:

1. No hay necesidad de aprender un lenguaje nuevo.
2. Se puede utilizar un editor de XML para editar el XML Schema.
3. Se puede utilizar un procesador de XML para procesar un XML Schema.
4. Se puede manipular un XML Schema utilizando XML DOM.
5. Se puede transformar un XML Schema utilizando XSLT.

Cuando se transmite información desde un emisor a un receptor es necesario que ambas partes conozcan el mismo protocolo de comunicación. Con XML Schema el emisor puede describir los datos de manera que el receptor lo entienda.

Una fecha como: "03-11-2006", dependiendo del país, puede ser interpretada como "3 de noviembre" y en otros países como "11 de marzo". Con lo cual, al definir un elemento de la siguiente manera:

```
<fechalinicio type="date">2006-03-11</fechalinicio>
```

Se asegura un mutuo entendimiento sobre el contenido, porque el tipo de dato "date" requiere el formato "YYYY-MM-DD".

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://dominio.com">

<xs:element name="Contacto">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="dni" type="xs:string"/>
      <xs:element name="telefono" type="xs:string"/>
      <xs:element name="descripcion" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Ejemplo XML Schema.

Ventajas de los XML Schema

Ventajas

Las ventajas de los XML Schema son:

- Los XML Schema son **extensibles**.
- Los XML Schema son **más ricos semánticamente que las DTD**.
- Los XML Schema están **escritos siguiendo la sintaxis estándar de XML**.
- Los XML Schema soportan **tipos de datos**.
- Los XML Schema soportan **namespaces**.

La característica más importante de los XML Schema es que soportan tipos de datos. Los tipos de datos permiten:

- Describir qué **elementos están permitidos**.
- **Validar** si los datos son correctos.
- Trabajar con los datos de una base de datos.
- Definir **Facets** (restricciones).
- Definir **patrones** de datos (formatos).
- Convertir datos considerando diferentes tipos de datos.

Reglas gramáticas XML

Además de las reglas definidas en cada XML Schema, los XML deben cumplir las siguientes normas para que puedan ser entendibles por sistemas externos:

1. Empezar con la declaración XML.
2. Tener un único elemento raíz.
3. Las etiquetas de apertura deben tener sus correspondientes etiquetas de cierre.
4. Los elementos son "case-sensitive" (sensibles a mayúsculas y minúsculas).
5. Todos los elementos deben cerrarse.
6. Todos los elementos tienen que estar adecuadamente anidados.
7. Los atributos deben estar entre comillas.
8. Se deben utilizar las entidades para utilizar caracteres especiales.

```
<?xml version="1.0"?>
<nota>
<a>Pedro</a>
<de>María</de>
<cabecera>A su atención</cabecera>
<cuerpo>¡En breve nos vemos en la cita prevista!</cuerpo>
</nota>
```

Despedida

Resumen

Has finalizado esta unidad, veamos los aspectos más importantes que hemos tratado.

En todo lenguaje necesitamos de unas reglas para transmitir la información deseada. Estas reglas son las que definen la estructura del propio lenguaje.

En esta unidad hemos visto las reglas gramáticas que tenemos que cumplir en los documentos XML y HTML.

En XML estas reglas están definidas por la estructura definida en su diseño y además podemos añadir las normas que definamos nosotros en la construcción de un DTD o XML Schema.

```
<xml> ::= <elemento> FIN_TOKENS
<elemento> ::= <inicio_elemento> <atributos> <cierres>
<inicio_elemento> ::= '<' TOKEN_IDENTIFICADOR
<atributos> ::= <atributo><atributos> | vacío
<atributo> ::= TOKEN_IDENTIFICADOR '='
TOKEN_CADENA
<cierres> ::= '/>' | '>' <mas_elementos> <fin_elemento>
<mas_elementos> ::= <elemento> <mas_elementos> | 
vacío
<fin_elemento> ::= '</>' TOKEN_IDENTIFICADOR '>'
```

Gramática XML simple.

En el caso de HTML las normas ya nos vienen fijadas por la propia DTD establecida por la organización W3C y las propias del los lenguajes XML del que procede.

1.4. Herramientas de edición de código



Índice

Objetivos	3
Introducción.....	4
Editores de código	5
Características de los editores.....	5
Funciones de un editor de código	6
Tipos de editores	6
Herramientas de diseño web	9
Sublime Text.....	9
Atom	9
Komodo Edit.....	10
Netbeans	11
Despedida	13
Resumen.....	13

Objetivos

Con esta unidad perseguimos los siguientes objetivos:

- Conocer las ventajas que proporcionan los editores de código o IDE.
- Conocer el proceso de diseño web.
- Conocer los tipos de herramientas que se utilizan en el diseño web.
- Conocer los editores más utilizados.
- Conocer las herramientas de creación disponibles para el diseño web.

Introducción

Es hora de conocer las fases básicas que sigue el desarrollo web y algunas de las herramientas que existen para trabajar con lenguajes de marcas y diseño web.

El desarrollo y diseño web es un proceso en el que se emplea un gran número de herramientas. Se utilizan herramientas para el **diseño**, para la **maquetación**, para la **programación** y también para la **depuración**.

Estas herramientas van desde el sistema operativo hasta el comando más insignificante, y por ello debemos elegir la más adecuada a nuestras necesidades y capacidades.

Pero para poder hacerlo, antes debemos identificar las fases del proceso que forman el ciclo de vida de un desarrollo web.

Diseño

Las más famosas son las herramientas de Adobe: Photoshop, Illustrator y Fireworks, que es una mezcla entre las dos anteriores. Estas herramientas posibilitan la creación de gráficos, edición de imágenes, diseño gráfico, fotografía, impresión, etc.

Maquetación

Estas herramientas comprenden un conjunto de lenguajes como JavaScript , Java, HTML/XHTML y CSS, entre otros.

Programación

Cuando hablamos de programación en el "lado del servidor" nos referimos al que se ejecuta en el servidor web justo antes de que se envíe al cliente. Entre los lenguajes más utilizados se encuentran PHP, PERL, JSP y ASP.NET.

Depuración

En la fase de depuración se somete a diferentes tipos de pruebas el código creado para detectar errores. Este proceso se puede realizar una vez terminada nuestra aplicación, aunque la mayoría de los entornos de desarrollo, nos proporcionan asistentes para detectarlos durante la implementación

Editores de código

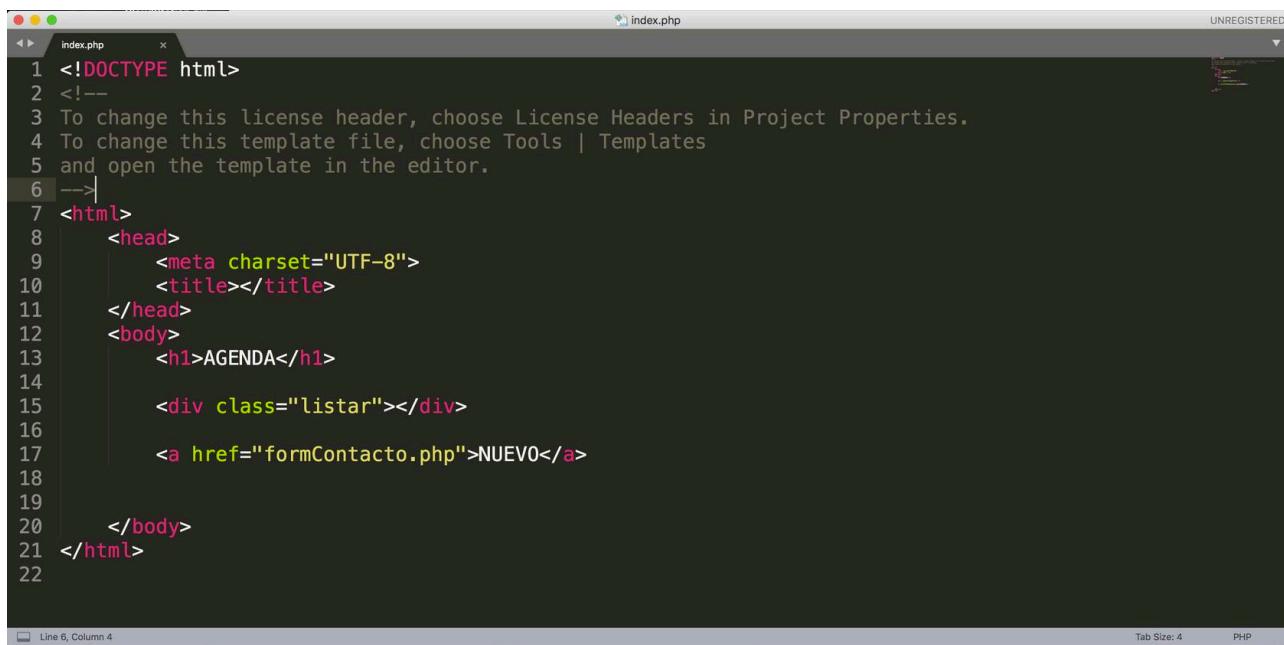
Características de los editores

Para crear nuestros documentos, ya sean códigos de lenguajes de marcas, como XML o HTML, u otros lenguajes, como JavaScript o PHP, lo único que necesitamos es un editor de texto plano.

Hoy en día, dada la complejidad de las aplicaciones, existen multitud de editores de texto cuyas características nos ayudan a la hora de escribir el código. Esto hace que la tarea de escribir, visualizar o probar nuestras estructuras sea algo más fácil.

Un editor es capaz de reconocer, resaltar y cambiar los colores de las variables, las cadenas de caracteres, las palabras reservadas, las instrucciones, el inicio y fin de los corchetes, etc. De esta manera el código fuente será mucho más visual, cómodo y se podrán reconocer los errores a simple vista.

Una parte importante del editor es que, no solamente nos corregirá en caso de que detecte un error, sino que nos mostrará sugerencias e indicaciones del posible fallo que estemos cometiendo.



```
index.php
1 <!DOCTYPE html>
2 <!--
3 To change this license header, choose License Headers in Project Properties.
4 To change this template file, choose Tools | Templates
5 and open the template in the editor.
6 -->
7 <html>
8   <head>
9     <meta charset="UTF-8">
10    <title></title>
11  </head>
12  <body>
13    <h1>AGENDA</h1>
14
15    <div class="listar"></div>
16
17    <a href="formContacto.php">NUEVO</a>
18
19
20  </body>
21 </html>
```

Código HTML en Sublime Text.

Funciones de un editor de código

Según el lenguaje de programación, los requisitos que precisamos de un entorno de desarrollo pueden ser diferentes.

Consideramos que **un buen entorno de desarrollo** debe tener las siguientes características:

- Ser **multiplataforma**.
- Soportar **diversos lenguajes de programación**.
- Integrarse con **sistemas de control de versiones**.
- Tener **reconocimiento de sintaxis**.
- Implementar **extensiones y componentes para el IDE**.
- Permitir **integración con frameworks** populares.
- Tener **depurador**.
- **Importar y exportar proyectos**.
- Tener opción para **múltiples idiomas**.
- Ofrecer **manual de usuario y ayuda**.

Tipos de editores

No es fácil elegir un editor de código, ya que depende de los lenguajes a utilizar y de los servicios que necesitemos.

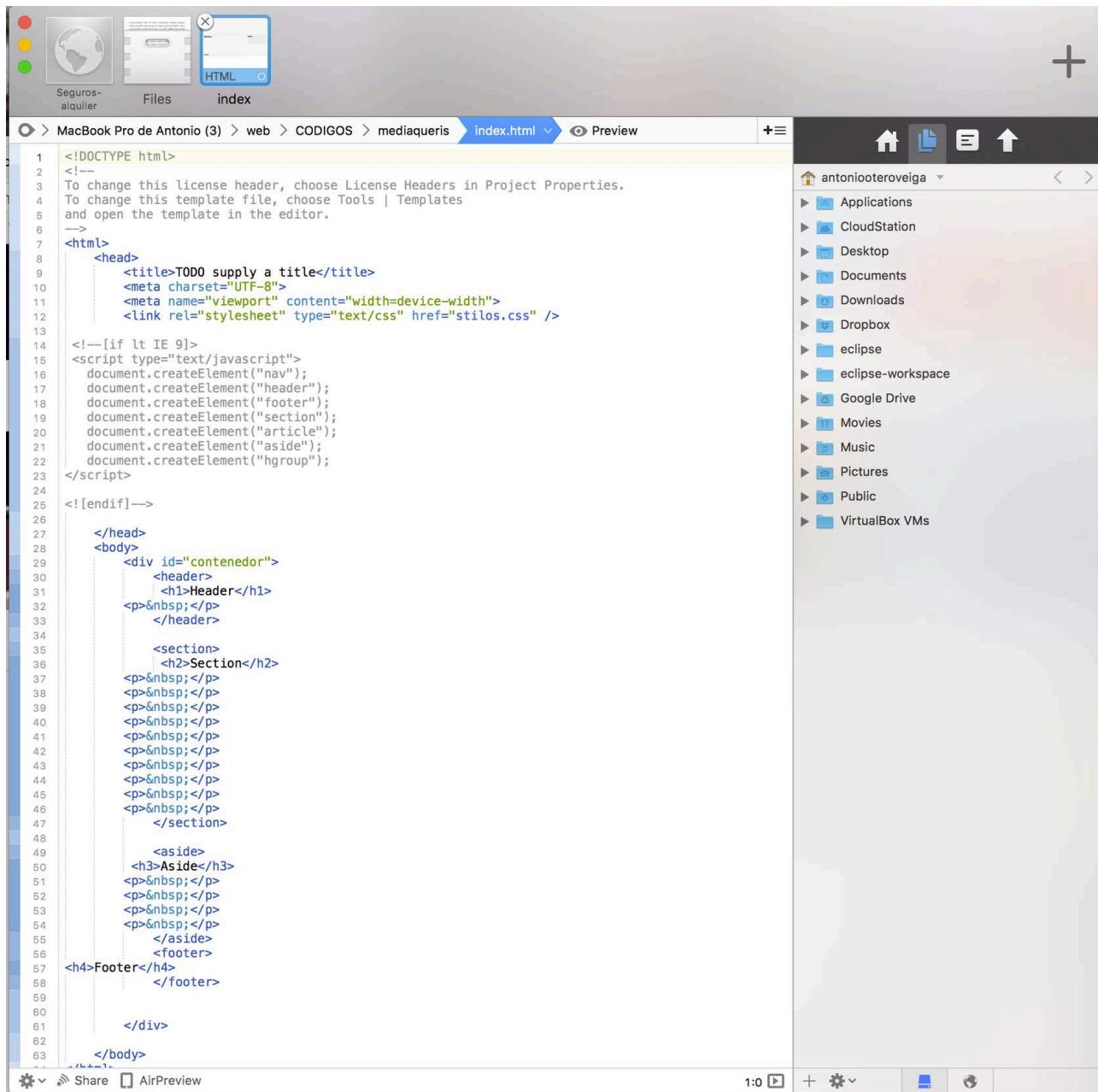
Puede ser que estemos realizando una aplicación web con bases de datos y necesitemos un sistema de control de versiones, o que sea capaz de transmitir ficheros vía FTP.

La variedad de estas herramientas es muy numerosa. En cada alternativa encontramos muchas de estas opciones integradas en la instalación básica, o bien la posibilidad de agregar funcionalidades mediante la instalación de *plugins*.

Una de las grandes preguntas que todo desarrollador se hace al iniciar su vida profesional es, ¿qué es mejor, un IDE o un editor de texto? ¿Cuál debo utilizar? Son preguntas difíciles de responder, pues depende de los recursos que necesitemos para trabajar.

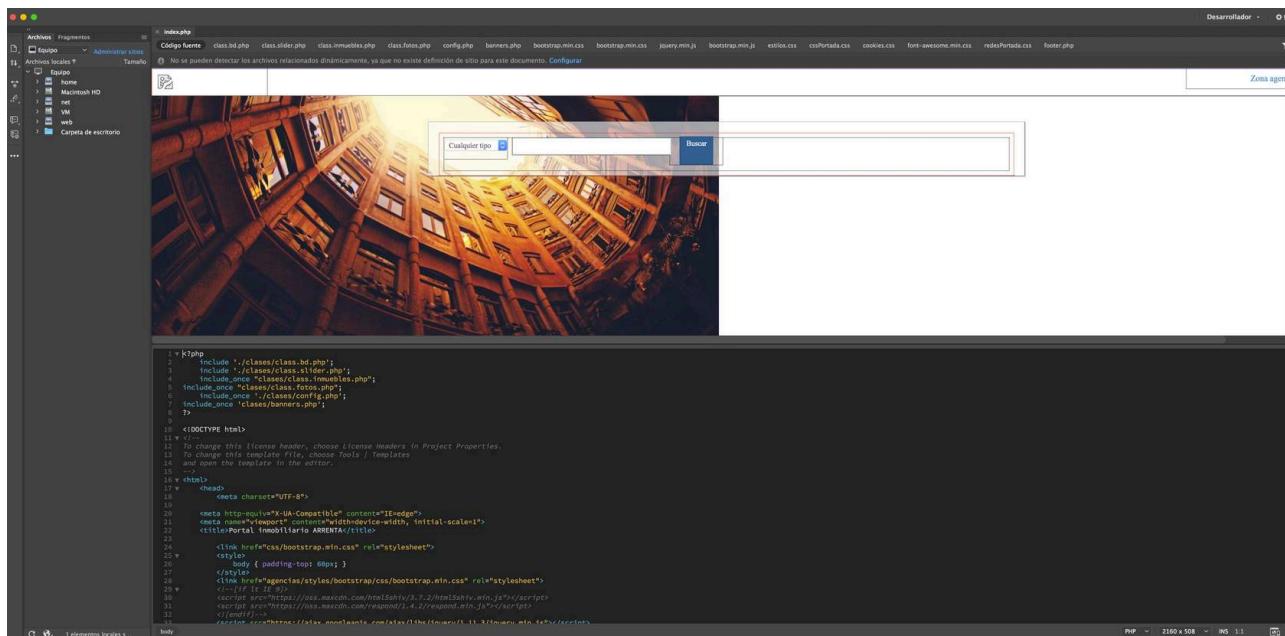
Podemos realizar una **clasificación de editores de código** según sus capacidades.

Editores de texto plano. Son programas que permiten escribir documentos en formato TXT y tabular o indentar el código. Estos editores reconocen las palabras reservadas del lenguaje en cuestión y realizan una codificación de colores para facilitarnos su lectura.



Captura del editor CODA 2.

Editores web WYSIWYG "What You See Is What You Get" ("Lo que ves es lo que obtienes"). Estos editores, además de los asistentes de escritura de código, nos permiten visualizar los resultados y maquetar o modificar elementos de forma visual. Un ejemplo de este tipo de editores es *Adobe Dreamweaver* o *Expression Web* de Microsoft.



Captura de Adobe Dreamweaver.

Entornos de desarrollo. Este tipo de herramientas son las más completas, aunque no por ello tiene que ser nuestra elección principal, pues depende del uso que le vayamos a dar.

Un entorno de desarrollo integrado (**IDE**) es una aplicación que nos ayuda en la creación de nuestros proyectos de software. Es una aplicación pensada para facilitarnos la escritura por medio de asistentes y capaz de realizar comprobaciones de que todo esté correcto. Es decir, es un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI).

Los IDE proporcionan un entorno de trabajo amigable para la mayoría de los lenguajes existentes hoy en día. Sin ellos la labor de realizar las aplicaciones que nos demandan actualmente sería mucho más difícil de realizar.

Un IDE nos permite escribir el código de una forma sencilla, resaltando la sintaxis. Posee un corrector sintáctico, y normalmente también dispone de un compilador y/o intérprete, y de un depurador, entre otras funcionalidades.



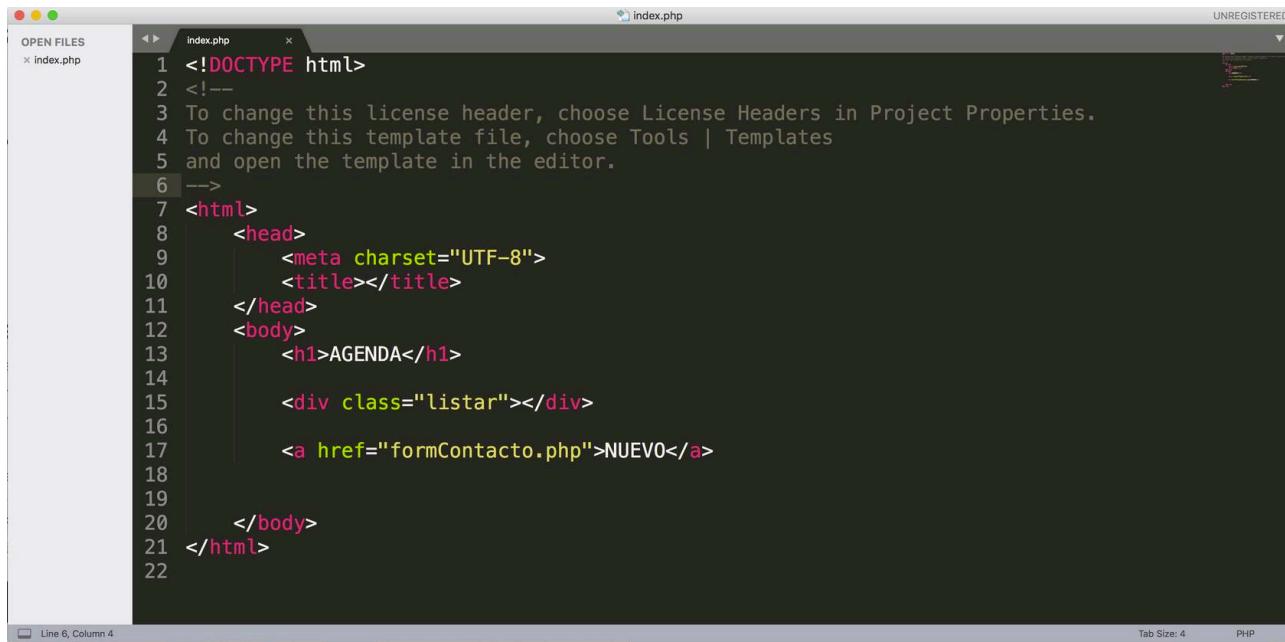
Herramientas de diseño web

Sublime Text

Sublime Text es un editor de texto avanzado y multiplataforma. Su instalación básica ofrece lo indispensable para trabajar con múltiples lenguajes de programación.

Pero Sublime Text nos da, además, la posibilidad de instalar **plugins** para darle mayor versatilidad. Permite **trabajar con varios documentos a la vez** mediante pestañas. El resultado de sintaxis soporta un gran número de lenguajes (C, C++, C#, CSS, D, Erlang, HTML, Groovy, Haskell, HTML, Java, JavaScript, LaTeX, Lisp, Lua, Markdown, Matlab, OCaml, Perl, PHP, Python, R, Ruby, SQL, TCL, XML).

Aunque se presenta como un programa de pago, se puede descargar una versión de prueba, plenamente funcional y sin limitación de tiempo.



```
<!DOCTYPE html>
<!--
3 To change this license header, choose License Headers in Project Properties.
4 To change this template file, choose Tools | Templates
5 and open the template in the editor.
-->
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <h1>AGENDA</h1>
    <div class="listar"></div>
    <a href="formContacto.php">NUEVO</a>
</body>
</html>
```

Sublime Text

Un sofisticado editor de texto para código, marcado y prosa. <https://www.sublimetext.com/>

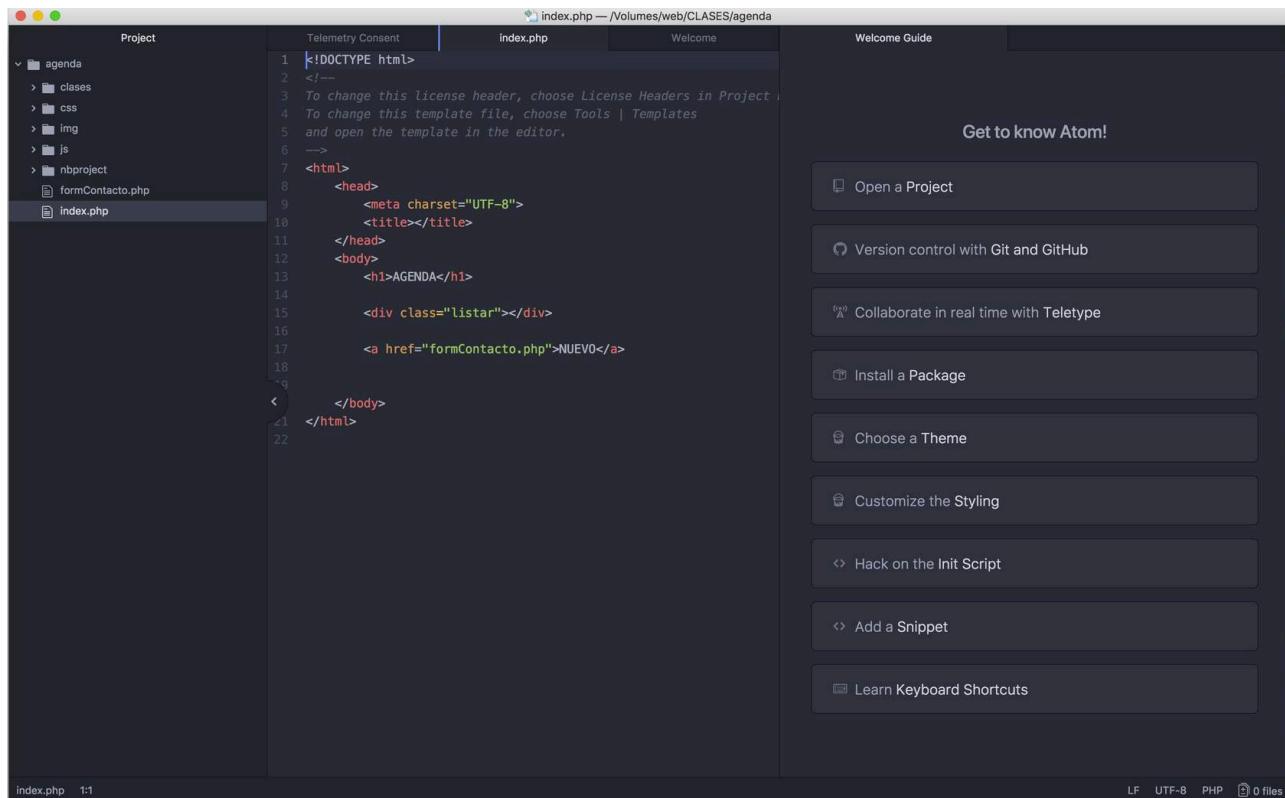
Atom

La principal característica de este editor de texto es la de ser fácilmente "hackeable", de forma que cualquier programador puede modificar el código y adaptarlo a sus gustos y necesidades.

Atom ha sido desarrollado por el equipo de *GitHub*. Es de "**código abierto**" y nos ofrece todas las posibilidades de resaltado y asistencia de escritura de código. Otra ventaja de este editor,

además de su facilidad de integración con *Github*, es la posibilidad de trabajar mediante proyectos, lo que nos facilita enormemente el trabajo con archivos.

Es multiplataforma, es decir, que podemos instalarlo tanto en Windows como en Mac OS.



Atom

El editor desarrollado por GitHub y "hackeable". <https://atom.io/>

Komodo Edit

Komodo es un editor que soporta varios lenguajes de programación: PHP, Python, Perl, Ruby, C++, HTML, Java, JavaScript, Django, CSS, HTML, etc.

Además, Komodo ofrece asistencia de escritura, como la autotabulación, el control de versiones, etc.

Incorpora un sistema de complementos o *add-ons* similar al del navegador Firefox para implementar nuevas funcionalidades.

Y es **multiplataforma**, por lo que podemos utilizarlo tanto en Windows como en Mac OS o Linux. Está distribuido bajo licencia GPL.

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="utf-8">
    <title>Explorador de archivos en PHP</title>
    <style>
        section div {clear:both;}
        .group {overflow:hidden;padding:2px;}
        section .group:nth-child(odd) {background:#e5e5e5;}
        .directory {font-weight:bold;}
        .name {float:left;width:25px;overflow:hidden;}
        .link {float:left;margin-left:10px;}
        .size {float:right;}
        .bold {font-weight:bold;}
        .footer {text-align:center;margin-top:20px;color:#808080;}
    </style>
</head>
<body>
<?php
// obtenemos la ruta a revisar, y la ruta anterior para volver...
if($_GET["path"]){
    $path=$_GET["path"];
    $back=implode("/",explode("/",$path,-2));
    if($back)
        $back.= "/";
    else
        $back="";
} else {
    $path="";
}
?>
<nav>
    <h2><?php echo $path?></h2>
</nav>
<section>
    <?php
        // si no estamos en la raiz, permitimos volver hacia otras
        if($path!="")
            echo "<div class='bold group'><a href=?path=$back.">...</a></div>";
        // devuelve el tipo mime de su extensión (desde PHP 5.3)
        $finfo1 = finfo_open(FILEINFO_MIME_TYPE);
        // devuelve la codificación mime del fichero (desde PHP 5.3)
        $finfo2 = finfo_open(FILEINFO_MIME_ENCODING);
        $folder=0;
        $file=0;
        # recorremos todos los archivos de la carpeta
        foreach(glob($path) as $filename)
        {
            $fileInfo=finfo_file($finfo1, $filename);
            $fileEncoding=finfo_file($finfo2, $filename);
            if($fileInfo=="directory")
            {
                $folder++;
                // mostramos la carpeta y permitimos pulsar sobre la misma
                echo "<div class='directory group'>";
            }
        }
    </?php
}

```

Captura de Komodo Edit.

Komodo Edit

El editor "políglota". <https://www.activestate.com/komodo-ide/downloads/edit>

Netbeans

Dentro del entorno académico Netbeans es uno de los editores más utilizados.

Con él podemos realizar codificaciones en PHP, C++ y lenguajes web, aunque es sobre todo muy utilizado para el lenguaje Java. Ofrece la posibilidad de ampliar sus funciones instalando *plugins*.

Dependiendo del lenguaje de programación nos ofrece unas características u otras, pero sus **características principales** son:

- Ser multiplataforma.
- Soporte para diversos lenguajes de programación.
- Integración con sistemas de control de versiones.
- Reconocimiento de sintaxis.
- Extensiones y componentes para el IDE.
- Integración con frameworks populares.

- Depurador.
- Importar y exportar proyectos.
- Múltiples idiomas.
- Manual de usuario y ayuda.

The screenshot shows the NetBeans IDE interface with the title bar "NetBeans IDE 8.2". The main window displays an HTML file titled "evento_raton.html". The code in the editor is as follows:

```
texto5 = document.getElementById("desplaza") //scroll de la página
texto5.innerHTML = "Desplazamiento de la página: " + despx + ", " + despy
}
}
function mover(ev){ //control del movimiento del cuadrado
var evento2 = ev || window.event //distinguir acción del ratón
tip = evento2.type
if (tipo == "click") { //coger el cuadrado (un click)
estado = "mover"
}
else if (tipo == "dblclick") { //soltar el cuadrado (doble click)
estado = "parar"
}
}
window.onload = function(){ //provocar los eventos
document.body.onmousemove = info
document.getElementById("caja").onclick = mover;
document.getElementById("caja").ondblclick = mover;
}
</script>
</head>
<body>
<h3>Arrastra este cuadrado: Haz un click encima de él para cogerlo y doble click para soltarlo.</h3>
<div id="caja"></div>
<br/><br/><br/><br/>
<div id="textos">
<p id="posicion">Coordenadas de la página: </p>
<p id="ventana">Coordenadas de la ventana: </p>
<p id="pantalla">Coordenadas de la pantalla:</p>
<p id="accion"> Estado: </p>
<p id="desplaza" ></p>
</div>
<br/><br/><br/><br/><br/><br/>
<p>Parrafo para alargar la página.</p>
<br/><br/><br/><br/>
<p>Parrafo para alargar la página.</p>
</body>
</html>
```

The code includes JavaScript for mouse events (mover, click, dblclick) and HTML for a drag-and-drop example and some placeholder text.

Captura de Netbeans.

Netbeans

Desarrolla rápida y fácilmente aplicaciones de escritorio, móviles y web con Java, JavaScript, HTML5, PHP, C / C ++ y más. <https://netbeans.org/>

Despedida

Resumen

Has terminado la lección. Vamos a recordar los puntos más importantes que hemos tratado.

- Para desarrollar nuestros códigos basta un simple editor de texto plano, como el bloc de notas de Windows. Pero **la creación de aplicaciones web es una tarea mucho más compleja, imposible de abordar con un editor de texto plano.**
- Teniendo en cuenta, además, que no solo tendremos que trabajar con un lenguaje de programación, hay que tener **herramientas que nos asistan en la escritura y nos avisen de posibles errores**, ya que resulta extremadamente útil para aumentar nuestra productividad.
- Así, hemos visto la **utilidad de los editores de código** y analizado alguno de ellos.
- Existe una **gran cantidad de alternativas en cuanto a editores de código**, por lo que nos hemos centrado en **los más utilizados**.
- La **elección del editor o IDE adecuado** a nuestras necesidades dependerá de los recursos que demande nuestro proyecto y de las preferencias personales de cada programador.

MP_0373. Lenguajes de marcas y sistemas de gestión de información

UF2. Lenguajes para la visualización de la información

2.1. Modelo de objetos DOM



Índice

Objetivos siguientes objetivos:.....	3
DOM.....	4
Modelo de Objetos del Documento (DOM)	4
Definición del árbol	5
Tipos de nodos.....	6
Attribute.....	6
Jerarquía de nodos.....	8
Clasificación de objetos en HTML	8
Despedida	10
Resumen.....	10

Objetivos

En esta unidad perseguimos los siguientes objetivos:

- Conocer qué es Modelo de Objetos del Documento (DOM)
- Conocer de dónde viene y para qué sirve el Modelo de Objetos del Documento.
- Conocer cuáles son sus entidades.
- Conocer la arquitectura del DOM.

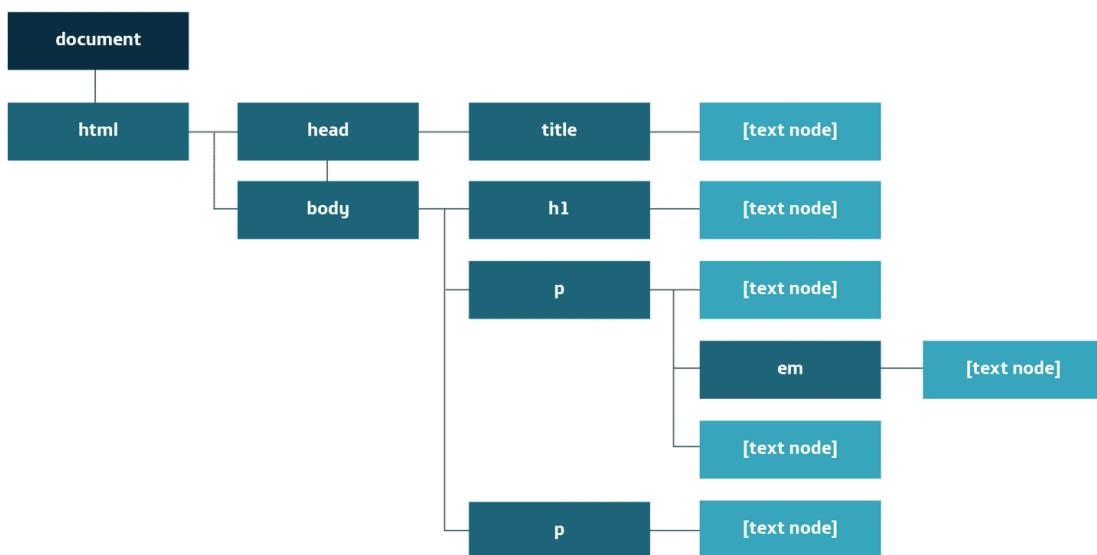
DOM

Modelo de Objetos del Documento (DOM)

El **Document Object Model** (DOM) es el árbol creado por las aplicaciones para representar la estructura de los documentos de tipo HTML y XML.

Proporciona una representación estructurada del documento, define su estructura creando diferentes nodos de forma jerarquizada. De este modo el programador es capaz de acceder a ellos pudiendo modificarlos, añadir nuevos, etc., dando dinamismo a la visualización del documento.

Cuando creamos un documento XML o HTML cada etiqueta es un nodo o un objeto del DOM, permitiendo a otros lenguajes acceder a ellos, identificando su tipo y asignando diferentes métodos o atributos. El lenguaje más extendido para trabajar con el DOM es JavaScript.



El **DOM** se originó como una especificación para permitir que los programas Java y los *scripts* de JavaScript fueran portables entre los navegadores web.

El "HTML Dinámico" fue el ascendiente inmediato del Modelo de Objetos del Documento; originalmente se pensaba en él principalmente en términos de navegadores.

Cuando se formó el grupo de trabajo DOM en el **W3C** también se unieron a él compañías de otros ámbitos, incluyendo los de la edición y archivo de documentos HTML y XML. Varias de estas compañías habían trabajado con SGML antes de que se hubiera desarrollado el XML.

Como resultado, el DOM ha recibido influencias SGML y del estándar HyTime. Algunas de estas compañías también habían desarrollado sus propios modelos de objetos para documentos, a fin de proporcionar un API para los editores o los archivos de documentos SGML/XML. Estos modelos de objetos también han influido en el DOM.

Definición del árbol

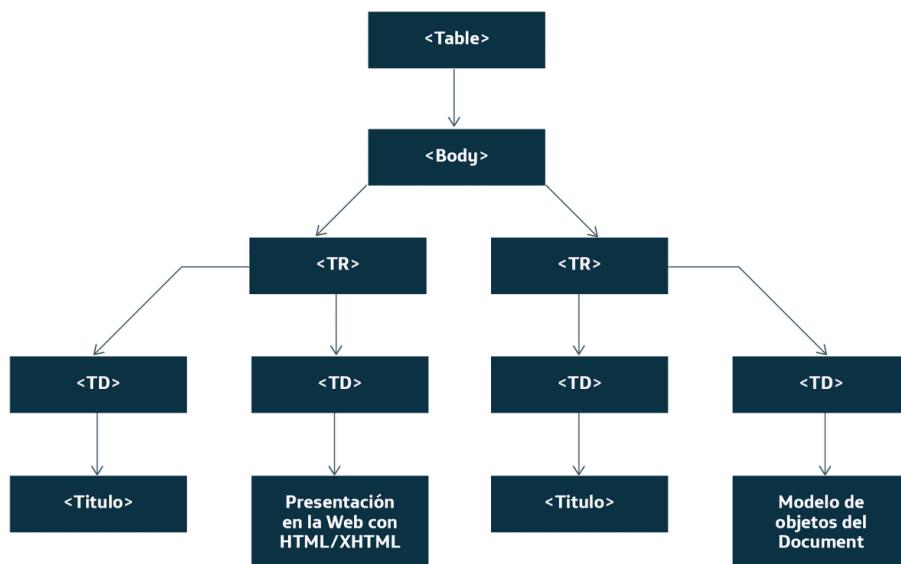
La interfaz del **Modelo de Objetos del Documento**(DOM) tiene una gran similitud con la estructura del documento al que modeliza.

Veamos un ejemplo de cómo se representaría este árbol partiendo de un código HTML.

```
<table>
  <tbody>
    <tr>
      <td>Título: </td>
      <td>Presentación en la Web con HTML/XHTML </td>
    </tr>
    <tr>
      <td>Título: </td>
      <td>El Modelo de Objetos del Documento</td>
    </tr>
  </tbody>
</table>
```

Código HTML para representar una tabla.

El resultado de la creación del DOM representado sería:



Tipos de nodos

Como ya hemos comentado, las aplicaciones de visualización de lenguajes de marcas crean el DOM (Document Object Model) del documento después de su lectura.

En el caso de las páginas web son los exploradores los encargados de esta tarea, creando una representación interna del código HTML en un árbol de nodos.

Los principales tipos de nodos en el DOM son:

Document

El objeto *document* es el nodo raíz, a partir del cual derivan el resto de nodos. Es la frontera entre el navegador y nuestro contenido.

Element

Son los nodos creados por las propias etiquetas de HTML. Un documento web está formado por una jerarquía de nodos. Lo habitual es aplicarles términos de parentesco; si dentro de una etiqueta contenedora *<div>* ponemos dos etiquetas de párrafo *<p>*, estas *<p>* serán las hijas de la etiqueta *<div>*.

Text

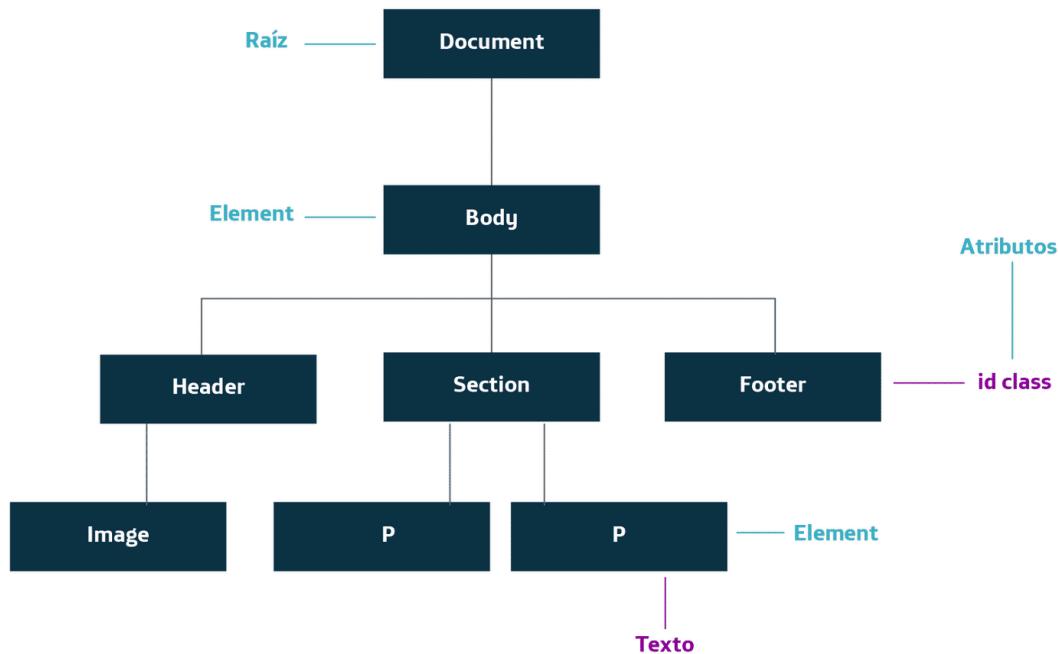
Es el contenido dentro de un nodo *element*. Este contenido, aunque no se representa dentro del árbol generado, se considera un nuevo nodo hijo de *element*.

Comentarios

Existen otros elementos en el documento, como comentarios o el DOCTYPE entre otros, que también generan nodos. Aunque lo habitual en el desarrollo es ignorarlos, dado que rara vez interactuamos con ellos.

Attribute

Attribute: existe otro elemento en el DOM que, aunque no genera un nodo, es de gran importancia. Los atributos son elementos asociados a cada nodo, imprescindibles en muchos casos para su localización o modificación.



Todos los nodos derivan del objeto base Node. En la siguiente tabla se recogen las constantes asociadas a los diferentes tipos de nodos, tal como están definidos por el W3C.

ELEMENT_NODE	El nodo es del tipo Element	1
ATTRIBUTE_NODE	El nodo es del tipo Attr	2
TEXT_NODE	El nodo es del tipo Text	3
CDATA_SECTION_NODE	El nodo es del tipo CDATASection	4
ENTITY_REFERENCE_NODE	El nodo es del tipo EntityReference	5
ENTITY_NODE	El nodo es del tipo Entity	6
PROCESSING_INSTRUCTION_NODE	El nodo es del tipo ProcessingInstruction	7
COMMENT_NODE	El nodo es del tipo Comment	8
DOCUMENT_NODE	El nodo es del tipo Document	9
DOCUMENT_TYPE_NODE	El nodo es del tipo DocumentType	10
DOCUMENT_FRAGMENT_NODE	El nodo es del tipo DocumentFragment	11

Fuente: W3C

Todos los nodos están organizados en forma de árbol jerárquico en un documento HTML, puedes ampliar la información sobre la jerarquía de los tipos del DOM en este enlace.

<http://www.gnu.org/software/classpathx/jaxp/apidoc/org/w3c/dom/tree.html>

Jerarquía de nodos

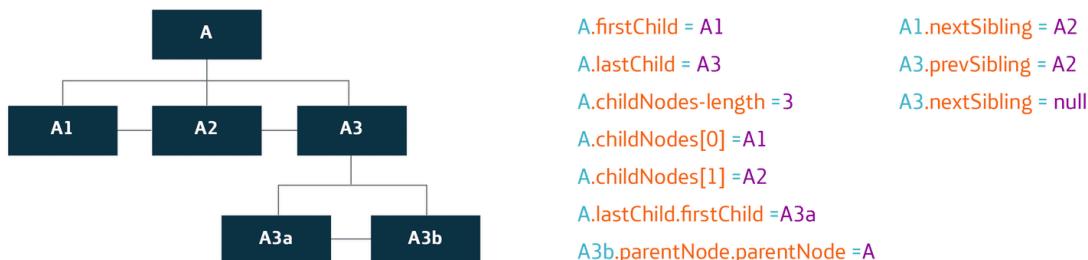
Una vez tenemos el DOM creado, debemos ser capaces de recorrerlo e identificar sus diferentes niveles de parentesco.

Para ello cada nodo (objeto) dispone de atributos y métodos para encontrar a sus padres. Uno de ellos, y el más utilizado, es el **childNodes**, que nos lista los elementos que contiene el nodo padre.

Una vez nos encontramos en un nodo podemos acceder a otros nodos aún cuando no sepamos sus identificadores.

- **childNodes**: es una matriz (*array*) que contiene todos los nodos hijos de un elemento.
- **getAttribute(id)**: devuelve el valor del atributo que identificamos por el parámetro *id*.
- **parentNode**: accede al nodo padre, o lo que es lo mismo, al nodo en el que se encuentra.
- **firstChild y lastChild**: contienen, respectivamente, el primer y último hijo de un nodo.
- **previousSibling y nextSibling**: para acceder a los hermanos del nodo en orden secuencial.
- **hasChildNodes()**: devuelve un valor booleano, indicando si el nodo tiene hijos.

Es importante entender que colocarnos en un nodo no es lo mismo que estar en su contenido. Para ir al contenido usaremos el método `nodeValue`.



Clasificación de objetos en HTML

Como ya hemos comentado, en la creación del árbol de nodos el navegador convierte cada etiqueta en un objeto. En esta conversión le asigna los atributos correspondientes según su tipo.

En HTML existen diferentes tipos de objetos, pero podemos hacer una clasificación de los tipos de objetos y atributos más significativos.

Tipos de atributos

Para cada etiqueta de HTML tenemos atributos y aunque pueden ser específicos (como el `src` para un objeto `img`), tenemos algunos comunes que están en todas las etiquetas:

- Atributos básicos.
- Atributos de foco.
- Atributos de internacionalización.
- Atributos de eventos.

Elementos

Los elementos son las propias etiquetas HTML. Se clasifican en dos grupos: elementos en línea o *inline* o elementos en bloque o *block*.

La diferencia está en el comportamiento que ejerce sobre su contenido; los elementos de bloque siempre empiezan en una nueva línea y ocupan todo el espacio disponible, mientras que los elementos en línea solo ocupan el espacio necesario para mostrar su contenido.

Despedida

Resumen

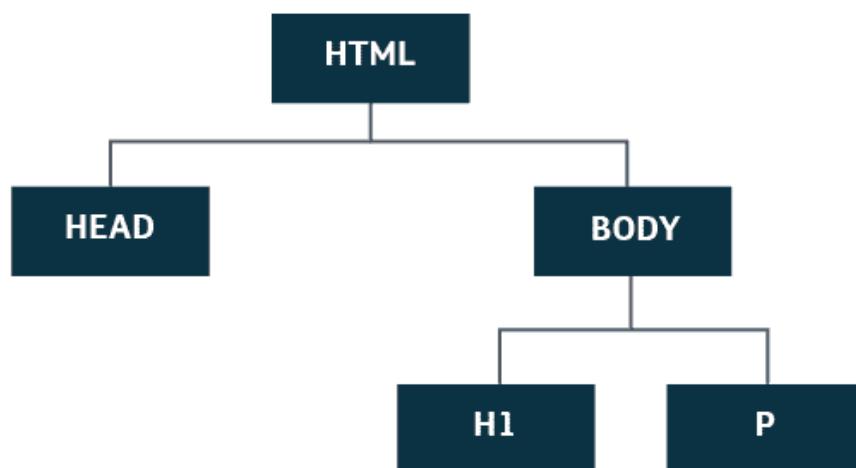
Has terminado la lección, veamos los puntos más importantes que hemos tratado.

El Modelo de Objetos del Documento (DOM) es una interfaz de programación de aplicaciones para documentos válidos y bien construidos en HTML y XML.

Esta interfaz de programación (API) define la estructura lógica de los documentos y el modo en el que se accede y manipula un documento.

La palabra "documento" representa información que puede estar presentada en diferentes sistemas, pero el DOM permite crear documentos, navegar por su estructura, añadir, modificar y eliminar contenido.

Por lo tanto, esta especificación del W3C persigue que el Modelo de Objetos del Documento proporcione una interfaz estándar que pueda ser utilizada en una variedad amplia de entornos y aplicaciones.



MP_0373. Lenguajes de marcas y sistemas de gestión de información

UF2. Lenguajes para la visualización de la información

2.2. Etiquetas HTML



Índice

Objetivos	4
El Documento	5
Introducción	5
Las etiquetas o tags	6
Los atributos	7
Atributos Básicos.....	8
Atributos para internacionalización.	8
Atributos de evento	8
Atributos de foco	9
Los comentarios	9
Los caracteres especiales	9
HTML5.....	13
Validación de marcado	14
La Página.....	15
La cabecera del documento <head>	15
El cuerpo de la página <body>	16
Los textos	17
Encabezados	17
Párrafos	17
Sangrado.....	18
Listas	19
Lista desordenada	19
Lista ordenada.....	19
Listas de definición.....	19
Anidamiento de listas	20
Los enlaces.....	20
Enlaces absolutos y relativos	21
Las tablas	22
Agrupaciones	22
Colspan y rowspan	23
Las imágenes.....	24
Características de las imágenes: tamaño, título, textos alternativos.	25

Introducción a SVG	25
Crear un círculo.....	26
Cuadrados y rectángulos.....	27
Elipse	28
Polígonos	28
Líneas.....	29
Etiquetas semánticas	30
Otras etiquetas	31
Contenedores <div>	31
Formularios	32
Creación de un formulario sencillo	32
Campos de formulario.....	34
<input>	34
Nuevos atributos para campos de formularios HTML5	38
Nuevos tipos de campos HTML5 con validación automática	39
Estructura de un documento HTML.....	40
Despedida	41
Resumen.....	41

Objetivos

En esta unidad perseguimos los siguientes objetivos:

- Identificar las etiquetas y atributos de HTML (Hyper TextMarkup Language).
- Estructurar documentos HTML.
- Conocer las etiquetas de contenido: títulos, párrafos, listas.
- Conocer los elementos de formulario y controles HTML5: campos de texto, botones, desplegables.
- Otros elementos de formato y agrupamiento: tablas, capas.

El Documento

Introducción

HTML (*HyperText Markup Language*) es un lenguaje de etiquetas usado por los diseñadores y programadores para crear páginas web.

Este lenguaje es un estándar reconocido por todo el mundo. Sus normas las define un organismo sin ánimo de lucro llamado **World Wide Web Consortium**. <https://www.w3.org/>

Se ha convertido en el estándar de visualización de la información, siendo la vista de muchas aplicaciones, tanto de las páginas web más tradicionales como de aplicaciones de servidor para servicios empresariales.

Actualmente, los lenguajes de programación más modernos como Java, .NET, Python o PHP disponen de librerías y tienen interiorizado este sistema de representación para mostrar la capa de vista de los programas.

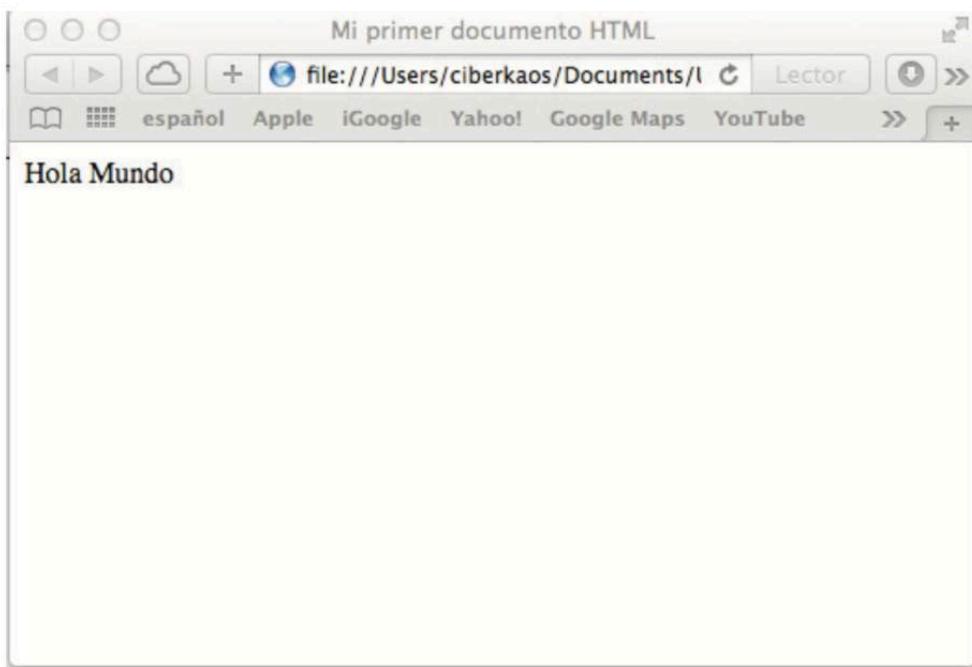
Un documento HTML está formado por su contenido y etiquetas que delimitan sus partes para dar formato.

Empecemos por mostrar un poco de código de una página HTML en su formato más básico:

```
<html>
<head>
<title>Mi primer documento HTML</title>
</head>

<body>
<p>Hola Mundo</p>
</body>
</html>
```

Como podemos ver, tenemos etiquetas (color azul) y contenido (color negro). En este caso estamos imprimiendo en pantalla el texto encerrado en un párrafo `<p>` *Hola Mundo*.



HTML viene de las siglas en inglés de **HyperText Markup Language**, en castellano lenguaje de marcas de hipertexto. Se trata de un lenguaje de marcado para la especificación del contenido de las páginas web, desarrollado y regulado por el consorcio W3C.

Las etiquetas o tags

Una **tag** o **etiqueta** es la marca que usa el lenguaje para delimitar a un elemento específico.

Las etiquetas o marcas están definidas por caracteres de escape que indican al software que lee el documento cuándo comienza y acaba el *tag* o etiqueta. En lenguajes de marcado como HTML existen etiquetas o marcas que usan el carácter "<", el "\\" o el ">".

En HTML y en XML casi todas las marcas indican el comienzo y fin de un elemento.

< p > Este es el texto de párrafo </ p >

Etiqueta de apertura

Etiqueta de cierre

Como podemos apreciar, en este tipo de etiquetas la de cierre es igual que la de apertura pero con la barra (/) delante.

Existen etiquetas que al no tener contenido propio, **no tienen su correspondiente cierre**. Como las etiquetas **
** (salto de línea) o **** (imagen).

Los atributos

Los atributos sirven para dotar al elemento de propiedades que tienen que ver con la marca en la que están contenidos.

La forma de representar dichas propiedades es a través de pares nombre/valor separados por un signo de igual (=), normalmente se escriben después de la etiqueta de comienzo de un elemento.

El valor que va asociado al atributo suele ir encerrado entre comillas dobles o simples.

```
<etiqueta atributo1="valor1" atributo2="valor2">contenido</etiqueta>
```

Existe una clasificación para los atributos:

Atributos básicos

Se pueden utilizar prácticamente en todas las etiquetas HTML.

Atributos para internacionalización

Los utilizan las páginas que muestran sus contenidos en varios idiomas.

Atributos de eventos

Solo se utilizan en las páginas web dinámicas creadas con JavaScript.

Atributos de foco

Relacionados principalmente con la accesibilidad de los sitios web.

Atributos globales en HTML5

Existen en la actualidad atributos globales a todas las etiquetas en HTML5.

https://www.w3schools.com/tags/ref_standardattributes.asp

Con el paso a HTML5 muchos de los atributos han sido eliminados, pues uno de sus objetivos es simplificar el código

Atributos Básicos

Los atributos básicos se utilizan en la mayoría de etiquetas HTML y XHTML, aunque adquieren mayor sentido cuando se utilizan hojas de estilo en cascada (CSS):

id = "texto"

Establece un identificador único a cada elemento dentro de una página HTML.

class = "texto"

Establece la clase CSS que se aplica a los estilos de elemento.

style = "texto"

Establece de forma directa los estilos CSS de un elemento.

title = "texto"

Establece el título a un elemento; mejora la accesibilidad y los navegadores lo muestran cuando el usuario pasa el ratón por encima del elemento.

Atributos para internacionalización

Estos atributos se utilizan en aquellas páginas que muestran sus contenidos en varios idiomas y en las que quieran indicar de forma explícita el idioma de sus contenidos:

lang = "código de idioma"

Indica el idioma del elemento mediante un código predefinido.

xml:lang = "código de idioma"

Indica el idioma del elemento mediante un código predefinido.

dir = "ltr|rtl|auto"

Indica la dirección del texto; útil para los idiomas que escriben de derecha a izquierda.

Atributos de evento

Los atributos de evento son "escuchadores" que lanzan una acción cuando se detecta un comportamiento. Cuando hacemos clic en un botón o movemos el ratón, el navegador puede detectarlo y lanzar una función de JavaScript.

Existe un gran número de eventos; puedes ver una lista de estos con su descripción en la [lista de eventos de w3schools](#).

Atributos de foco

Se denomina **foco** o *focus* cuando un control o elemento del documento ha sido seleccionado. Cuando ese elemento deja de estar seleccionado " pierde el foco" y es el nuevo elemento seleccionado el que se dice que tiene "el foco".

acceskey = "letra"

Establece una tecla de acceso rápido a un elemento HTML.

tabIndex = "número"

Establece la posición del elemento en el orden de tabulación de la página. Su valor debe estar comprendido entre 0 y 32 767.

onfocus, onblur.

Controlan los eventos JavaScript que se ejecutan cuando el elemento obtiene o pierde el foco.

Los comentarios

Los **comentarios** sirven para incluir **información extra que no es interpretada por el navegador** y puede aparecer en cualquier parte de un documento.

Para definirlos en HTML se usa la siguiente etiqueta:

```
<!-- Esto es un comentario -->
```

Los comentarios de HTML no son interpretados por el navegador, por lo que no aparecen en el documento que ve el usuario. Pero sí están presentes en el código, de tal modo que el usuario puede visualizarlos si le pide al navegador que muestre el código fuente.

Los caracteres especiales

La **codificación de caracteres** consiste en la **conversión de un carácter** desde el lenguaje natural a un símbolo que pertenece a otro sistema de representación y se realiza mediante el uso de unas normas y reglas.

En este caso serviría para codificar caracteres a través del lenguaje de marcas y su contenido de lenguaje escrito en los distintos alfabetos existentes.

Una de las primeras codificaciones fue el código **ASCII**, que con 8 caracteres codificó 128 del alfabeto latino.

En la confección de documentos HTML o XML se indica en el encabezado el juego de caracteres que queremos que use el navegador para decodificar la página o documento web.

La codificación de caracteres más moderna se realiza en **Unicode (UTF-8, UTF-16 o UTF-32)** que es un estándar definido por la **ISO10646**.

Por ejemplo, si queremos usar el UTF-8 en una página HTML podríamos incluir en la **cabecera de nuestro documento la siguiente directiva:**

```
<meta charset="UTF-8"/>
```

Puedes ampliar información sobre HTML Document Representation.

<https://www.w3.org/TR/1999/REC-html401-19991224/charset.html>

Cuando se quisieron codificar caracteres especiales o los caracteres de lenguajes de todo el mundo y representarlos en HTML, se crearon unas codificaciones que hacían que el navegador o el intérprete del documento los reconociera como caracteres especiales.

Códigos ASCII

Puedes consultar la tabla de códigos ASCII para caracteres y caracteres especiales.

<https://ascii.cl/es/códigos-html.htm>

De este modo podemos insertar cualquier carácter de la tabla en un documento, solucionando problemas de incompatibilidad de comandos o de internacionalización de caracteres.

```
<!-- Por número -->
<p>&#169; Copy</p>

<!-- Por nombre -->
<p>&copy; Copy</p>
```

Ejemplo de presentación del símbolo de copyright "©" en HTML.

El uso de estos caracteres especiales puede resultarnos útil en muchas ocasiones, pero sobre todo cuando queramos escribir caracteres reservados para el propio lenguaje de marcas, como "<" y ">", por lo que los más habituales son:

- **<** para <.
- **>** para >.
- **&** para &.
- **"** para “.

Estos códigos empiezan siempre con el signo "&" y acaban con ";".

Otro caso en el que nos encontraremos habitualmente es el del idioma. Aunque ya existe el atributo `charset` de HTML, en ocasiones nos encontraremos con problemas de interpretación de caracteres, propios de un idioma, como en nuestro caso la letra ñ. Por eso es importante recordar:

- ñ ñ
- Ñ Ñ
- á á
- é é
- í í
- ó ó
- ú ú
- Á Á
- É É
- ĺ í
- Ó Ó
- Ú Ú
- € € euro

```
<p>Este párrafo contiene caracteres acentuados</p>
```

Ejemplo.

Estructura del código de la página

La **estructura de un documento** permite identificar sus partes de manera que podemos ubicar en cada zona lo referente a esta y construyamos un documento válido.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
    <tittle>No title</tittle>
    <meta name="generator" content="Amaya, see http://www.w3.org/Amaya/" />
  </head>
  <body>
    <h1>Mi primer encabezado</h1>
    <p>Mi primer párrafo</p>
  </body>
</html>
```

La etiqueta `<!DOCTYPE ...>` es de uso obligado, ya que informa al navegador del tipo de documento (si es HTML o XHTML) y cuál es la versión. Recuerda que existen varios tipos de documentos y diferentes versiones.

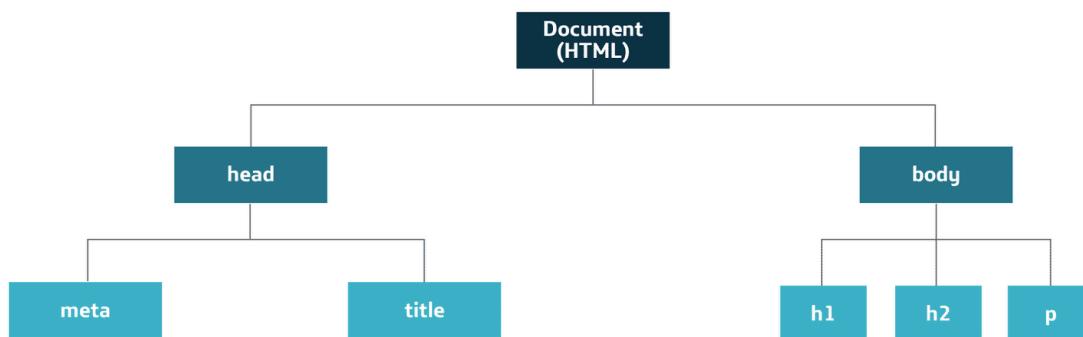
La etiqueta `<html> ... </html>` define el comienzo y el final del documento.

La cabecera `<head> ... </head>` es un elemento que contiene los encabezados de la página.

En la cabecera se pueden incluir:

- Etiquetas *meta*.
- Título.
- Recursos enlazados.

La etiqueta **<body>...</body>** define el cuerpo del documento HTML, en esta se incluyen todas las etiquetas que veremos a continuación en el curso, y por supuesto los atributos y el texto o contenido de cada elemento.



El DOCTYPE

La etiqueta <!DOCTYPE ...> es de uso obligado, ya que informa al navegador del tipo de documento (si es HTML o XHTML) y cuál es la versión. Recuerda que existen varios tipos de documentos y diferentes versiones.

Esto es necesario para traducir las etiquetas, ya que no son iguales en todas las versiones. En la etiqueta **<!DOCTYPE ...>** se incluye la URL de la DTD en la que se especifica la versión del lenguaje de etiquetas utilizado en el documento, además de la estructura, etiquetas y atributos permitidos para cada una de ellas.

En HTML 4.01 se indicará si el **doctype es strict, transitional o frameset**. Para HTML5 se usa **<!DOCTYPE html>**

Tipos de doctype

STRICT

Es la versión más estricta y severa. Los sitios que incluyan este tipo de *doctype*, no pueden incluir atributos que estén relacionados con las características y el aspecto de los contenidos. Esto supone una separación del documento HTML de los estilos CSS. Es la recomendada por la W3C.

TRANSITIONAL

Esta versión, a diferencia de la anterior, sí permite utilizar alguno de los atributos de aspecto para el contenido.

FRAMESET

Esta versión la utilizan las páginas o sitios que están formados por *frames* (marcos), algo que ya está muy en desuso y prácticamente obsoleto.

Actualmente, con HTML5 se ha simplificado el *Doctype* indicando solo el parámetro "html".

```
<!DOCTYPE html>
```

Ejemplos

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

HTML 4.01 Transitional: esta DTD contiene todos los elementos HTML y atributos, incluyendo elementos de presentación y obsoletas (como **). *<frameset>* no está permitida.

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

HTML 4.01 Frameset: igual que el anterior, pero se permiten los *<frameset>*.

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

XHTML 1.0 Strict: esta DTD contiene todos los elementos HTML y atributos, pero no incluye los elementos de presentación o en desuso (como **). *<frameset>* no está permitido. El marcado también debe ser escrito como XML bien formado.

```
<!DOCTYPE html>
```

XHTML5

HTML5

HTML5 es el estándar más actual para realizar nuestras páginas. Se basa en la unión de HTML, CSS y JavaScript.

HTML5 es la última actualización de HTML. HTML5 también es un término que **sirve para agrupar las tecnologías web HTML5, CSS3 y nuevas capacidades de JavaScript.**

La necesidad de esta nueva versión viene por que su antecesora HTML 4 carece de las características necesarias para la creación de aplicaciones modernas basadas en un navegador y los requerimientos de indexación (SEO) que demandan los buscadores de Internet como Google.

Otra de las razones es el declive de uso de Adobe Flash para dotar de dinamismo nuestras páginas. Así que es necesario que HTML disponga de herramientas para que nuestros navegadores puedan gestionar audio, vídeo, animaciones vectoriales, componentes de interfaz, entre muchas otras cosas. Ahora HTML5 es capaz de hacer esto sin necesidad de *plugins* y con una gran compatibilidad entre navegadores.



Cuando trabajes con documentación, fíjate bien que esté en HTML5. Muchas de las estructuras y atributos usados en HTML 4 ya no son validadas y nos perjudicaran en el posicionamiento natural de nuestras páginas (SEO).

Validación de marcado

Para saber si nuestro código cumple los estándares de HTML5 podemos utilizar las herramientas de validación disponibles en Internet.

<http://validator.w3.org/>

A screenshot of the W3C Markup Validation Service website. The top navigation bar includes links for 'Validate by URI', 'Validate by File Upload', and 'Validate by Direct Input'. Below this is a 'Validate by URI' section with a 'Address:' input field and a 'Check' button. A note at the bottom explains the service's purpose and provides links for RSS/Atom feeds, CSS stylesheets, and mobile content validation. The footer contains links for 'Home', 'About...', 'News', 'Docs', 'Help & FAQ', 'Feedback', and 'Contribute', along with the W3C logo and a copyright notice.

Una de las características más relevantes de HTML5 no es la inclusión de nuevas etiquetas, sino la simplificación de etiquetas y el uso de atributos existentes en versiones anteriores.

La Página

La cabecera del documento <head>

La cabecera **<head> ... </head>** es un elemento que contiene los encabezados de la página. En HTML5 puede ser omitida.

En la cabecera se pueden incluir:

- Etiquetas *meta*.
- Título.
- Recursos enlazados.

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso- 8859-1"/>
    <tittle>No tittle</tittle>
    <meta name="generator" content="Amaya, see
http://www.w3.org/Amaya/" />
</head>
```

Etiquetas que se pueden incluir en la etiqueta <head>

Charset

Valor: conjunto de caracteres UTF-8 Unicode.

Descripción: especifica la codificación de caracteres del HTML.

Content

Valor: texto.

Descripción: valor dado asociado a la etiqueta *http-equiv* o el valor de un atributo.

http-equiv

Valor: *application-name, author description, generator, keywords*.

Descripción: metadatos que definen los conceptos asociados.

Link

Valor: recursos enlazados, p. ej.: `<link rel="stylesheet" type="text/css" href="theme.css">`.

Descripción: sirve para incluir estilos enlazándolos con el fichero que se indica.

Script

Valor: p. ej.: <script>

src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>.

Descripción: en la cabecera se puede enlazar con librerías de JavaScript.

Base

Valor: indica la raíz de la página, p. ej.: <base href='http://www.midominio.org/'>.

Descripción: establece la URL base del documento.

Style

Valor: p. ej.: <style> h1{color:red;} p{color:blue} </style>.

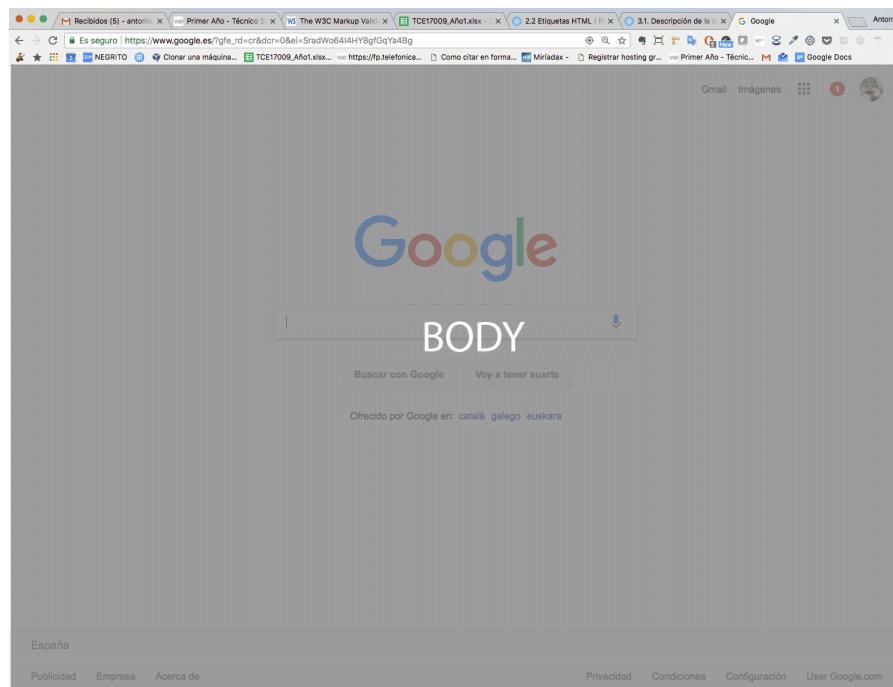
Descripción: define estilos de la página web en la cabecera.

El cuerpo de la página <body>

La etiqueta <body>...</body> define el cuerpo del documento HTML.

En esta se incluyen todas las etiquetas que veremos a continuación en el curso, y por supuesto los atributos y el texto o contenido de cada elemento.

```
<body>
  <h1>Mi primer encabezado</h1>
  <p>Mi primer párrafo</p>
</body>
```



Entre las etiquetas <body> se encuentra el contenido visible de la página.

Los textos

Encabezados

La estructura de un documento web, en principio, debería ser parecida a un documento de texto preparado para la lectura y organizado según temas y subtemas. **Las etiquetas de los encabezados o "headings"** sirven para ello.

Hay hasta **seis encabezados**, que difieren en el tamaño del texto que incluyen. Los encabezados se delimitan con el tag `<h1>` hasta el `<h6>`, `<h1>` es el título mayor y así van disminuyendo hasta el sexto, que sería el título menor.

[Puedes verlo en este ejemplo](#)

La mayoría de los navegadores visuales muestran encabezados como texto grande y en negrita de forma predeterminada, aunque esto puede ser configurado si modificamos los estilos en la CSS asociada.

Los elementos de título no pueden ser usados para agrandar o poner en negrita texto normal, ya que su uso está asociado al título de una sección. Los encabezamientos o títulos existen en etiquetas HTML desde su estandarización en la versión HTML 2.0, y todavía se usan. Todos los navegadores soportan esta etiqueta.

Párrafos

Los documentos en HTML, como cualquier documento de texto, están divididos en párrafos. Un párrafo está definido por la etiqueta o elemento `<p>...</p>`.

[Puedes verlo en este ejemplo](#)

Tenemos otras etiquetas que sirven para introducir un salto de línea, como el elemento `
`. También existe la etiqueta `<pre>...</pre>` y sirve para que el texto se visualice tal como está en el código fuente, con lo que respetaría los saltos de línea o espacios en blanco extra y el texto se representa con un tipo de fuente de ancho fijo.

En HTML5 todas las modificaciones de apariencia y estilo se realizarán con CSS, por lo que se han eliminado los atributos para ese fin.

Tanto en párrafos como en otros contenedores de texto podemos querer poner partes en negrita, cursiva, subrayado, etc. Para ello disponemos también de multitud de etiquetas de estilo lógico o físico.

Etiquetas de estilo físico

Especifican un cambio concreto en el tipo de letra, estas etiquetas están en desuso frente a las de estilo lógico o semántico, dado que favorecen el posicionamiento SEO de la página.

- texto en negrita: ...
- texto en cursiva: <I> ... </I>
- texto subrayado: <U> ... </U>
- texto monoespaciado: <TT> ... </TT>
- texto tachado: <STRIKE> ... </STRIKE>
- aumentar el tamaño del texto: <BIG> ... </BIG>
- disminuir el tamaño del texto: <SMALL> ... </SMALL>
- texto en subíndice: _{...}
- texto en superíndice: ^{...}

Etiquetas de estilo lógico

No solo especifican una zona de texto a resaltar, sino que le dan contenido semántico:

- texto destacado: ...
- texto importante: ...
- citas y referencias: <CITE> ... </CITE>
- información sobre el autor: <ADDRESS> ... </ADDRESS>
- definición de un término: <DFN> ... </DFN>
- fragmento de código de programa: <CODE> ... </CODE>
- Ejemplos de programa: <SAMP> ... </SAMP>
- Variables, argumentos de un programa: <VAR> ... </VAR>

[Puedes verlo en este ejemplo](#)

Sangrado

<blockquote>... </blockquote> sirve para realizar un sangrado en el texto.

Esta etiqueta se usaba inicialmente para el marcado de citas, como los navegadores hacen un sangrado para la cita así, se convirtió en la etiqueta para el sangrado. Esta etiqueta tiene el atributo *cite*, que es del tipo URL.

Para aplicar un sangrado desde CSS usaremos el atributo *text-indent*, al que le indicaremos un valor numérico.

```
<blockquote cite="http://www.google.com"> ... </blockquote>
```

Listas

Las listas de conceptos nos ayudan a definir conceptos y valores.

- Elemento 1
- Elemento 2
 - 1. Subelemento1
 - 2. Subelemento1
 - 3. Subelemento1
 - 4. Subelemento1
- Elemento 3
- Elemento 4

Ejemplo lista anidada.

Lista desordenada

**** define una lista sin orden. Cada elemento de la lista comienza con la etiqueta ****.

```
<ul>
    <li>Café</li>
    <li>Leche</li>
</ul>
```

Lista ordenada

**** define una lista ordenada, cada elemento de la lista comienza con la etiqueta ****.

```
<ol>
    <li>Café</li>
    <li>Leche</li>
</ol>
```

Listas de definición

<dl> es una etiqueta que define una lista de descripciones, se utiliza en conjunción con:

- **<dt>** términos a definir.
- **<dd>** describe cada término.

Las listas tienen unos atributos específicos que definen la marca delante del elemento de la lista.

```
<dl>
  <dt>Café</dt>
    <dd>- bebida negra caliente</dd>
  <dt>Leche</dt>
    <dd>- bebida blanca fría</dd>
</dl>
```

Anidamiento de listas

Para anidar listas se usan las mismas estructuras que hemos estudiado, teniendo en cuenta que dentro de cada elemento definido por **** incluiremos elementos ****. Veamos un posible anidamiento.

```
<ol>
  <li><h3>Primer plato</h3>
    <ul>
      <li>Macarrones</li>
      <li>Gazpacho</li>
      <li>Judías pintas</li>
    </ul>
  </li>
  <li><h3>Segundo plato</h3>
    <ul>
      <li>Pollo al ajillo</li>
      <li>Merluza frita</li>
    </ul>
  </li>
</ol>
```

[Puedes verlo en este ejemplo](#)

Los enlaces

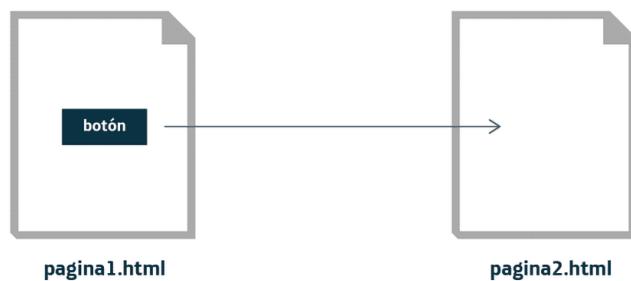
URL "**Uniform Resource Locator**" se traduce como localizador de recursos uniforme. Este recurso se diseña y se usa por primera vez por Tim Berners-Lee para enlazar documentos a través de hiperenlaces en la red *World Wide Web*.

URL está formado por un formato específico que está estandarizado desde 1994 y forma parte del estándar **URI "Uniform Resource Identifier"**, en español identificador uniforme de recurso. El recurso tiene una dirección única disponible en Internet. El formato general de una URL es: **esquema://máquina/directorio/archivo**

<a> es la etiqueta usada para enlazar con recursos a través de una URL.

El enlace se puede asociar a un texto, una imagen o un elemento HTML. Una vez enlazado, pulsando en el elemento se navegará hacia el recurso definido. **El atributo más importante del elemento <a> es el atributo href**, que indica el destino del enlace, es decir, la URL del recurso al que queremos acceder.

Otros atributos como *download*, *hreflang*, *media*, *rel*, *target*, *type*... no tiene sentido usarlos mientras que no tengamos el atributo *href*.



```
<A href="pagina2.html"> botón </A>
```

Atributo target

El atributo *target* está soportado por HTML5 y los navegadores más usados. Antiguamente se usaba para indicar en qué división (*frame*) de la página queríamos que actuase el enlace.

Actualmente en HTML5 solo se usa la opción *_blank* para abrirlo en una ventana nueva.

```
<a href="dirección" target="_blank">Botón</a>
```

Enlaces absolutos y relativos

Normalmente las páginas que se desarrollan en un proyecto estarán alojadas en un servidor web, que además tendrá referencias del dominio que apunta a dichas páginas web. Esto implica que la URL base del dominio apunta al directorio raíz del sitio, por ejemplo <http://www.misitio.com>. El resto de recursos estarán, en la mayoría de los casos, alojados en directorios del sistema de ficheros del servidor que parten de la raíz del sitio o en directorios virtuales.

Dirección absoluta

Para hacer referencia a un sitio externo se usarán direcciones absolutas que indicarán la dirección raíz del sitio, por ejemplo www.misitio.com. Si es necesario acceder a un recurso como "fichero.html", alojado en una carpeta o directorio hijo, por ejemplo "/documentos/", se añadirá a la dirección global de esta forma:

Dirección relativa

Para hacer referencia a un recurso interno que forma parte del sitio web que está por encima, por debajo o al mismo nivel en el árbol de directorio del sitio, usaremos direcciones relativas al directorio donde está la página que contiene el enlace al recurso.

Si el sitio web es *www.misitio.com* y quiero hacer referencia a un recurso que está en el directorio "/documentos/" y tengo que acceder a un recurso que cuelga de ese directorio, por ejemplo en "/documentos/pdf/fichero.html", podré crear la dirección de esta forma:

Las tablas

<table> define una tabla en HTML. Cada **<tr> define una fila** y cada **<td> define una celda**, cuyo número deberá coincidir con el número de columnas definidos.

Listado de personas por edad

Nombre	Apellido	Edad
Juan	Escudero	50
Josefa	Valcarcel	94

Cada celda puede contener otros elementos como texto, tablas, listas imágenes, etc. El elemento **<th> define un encabezado** de la tabla.

<caption> sirve para definir el título de la tabla, saldrá al principio de la tabla. HTML5 solo admite el atributo "*border*", y su valor puede ser "1" o "".

[Puedes verlo en este ejemplo](#)

Igualmente que el resto de elementos, **en HTML5** el aspecto visual (bordes, márgenes, líneas...), **se aplicarán únicamente con CSS**.

Agrupaciones

Para formatear una tabla se utilizan los estilos de CSS, pero también existen marcas para agrupar y aplicar estilos como estas:

- **<colgroup>**. Especifica un grupo de una o más columnas en una tabla para formatear.

- **<col>**. Especifica las propiedades de columna para cada columna dentro de un elemento `<colgroup>`.
- **<thead>**. Agrupa elementos de la cabecera de la tabla.
- **<tbody>**. Agrupa elementos del cuerpo de la tabla.
- **<tfoot>**. Agrupa elementos del pie de la tabla.

[Puedes verlo en este ejemplo](#)

Las agrupaciones con los elementos `thead`, `tbody` y `tfoot` no afectan al diseño de la tabla por defecto, pero se podrían usar estilos para cambiarlos.

El anidamiento de tablas hace que el navegador tenga que procesar más instrucciones, por lo que hay que intentar evitarlo.

Colspan y rowspan

Países Europeos	Madrid
	Francia
	París

Países Europeos		
Madrid	Francia	París

En ocasiones queremos formar tablas donde una celda abarque más que otras, para ello utilizaremos `colspan` y `rowspan`.

- **rowspan**: indica el número de celdas que ocupará horizontalmente en la fila.
- **colspan**: indica el número de columnas que ocupará la celda.

De esta forma, si ponemos `<td colspan="2">` quiere decir que la celda actual se extiende en el ancho de dos celdas.

Algo parecido ocurre si ponemos `<td rowspan="3">`, la celda ocupará el alto de 3 celdas normales.

[Puedes verlo en este ejemplo](#)

Las imágenes

La etiqueta nos permite integrar imágenes en nuestro documento HTML.

Una imagen en formato digital contiene información sobre cada parte de la imagen, es decir, cada píxel. A través de una codificación se especifica información como el color y las capas.

Los formatos más utilizados son:

JPEG

Joint Photographic Experts Group; comprime la información de la imagen mediante un algoritmo. Eso supone pérdida de calidad.

PNG

Portable Network Graphics; comprime la información sin ninguna pérdida y no tiene patente. Almacena por lo tanto más datos que el formato anterior.

GIF

Graphics Interchange Format; este formato admite hasta 8 bits por píxel en cada imagen, y permite una sola imagen para hacer referencia a su propia paleta de hasta 256 colores diferentes, elegidos de los 24 bits de espacio de color RGB. También es compatible con animaciones y permite una paleta de hasta 256 colores para cada fotograma. Estas limitaciones en la paleta hacen que el formato GIF sea menos adecuado para la reproducción de fotografías en color y otras imágenes con color continuo, pero es muy adecuado para las imágenes más simples, tales como gráficos o logos con áreas sólidas de color.

SVG

SVG (*Scalable Vector Grafics*) o Vectores Gráficos Escalables es la herramienta que nos proporciona HTML5 para crear gráficos o animaciones mediante vectores.

```

```

Características de las imágenes: tamaño, título, textos alternativos.

 es la etiqueta para incluir imágenes en nuestra página web. no tiene cierre y requiere llevar dos atributos, que son *src* y *alt*. tiene atributos globales y de eventos y además unos específicos.

```

```

Ruta y nombre del archivo de imagen

Texto alternativo de la imagen, es decir, si no se pudiera cargar la imagen, sería el texto que sustituiría dicha imagen.

Atributos

Los atributos de la etiqueta más habituales son:

- **src** - Ruta relativa o absoluta del archivo de la imagen.
- **alt** - Texto alternativo de la imagen.
- **title** - Título de la imagen.

Introducción a SVG

SVG (*Scalable Vector Grafics*) o Vectores Gráficos Escalables es la herramienta que nos proporciona HTML5 para crear gráficos o animaciones mediante vectores.

Con el uso de Flash desaprobado para realizar elementos como dibujos, banners, animaciones, etc. Las alternativas actuales son *Canvas* y *SVG*, con ambos podemos hacer las mismas cosas pero con diferente forma de trabajar.

Mientras que con *canvas* se basa en mapas de bits, con *SVG* los dibujos se crean mediante vectores. Esto significa que la creación de los elementos se realiza mediante parámetros y coordenadas en lugar de pixeles, lo que hace que la **calidad sea óptima** dado que se **adaptará a cualquier resolución**.

SVG se basa en el lenguaje XML. Mediante etiquetas construiremos las figuras teniendo la posibilidad de crear animaciones en conjunción con Javascript.

SVG se basa en el lenguaje XML. Mediante etiquetas construiremos las figuras teniendo la posibilidad de crear animaciones en conjunción con Javascript.

Veamos cómo se construyen algunas figuras básicas en SVG

Para integrar una imagen en SVG es necesario integrar las etiquetas **<svg> ... </svg>** e indicar las dimensiones totales del gráfico (*width* y *height*) como atributos. Dentro de esta etiqueta definiremos el tipo de figura a integrar y sus propiedades.

<svg width=200 height=200> ... gráfico ... </svg>

Puedes consultar la web de w3schools para ampliar la información sobre SVG.

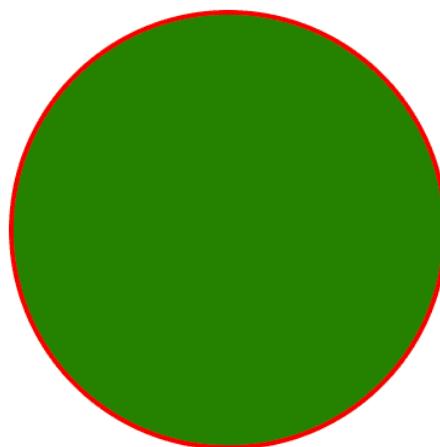
https://www.w3schools.com/html/html5_svg.asp

Puedes ampliar la información sobre los atributos de los SVG en este enlace.

<https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute>

Crear un círculo

Para construir el elemento, tendremos que darle una serie de propiedades como sus dimensiones, borde, relleno, etc.



```
<svg width="200" height="200">
<circle cx="100" cy="100" r="90" stroke="red" stroke-width="2" fill="green">
</svg>
```

Propiedades:

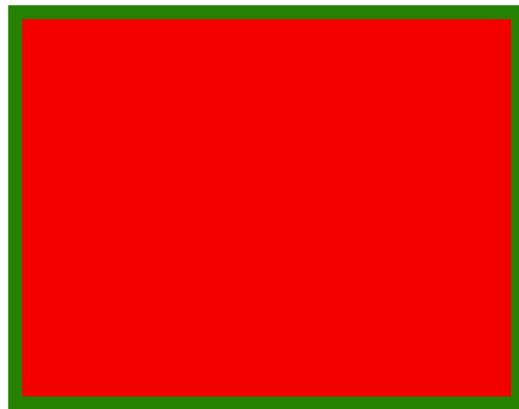
- **circle**: etiqueta correspondiente a la figura.
- **cx=n cy=n**: las coordenadas para la construcción del círculo. Indica en pixeles las distancias desde el centro tanto del eje x como del eje y.
- **r=n**: Longitud del radio de la circunferencia.

En el ejemplo se aplican otros atributos que podemos usar en todas las figuras:

- **stroke="color"**: Color del borde.
- **stroke-width=n**: Grosor de la línea exterior.
- **fill="color"**: Color del relleno de la figura.

Cuadrados y rectángulos

Como en el ejemplo anterior usaremos la etiqueta correspondiente a la figura, dando valores a los atributos que queramos aplicar.



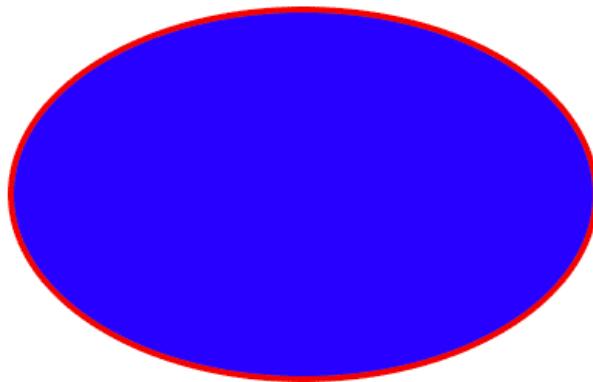
```
<svg width="200" height="200">
<rect x="10" y="30" width="180" height="140" stroke="green" stroke-width="5"
fill="red"/>
</svg>
```

Propiedades:

- **rect**: con esta etiqueta indicamos que vamos a dibujar un rectángulo.
- **x=n, y=n**: Esquina superior izquierda del rectángulo.
- **width=n, height=n**: Anchura y altura del rectángulo.

Elipse

Al igual que en las figuras anteriores utilizamos los atributos ***stroke***, ***stroke-width*** y ***fill***, para indicar el color del borde, el grosor del borde, y el color de relleno.



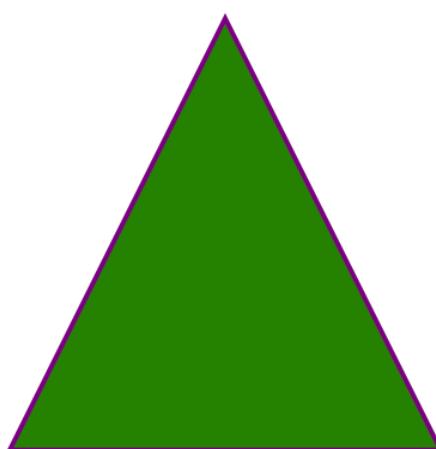
```
<svg height="200" width="200">
<ellipse cx="100" cy="100" rx="95" ry="60"
  stroke="red" stroke-width="2" fill="blue" />
</svg>
```

Propiedades:

- ***ellipse***: etiqueta para elipse.
- ***cx=n, cy=n***: coordenadas del centro de la elipse.
- ***rx=n***: radio horizontal.
- ***ry=n***: radio vertical.

Polígonos

En el caso de los polígonos iremos indicando los puntos con el atributo ***point*** en una serie de parejas de números.



```
<svg height="200" width="200">
  <polygon points="100,10 190,190 10,190"
    stroke="purple" stroke-width="2" fill="green"/>
</svg>
```

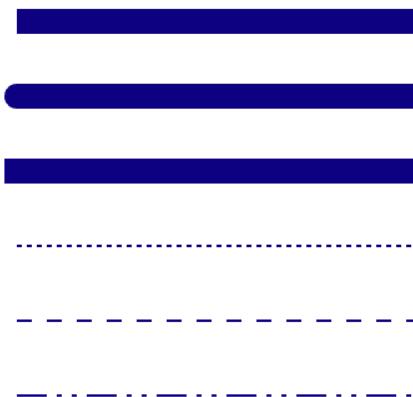
Propiedades:

- ***polygon***: etiqueta para crear el polígono.
- ***points***: cada pareja de números sirve para definir la coordenada *x* e *y* de cada vértice del polígono.

Líneas

Para dibujar líneas usaremos la etiqueta *line*, e indicando mediante coordenadas *x* e *y* tanto el inicio como el final de la linea.

```
<svg height="200" width="200">
  <line x1="5" y1="5" x2="190" y2="190" stroke="red" stroke-width="2"/>
</svg>
```



```
<svg width="200" height="200">
<line x1="20" y1="20" x2="180" y2="20" stroke="navy" stroke-width="10" stroke-linecap="butt"/>
<line x1="20" y1="50" x2="180" y2="50" stroke="navy" stroke-width="10" stroke-linecap="round"/>
<line x1="20" y1="80" x2="180" y2="80" stroke="navy" stroke-width="10" stroke-linecap="square"/>
<line x1="20" y1="110" x2="180" y2="110" stroke="navy" stroke-width="2" stroke-dasharray="2,2"/>
<line x1="20" y1="140" x2="180" y2="140" stroke="navy" stroke-width="2" stroke-dasharray="6,6"/>
<line x1="20" y1="170" x2="180" y2="170" stroke="navy" stroke-width="2" stroke-dasharray="12,4 2,4 2,4"/>
</svg>
```

- ***x1, y1***: coordenadas horizontal y vertical del principio de la línea.
- ***x2, y2***: coordenadas horizontal y vertical del final de la línea.
- ***stroke-linecap="butt"***: forma de inicio de la línea. Admite los siguientes valores:
 - *butt*

- *round*
- *square*
- *inherit*
- ***stroke-dasharray***: líneas discontinuas.

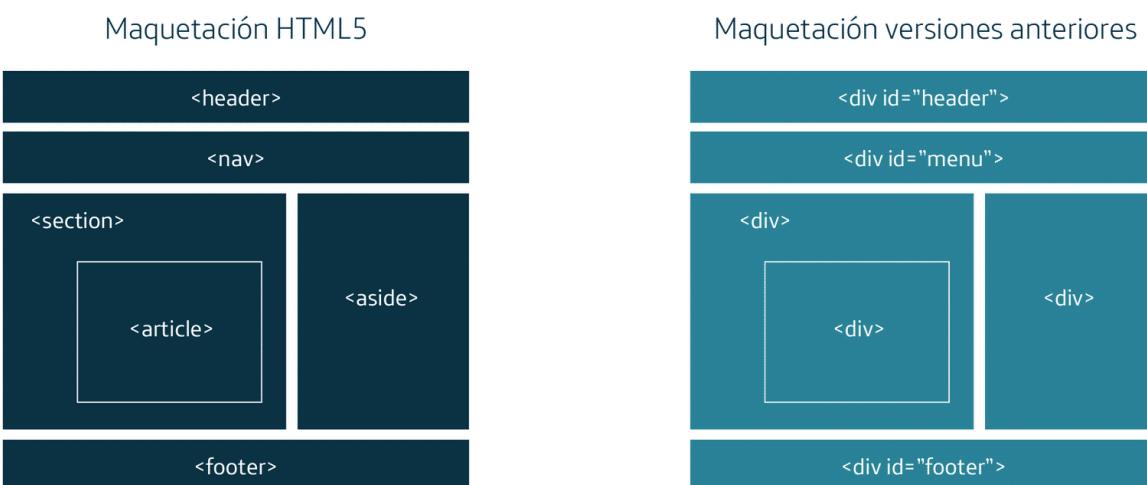
[A continuación tienes un ejemplo de integración de varios SVG.](#)

Etiquetas semánticas

HTML5 incorpora nuevas etiquetas denominadas etiquetas semánticas. Estas etiquetas no tienen contrapartida visual, actúan como capas igual que la etiqueta `<div>`. Lo que las diferencia de las etiquetas `<div>` es que sirven para informar sobre el tipo de contenido que soporta.

Nuevas etiquetas semánticas:

- **`<header>`**. Cabecera de la página. Normalmente contendrá el logotipo y título.
- **`<nav>`**. Área de navegación. Normalmente contendrá enlaces, menús o botones con vínculos.
- **`<section>`**. Una sección dentro de la página que podrá estar dividida en artículos.
- **`<article>`**. Un artículo.
- **`<aside>`**. Área de *banners* o publicidad.
- **`<footer>`**. Pie de página.



Maquetación de una web con etiquetas semánticas HTML5 y maquetación a base de etiquetas `<div>` con las versiones anteriores.

El uso de las etiquetas semánticas nos ayuda a posicionar mejor nuestra web en los buscadores, ya que los motores de búsqueda sabrán el tipo de contenido que encierran y darán más peso a

unas áreas que a otras. Por ejemplo, para los motores de búsqueda tiene más peso el texto incluido en el encabezado del área de navegación que el texto del resto de la página.

Un ejemplo de documento HTML maquetado con HTML5:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Página con formato HTML5</title>
<link href="estilos/estilo.css" rel="stylesheet">
</head>
<body>
    <header> header </header>
    <nav>
        <a href="#">Opción 1</a>
        <a href="#">Opción 2</a>
        <a href="#">Opción 3</a>
    </nav>
    <div id="contenido">
        <section>
            <article>article 1</article>
            <article>article 2</article>
            <article>article 3</article>
            <article>article 4</article>
        </section>
        <aside>aside</aside>
    </div>
    <footer> footer </footer>
</body>
</html>
```

Página web con etiquetas semánticas HTML5.

Otras etiquetas

Contenedores <div>

La etiqueta **<div>** establece un bloque mediante una división o una sección de un documento web en HTML. Además, se usa para establecer y/o formatear a través de un CSS un bloque de texto.

```
<!DOCTYPE html>
<html>
    <body>
        <div style="color:#0000FF">
            <h3>Un título de texto en un bloque div que es azul</h3>
            <p>Texto normal en un elemento div que es azul.</p>
        </div>
        <p>Texto fuera del div.</p>
    </body>
</html>
```

Etiqueta

La etiqueta <**span**> se usa para realizar agrupaciones en línea de los elementos de un documento. Esta etiqueta no cambia la visualización de un texto ella sola, necesitará de estilos de CSS para realizarlo. Además, facilita añadir una clase a una parte específica de un texto o de una parte de un documento.

```
<!DOCTYPE html>
<html>
  <body>
    <p>Mi hermano tiene los ojos <span style="color:blue; font-weight:bold;">azules</span>. </p>
  </body>
</html>
```

Formularios

Creación de un formulario sencillo

Los formularios permiten recoger información del usuario para enviarla al servidor web. También pueden ser utilizados para mostrar información al usuario.

Vamos a comenzar por crear un formulario simple y a partir de su código iremos explicando la teoría.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Formulario</title>
<link href="estilos/estilo.css" rel="stylesheet">
</head>
<body>
  <form action="procesa.jsp" method="get">
    <fieldset><legend>Datos personales:</legend>
      <label for="nombre">Nombre: </label>
      <input type="text" name="nombre" />
      <br />
      <label for="edad">Edad: </label>
      <input type="text" name="edad" />
      <br />
      <input type="submit" value="Enviar datos" />
      <input type="reset" value="Limpiar formulario" />
    </fieldset>
  </form>
</body>
</html>
```

Los datos del formulario se entregan al recurso de servidor procesa.jsp cuando el usuario pulsa el botón submit (enviar).

Vamos a estudiar las nuevas etiquetas:

FORM

<form>

Un formulario está situado dentro de la etiqueta `<form>` y encierra elementos de formulario que permiten al usuario interactuar. Estos elementos pueden ser: campos de texto, etiquetas, botones, listas desplegables, casillas de verificación, etc.

El **atributo action** determina el proceso al que se enviarán los datos introducidos por el usuario cuando se pulse el botón *submit*.

El **atributo method** determina la forma en que se enviarán los datos. Con el valor *get* los datos se envían en la cabecera de la petición HTTP, por lo que serán visibles en la URL. Este método no es recomendable si los datos son confidenciales. Con el valor *post* los datos se envían en el cuerpo de la petición HTTP, por lo que no serán visibles en la URL.

FIELDSET

<fieldset>

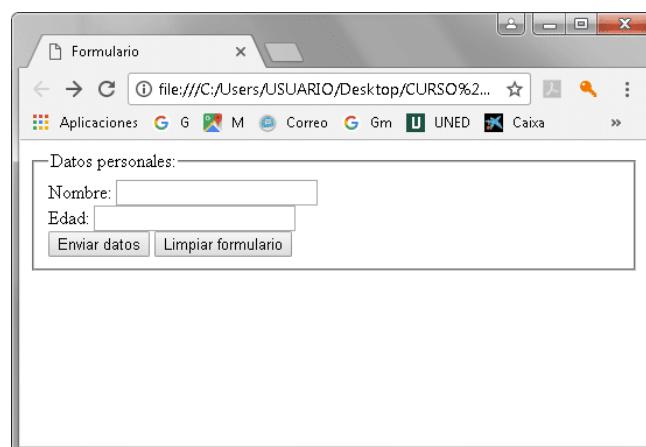
Genera un marco visible que encierra los datos del formulario.

LEGEND

<legend>

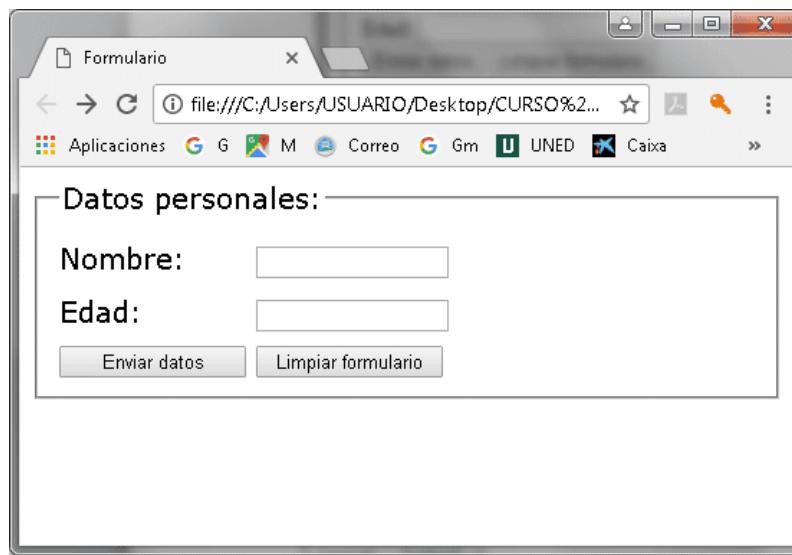
Colocado inmediatamente después de una etiqueta `<fieldset>`, permite colocarle una etiqueta de texto (título).

El resultado en el navegador, sin haber creado todavía el archivo CSS es el siguiente:



Con un poco de CSS podemos lograr que el estilo cambie bastante.

```
@CHARSET "ISO-8859-1";  
  
body {  
    font-family: Verdana;  
    font-size: 20px;  
}  
input, label {  
    margin-top: 10px;  
    display: inline-block;  
    width: 130px;  
}
```



Campos de formulario

Un formulario de entrada de datos está compuesto por un conjunto de campos que pueden ser de distinto tipo: cajas de texto, casillas de verificación, listas de opciones, etc.

Vamos a descubrir poco a poco las etiquetas HTML que representan campos de formulario.

<input>

La etiqueta `<input>` define un elemento de campo de formulario. Estos son sus atributos más comunes:

- **type:** tipo de campo. El valor más usual es `text` (caja de texto).
- **value:** valor por defecto.
- **readonly:** solo lectura. No puede ser modificado por el usuario.
- **name:** el valor de este atributo será el que usará el recurso de servidor para recuperar el valor del campo.

```
<input type="text" name="edad" />
```

<label>

Define una etiqueta estática, que normalmente estará asociada con un elemento de campo de formulario que se identifica mediante el atributo *for*.

También se utiliza para presentar resultados de operaciones.

<input type="submit">

Se representa como un botón y cuando el usuario hace clic en él, se realiza la petición HTTP al programa de servidor especificado en el atributo *action*.

<input type="reset">

Se representa como un botón y cuando el usuario hace clic en él, se borra el contenido de todos los campos del formulario.

<input type="hidden">

Campo oculto. Puede ser útil para enviar una información al servidor que no queremos que sea vista por el usuario.

<input type="password">

Funciona como una caja de texto, pero en lugar de los caracteres que escribe el usuario se ven solo asteriscos o círculos.

<input type="radio">

Se representa como un botón de *radio*. Todos los botones de *radio* que tienen el mismo *name* pertenecen al mismo grupo y solamente uno de los elementos del grupo podrá ser seleccionado por el usuario.

```
<label for="ec">Estado civil: </label>
<br />
<input type="radio" name="ec" value="S" />Soltero<br />
<input type="radio" name="ec" value="C" />Casado<br />
<input type="radio" name="ec" value="V" />Viudo<br />
<input type="radio" name="ec" value="D" />Divorciado<br />
```

Este es el resultado:

Estado civil:

- Soltero
- Casado
- Viudo
- Divorciado

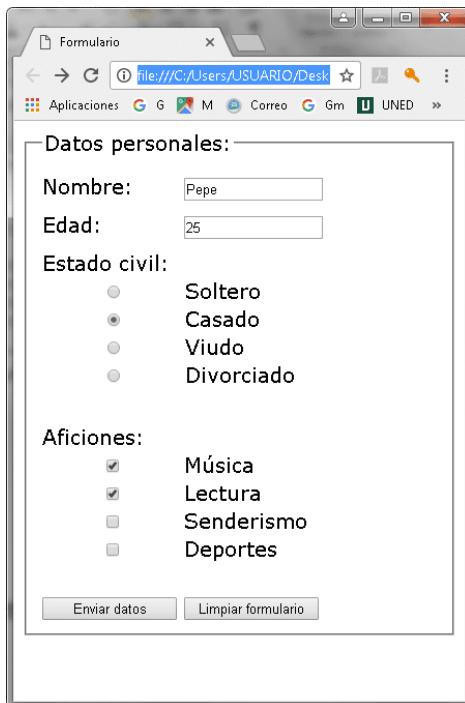
El usuario solamente podrá seleccionar uno de los cuatro valores. La variable que recogerá el programa *procesa.jsp* será *ec* y el valor enviado será *S*, *C*, *V* o *D* según el botón de *radio* seleccionado por el usuario.

<input type="checkbox">

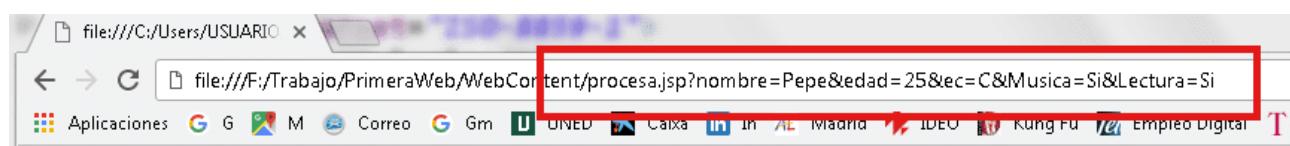
Se representa como una casilla de verificación que el usuario podrá marcar y desmarcar.

```
<label for="ec">Estado civil: </label>
<br />
<input type="radio" name="ec" value="S" />Soltero<br />
<input type="radio" name="ec" value="C" />Casado<br />
<input type="radio" name="ec" value="V" />Viudo<br />
<input type="radio" name="ec" value="D" />Divorciado<br />
<br />
```

Si has ido añadiendo todos los ejemplos a tu página, el formulario se visualizará así:



Al pulsar el botón "Enviar datos" con los datos especificados en la imagen, la URL transportará los datos, que podrá recoger el programa de servidor *procesa.jsp*.



Parámetros enviados a través de la URL al recurso de servidor *procesa.jsp*. Como no existe el recurso, se producirá un error en la página, pero te sirve igualmente para comprender el mecanismo de las peticiones HTTP.

<textarea>

Se representa como una caja de texto multilínea. Tiene los siguientes atributos:

- **rows:** número de filas visibles del campo.
- **cols:** ancho del campo.
- **readonly:** solo lectura.

```
<label for="observaciones">Observaciones: </label>
<br />
<textarea rows="5" cols="80" name="observaciones"></textarea>
```

Resultado:

Observaciones:

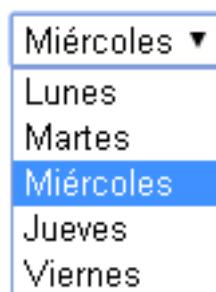
En un lugar de la Mancha, de cuyo nombre no quiero acordar-me, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocin flaco y galgo corredor. Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados, lantejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda. El resto

<select><option>

La etiqueta `<select>` representa una lista desplegable de opciones definidas por medio de etiquetas `<option>`.

```
<select name="dia">
    <option value="LUN">Lunes</option>
    <option value="MAR">Martes</option>
    <option value="MIE" selected>Miércoles</option>
    <option value="JUE">Jueves</option>
    <option value="VIE">Viernes</option>
</select>
```

El texto contenido en cada `<option>` es el valor visible para el usuario, tal y como puedes apreciar en la imagen:



El valor que se enviará al servidor es el que contiene el atributo *value* de la opción seleccionada por el usuario (*LUN*, *MAR*, *MIE*, *JUE* o *VIE*).

Nuevos atributos para campos de formularios HTML5

Los formularios son uno de los ejemplos más claros donde pueden apreciarse las mejoras que ha aportado HTML5, con nuevos atributos y nuevos tipos de campos.

En este apartado vamos a ver los nuevos atributos que HTML5 ha aportado para la etiqueta *<input>*.

placeholder

Un texto que se muestra al hacer clic sobre una caja de texto para guiar al usuario y que desaparece cuando se sitúa el cursor.

Edad:

```
<input type="text" name="edad" placeholder="Escribe tu edad"/>
```

autofocus

El campo al que se le añade el atributo *autofocus* será el que esté seleccionado por defecto al cargar la página, es decir, tendrá el cursor.

```
<input type="text" name="edad" autofocus />
```

required

Hace que el campo al que se le aplica sea obligatorio. No se podrá enviar el formulario sin rellenarlo.

En la imagen, si se intenta enviar el formulario sin llenar la edad, sale un mensaje de advertencia y no permite enviar.

Nombre:	<input type="text" value="Pepe"/>
Edad:	<input type="text"/>
Estado civil	 Completa este campo
	<input type="radio"/> Soltero
	<input type="radio"/> Casado

```
<input type="text" name="edad" required />
```

Nuevos tipos de campos HTML5 con validación automática

En este apartado veremos nuevos valores que se pueden aplicar al atributo *type* de la etiqueta `<input>`. Se trata de valores nuevos en HTML5 y no funcionan con todos los navegadores, y aunque funcionen, la presentación será distinta entre unos y otros. El navegador validará la entrada de estos campos antes de enviar los datos al servidor.

<input type="email">

Campo que solo admite una dirección de correo electrónico (*e-mail*) correcta.

<input type="tel">

Solo admite un teléfono en el formato correcto.

<input type="url">

Solo admite una URL en el formato correcto.

<input type="number" min=0 max=10 step=1>

Admite un número del 0 al 10; esto queda determinado por los valores *min* y *max*. En varios navegadores además muestra un control deslizante o *slider*. *Step* es el valor que se suma y resta al utilizar los botones del *slider*. Si queremos una caja de texto que admita números decimales tendremos que poner un valor decimal en la propiedad *step* (por ejemplo: *step=0.01*). Si cambias el campo *edad* como tipo *number* quedará así:

Edad:

<input type="range" min=0 max=10>

Genera un número del 0 al 10, pero no se visualiza como una caja de texto, sino como una barra con un deslizador.



Estructura de un documento HTML

En el [siguiente vídeo](#) te mostramos cómo crear la estructura de una página web para realizar una aplicación de gestión de tareas.

Puedes consultar el listado de etiquetas HTML en el siguiente enlace:

https://developer.mozilla.org/es/docs/HTML/HTML5/HTML5_lista_elementos

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

En esta unidad hemos realizado un recorrido por las principales etiquetas de HTML5. Gracias a estas etiquetas podremos dotar de contenido nuestros documentos web, aplicándoles la estructura idónea.

Es importante entender que con HTML5 las etiquetas solo se ocupan de estructurar la información y darle sentido semántico al contenido, siendo CSS el encargado de darle el aspecto visual adecuado.

Una de las principales dificultades que nos encontramos cuando empezamos nuestro estudio de esta nueva versión de HTML5, es el encontrar documentación y ejemplos con otras versiones anteriores, que incluyen atributos y etiquetas que han sido eliminadas en esta nueva versión.

Con HTML5, aunque tenemos etiquetas nuevas, como las semánticas de contenidos `<header><section>`, etc. el principal cambio es la simplificación.

Al unir las tecnologías HTML, CSS y JavaScript podemos realizar de forma más adecuada muchas de las acciones que antes hacíamos con HTML.

Nombre de archivo: MP_0373_UF2_WBT2.docx
Directorio: /Users/anak81/Desktop/Telefonica/IANIRE
Plantilla: /Users/anak81/Desktop/Telefonica/IANIRE/Maquetacion
Rise:Graficos/Estilos_Plantillas/Plantillas/Word/Plantilla_PDF_Lesson.tx
Título:
Asunto:
Autor: Usuario de Microsoft Office
Palabras clave:
Comentarios:
Fecha de creación: 26/3/18 15:44:00
Cambio número: 2
Guardado el: 26/3/18 15:44:00
Guardado por: Ana Carolina Landi
Tiempo de edición: 1 minuto
Impreso el: 26/3/18 15:44:00
Última impresión completa
Número de páginas: 41
Número de palabras: 8.319 (aprox.)
Número de caracteres: 45.758 (aprox.)

2.3. Hojas de estilo en cascada



Índice

Objetivos	4
Conceptos	5
¿Qué es una hoja de estilos?	5
Tipos de CSS.....	6
¿Cómo aplicamos los estilos a la página?	7
Aplicar estilos dentro de la página con una etiqueta <style>	7
Estilos en línea usando atributo style de las etiquetas HTML.....	7
Estilos definidos en una hoja de estilo independiente.....	8
Sintaxis CSS	8
La herencia.....	9
Herencias no deseadas	10
La cascada.....	10
Unidades en CSS.....	11
Tipos de unidades.....	12
Reglas en CSS	13
Principales propiedades.....	13
Referentes al texto.....	13
Referentes a la alineación	15
Referente a contenedores.....	16
Referente a listas.....	17
Márgenes y relleno. Atributos margin y padding.....	18
Especificación de tamaños mínimos.....	19
Posicionamiento de contenedores	21
Capas y posicionamiento	21
Tipo de posicionamiento. Atributo display	26
Propiedad overflow.....	30
Z-index	31
Ejemplo: estructura página HTML5	32
Otros efectos.....	33
Bordes, sombras y degradados	33

Bordes redondeados.....	33
Sombras	33
Degradiados	34
Pseudoclases y pseudoelementos.....	36
Definición.....	36
Pseudoclases	36
Pseudoelementos.....	36
Pseudoclases.....	36
Pseudoelementos	38
Diseño responsive.....	39
Media queries	39
Ejemplo de aplicación.....	43
Despedida	44
Resumen.....	44

Objetivos

Los objetivos de esta unidad son los siguientes:

- Aplicar estilos CSS a los documentos HTML.
- Conocer las principales reglas CSS.
- Conocer las propiedades de los elementos en CSS.
- Adaptar los elementos de la página a diferentes dispositivos.

Conceptos

¿Qué es una hoja de estilos?

El lenguaje de marcas HTML surgió como una forma de definir un documento, pero nunca para darle formato.

Es en la versión HTML 3.2 cuando comienzan a usarse etiquetas como `` y otras que generan documentos HTML poco estructurados y con demasiadas etiquetas de formato.

Estos documentos eran de alguna forma caóticos y poco claros. No se podía distinguir de forma sencilla la semántica ni la estructura del documento.

La revisión de HTML 4 aportó la creación del etiquetado de estilos para definir, por un lado el formato, y por otro el contenido de un documento, estableciendo que:

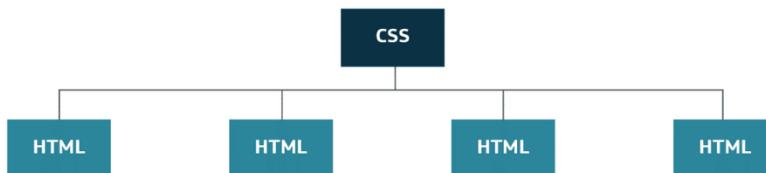
- La página web (documento HTML) solo debe contener información.
- El formato se debe definir en las llamadas hojas de estilo (CSS).
- Las hojas de estilo son estructuras de marcas que definen el formato de los elementos. Se pueden incluir dentro del archivo HTML o fuera de él **enlazando el archivo .css** en la página.
- CSS significa Cascading Style Sheets.
- CSS son los estilos que se van a aplicar en documentos de marcas como HTML.
- CSS ahorra mucho trabajo, ya que los estilos se definen en un solo lugar.



Ejemplo de aplicación de estilos a un párrafo.

Con las hojas de estilo no solo podremos hacer que nuestros documentos se vean del modo que deseemos, además definiremos la estructura de nuestra página, menús, columnas, etc.

Además, hoy en día, debemos hacer que nuestras páginas se adapten a múltiples dispositivos, y CSS será un gran protagonista en esta tarea.



Tipos de CSS

En este módulo trataremos CSS de forma estática, aplicando el estilo a los objetos de nuestro documento, pero es importante conocer que podemos dotar a nuestras hojas de estilos de dinamismo con tecnologías como SASS o LESS.

CSS Estático

Las hojas de estilo estáticas son las que usaremos en este módulo. Los elementos son configurados según unas reglas no cambiantes (estáticas), sin la posibilidad de condicionar su estado.

Características de un CSS estático:

- Cada elemento del estilo tiene sus atributos estáticos, por lo que no pueden modificarse en tiempo de carga o de ejecución.
- Los atributos no pueden ser calculados en tiempo de carga o ejecución.
- Si varios elementos tienen atributos iguales, se definen para cada uno de ellos.
- Para cambiar los estilos se accede al fichero CSS directamente y se cambia el valor del atributo.

Aunque CSS estático, podremos utilizar algunas propiedades como *:hover*, *:focus*, etc., que aunque generan algo de dinamismo al documento, se sigue considerando estático.

CSS Dinámico

Para facilitar la creación de los estilos CSS existen una serie de herramientas que permiten manejar los estilos mediante un pre-procesamiento o en tiempo de ejecución. Con esto se consigue mejorar la estructura y eficiencia de la hoja de estilos.

Para crear este tipo de CSS es necesario usar:

- **Herramientas como Stylus, LESS o SASS.** Estas herramientas se constituyen como un lenguaje de programación que realiza un pre-procesamiento de las hojas de estilos, de manera que antes de ser cargadas en la página se han calculado y definido.

- **Lenguaje JavaScript.** Con JavaScript podemos interactuar con el DOM de la página modificando los estilos al producirse un evento.

¿Cómo aplicamos los estilos a la página?

En este apartado veremos los distintos métodos para asociar estilos CSS a las páginas HTML.

Para aplicar estilos a nuestras páginas podemos hacerlo de tres formas:

- En la propia página con una etiqueta `<style>`.
- En una etiqueta por medio del atributo `style`.
- En una hoja de estilos independiente. Este es el método recomendado y el que utilizaremos con más frecuencia durante el curso.

Aplicar estilos dentro de la página con una etiqueta `<style>`

```
<head>
<meta charset="UTF-8">
<title>Mi primera web</title>
<style type="text/css">
    p {
        color: blue;
        font-family: 'Comic sans MS', Verdana;
        font-size: 18px;
    }
</style>
</head>
```

La etiqueta HTML `<style>` permite encerrar especificaciones de estilo en formato CSS. Este sistema no es recomendable a nivel de productividad, pues impide la reutilización del código y dificulta las tareas de mantenimiento. Además es penalizado por el posicionamiento SEO.

Se suele usar para piezas de diseño como boletines informativos (*newsletter*) o bocetos.

Estilos en línea usando atributo `style` de las etiquetas HTML

```
<p style='background-color: aqua; color: blue'>Esto es un párrafo</p>
```

Este sistema **tampoco es recomendable** porque sobrecarga y complica el contenido de las etiquetas HTML del documento innecesariamente.

Estilos definidos en una hoja de estilo independiente

Este **sistema es el más recomendable** y consiste en la creación de un **fichero independiente con extensión CSS**, que será denominado “**archivo de hoja de estilo**”. Siguiendo con la idea de mantener cierto orden en la estructura de la web, las hojas de estilo deberían guardarse en una carpeta específica.

Para enlazar el archivo HTML con el de la hoja de estilos .css, usaremos la etiqueta ***link***, indicando **la relación** que tiene con este archivo (***rel="stylesheet"***), el **tipo** de documento (***type="text/css"***) y la **ruta** en la que se encuentra (***href="estilos/estilo.css"***).

```
<head>
<meta charset="UTF-8">
<title>Mi primera web</title>
<link rel="stylesheet" type="text/css" href="estilos/estilo.css">
</head>
```

Para comenzar a crear nuestra hoja de estilo generaremos un fichero con el editor que se haya escogido y escribiremos el conjunto de reglas CSS que queramos aplicar al documento.

```
p {
font-family: Arial, Helvetica, sans-serif;
font-size: 17px;
text-align: justify;
text-indent: 40px;
margin: 10px 10px 10px 10px;
color:#red;
}
.verde {
color:green;
}
```

Sintaxis CSS

La sintaxis de un estilo es muy simple e intuitiva. Su estructura consta de una serie de reglas que describen la forma en la que se visualiza cada uno de los elementos.

El Selector

Hace referencia a uno o varios elementos HTML; en el ejemplo afectará a las cabeceras de nivel 1.



Propiedad

Con la propiedad se identifica el elemento que se va a modificar. Por ejemplo el color, la alineación, el margen, etc.

La Declaración

Sirve para indicar cómo se va a ver el selector; es un conjunto de propiedades a las que se les asigna un valor.

Valor

En el valor se define cómo es la propiedad. En este ejemplo estamos "alineando el Título 1 a la derecha".

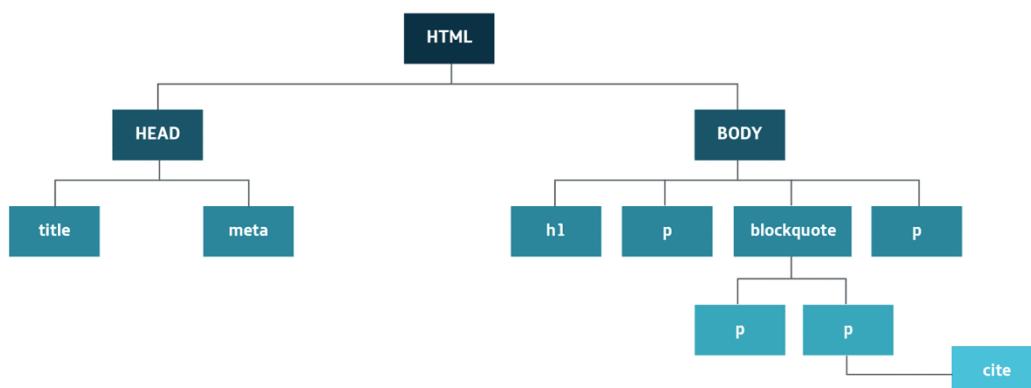
Cada propiedad acepta un tipo de valor, pudiendo ser un número, el valor hexadecimal de un color, etc.

La herencia

La herencia o *inheritance* es una de las características principales de los estilos.

Para facilitar la comprensión del concepto de herencia debemos entender que un documento HTML se puede representar como un árbol genealógico.

```
h1 {  
    font: 1em/1.5 Verdana, Helvetica, sans-serif;  
}  
p {  
    font: 1em/1.5 Verdana, Helvetica, sans-serif;  
}  
Blockquote {  
    font: 1em/1.5 Verdana, Helvetica, sans-serif;  
}  
Cite {  
    font: 1em/1.5 Verdana, Helvetica, sans-serif;  
}
```



Algunas de las propiedades de las hojas de estilo se propagan por los descendientes de los elementos. Por ejemplo, en un documento sencillo de estilos podríamos definir las características del documento así:

```
body {  
    font: 1em/1.5 Verdana, Helvetica, sans-serif;  
}
```

En este ejemplo establecemos a nivel de *body* que la fuente tiene un tamaño y una tipografía específica, afectando a todos los elementos de texto.

Aun así, **no todas las propiedades se heredan**. Una de las formas de saber qué propiedades son heredables y cuáles no es leer la especificación de W3C, que nos proporciona la información necesaria para saber si “inherited” con un “yes” o con un “no”.

Herencias no deseadas

Algunas veces este tipo de comportamiento hace que heredemos propiedades no deseadas, ya que con la cascada unas anulan a otras. Para solucionar este problema, existe el atributo *!important* que fuerza este cambio si el explorador no aplica la nueva regla.

```
ul {  
    margin: 10px !important;  
}
```

Aunque este atributo es muy útil y nos soluciona los problemas de herencias, su utilización no es muy recomendable.

La cascada

El mecanismo más importante de CSS es la cascada (*Cascading Style Sheets*).

La cascada nos sirve para saber qué ocurre si dos reglas diferentes asignan la misma propiedad al mismo elemento.

Se encarga de controlar el resultado final cuando surgen conflictos sobre qué estilos se aplican y en qué orden de preferencia en un elemento.

Para saber cómo se aplican las reglas de la cascada debemos tener en cuenta tres aspectos importantes:

Especificidad

Se produce cuando dos declaraciones tienen la misma importancia, bien porque ambas cuenten con *!important*, bien porque ambas carezcan de él.

La especificidad de los elementos se realiza con un cálculo, que se puede representar como cuatro valores separados por comas. Para ello se tienen en cuenta los siguientes aspectos:

1. El número de estilos en línea aplicados al elemento a través del atributo *style* en el marcado.
2. El número de selectores de la regla de estilo que son un identificador (*id*).
3. El número de selectores que son una clase (*class*), un selector de atributo o una pseudoclase.
4. El número de selectores que son un elemento o un pseudoelemento.

Importancia

Es posible que una de las declaraciones que utilicemos en nuestros documentos sea lo bastante importante para que se aplique, independientemente de que otra regla pueda afectarla.

Para eso utilizamos *!important*, que indica la importancia de la declaración que queremos mantener pase lo que pase.

Debemos incluirla antes del punto y coma final para que surja efecto.

Orden

El orden básicamente hace que, si un elemento tiene dos reglas con el mismo peso, tenga preferencia la última que ha sido especificada.

Unidades en CSS

Para establecer la medida de espacios, tamaños o márgenes, en CSS tenemos diferentes unidades de medida.

Todas las medidas se indican como un **valor numérico entero o decimal seguido de una unidad de medida**.

```
p {  
    font-size:18px;  
}
```

Tipos de unidades

Existen dos tipos de unidades, absolutas y relativas.

Unidades absolutas

- **in**, pulgadas ("inches"). Una pulgada son 2.54 centímetros.
- **cm**, centímetros.
- **mm**, milímetros.
- **pt**, puntos. Un punto equivale a unos 0.35 milímetros.
- **px**, píxel. Relativa respecto de la resolución de la pantalla del dispositivo en el que se visualiza la página HTML.
- **pc**, picas. Una pica equivale a 12 puntos, es decir, a unos 4.23 milímetros.

Unidades relativas

- **em**, relativa respecto del tamaño de letra del elemento.
- **%**, el tanto por cien relativo al tamaño de la ventana donde se está visualizando el documento.

Reglas en CSS

Principales propiedades

Las propiedades nos ayudan a especificar qué se va a modificar. Hay un gran número de propiedades, vamos a ir viendo las más importantes.

Referentes al texto

font-family

Tipo de fuente.

Valores: Arial; Helvetica; sans-serif; serif; Times New Roman; Times; Verdana; Georgia; Geneva; Courier; Corier New ...

```
selector {  
    font-family: 'arial';  
}
```

Las tipografías utilizadas deben ser las incluidas en los sistemas operativos, pues usará el instalado en el cliente. En CSS3 existe la posibilidad de usar tipos de letra personalizados, aunque nos penalizará en el peso de la página.

font-size

Tamaño de la fuente.

Valores:

- Unidades.
- Porcentaje.

```
selector {  
    font-size:19px;  
}
```

font-style

Estilo de la fuente.

Valores:

- *italic*;
- *normal*;

font

Permite definir en una única propiedad el tamaño de la letra, el espacio entre renglones y la familia tipográfica. Se integra en una línea siguiendo:

propiedad: Tamaño Texto / Interlineado Fuente;

```
p {  
font: 1em/1.5em Verdana, Helvetica, sans-serif;  
}
```

font-variant

Permite modificar cómo se visualiza el texto.

Valores:

- *normal;*
- *small-caps;* (versalitas)

font-weight

Aplica el formato de "negrita" al texto.

Valores:

- valores absolutos: 100, 200, 300 ... 900
- *bold;*
- *normal;*

```
selector {  
font-weight:bold;  
}
```

text-decoration

Permite definir diferentes elementos para aplicar al texto, como subrayado o tachado.

Valores:

- *blinks;* (parpadeante)
- *line-through;* (tachado)
- *underline;* (subrayado)
- *overline;* (sobrerayado)

text-transform

Transformación del texto.

Valores:

- *capitalice;* (convierte la primera letra de cada palabra a mayúsculas).
- *uppercase;* (todo en mayúsculas)
- *lowercase;* (todo en minúsculas)

color

Color del elemento.

Valores: nombre del color, valor del color.

[En este ejemplo puedes ver aplicadas estas propiedades.](#)

Referentes a la alineación

letter-spacing

Espacio entre letras.

Valores: unidades.

word-spacing

Espacio entre palabras.

Valores: unidades.

line-height

Espacio entre renglones.

Valores: unidades, porcentaje.

text-align

Alineación horizontal del texto.

Valores:

- *left;*
- *right;*
- *center;*
- *justify;*

text-indent

Sangrado.

Valores: unidades, porcentaje.

[En este ejemplo puedes ver aplicadas estas propiedades.](#)

Referente a contenedores

border

Para el borde de los elementos, en la misma regla podemos indicar el grosor, el color y el tipo.

```
div {  
    border: 1px #000 dashed;  
}
```

También podemos dar aspecto a cada uno de los cuatro bordes por separado. Para cada borde se puede establecer su anchura, su color y su estilo.

```
div {  
    border-top-width: 10px;  
    border-right-width: 5px;  
    border-bottom-width: 20px;  
    border-left-width: 6px;  
}
```

Para definir el grosor.

```
div {  
    border-top-color: #CC0000;  
    border-right-color: blue;  
    border-bottom-color: #00FF00;  
    border-left-color: #CCC;  
}
```

Para definir el color.

```
div {  
    border-top-style: dashed;  
    border-right-style: double;  
    border-bottom-style: dotted;  
    border-left-style: solid;  
}
```

Para definir el estilo del borde

width y height

Ancho y alto de la caja.

Valores:

- unidades.
- porcentaje.

background-color

Color de fondo.

Valores: nombre del color, valor del color.

background-image

Poner imagen de fondo.

Valores: *url* ('URL').

```
div {  
  background-image: url ('../directorio/foto.gif');  
}
```

[En este ejemplo puedes ver aplicadas estas propiedades.](#)

Referente a listas

list-style-type

Para el tipo de símbolo (listas sin orden) o tipo de numeración (listas ordenadas) que precede a cada elemento de una lista.

Valores:

- *circle*;
- *disc*;
- *square*;
- *decimal*;
- *lower-alpha*;
- *lower-roman*;
- *upper-alpha*;
- *upper-roman*;
- *none*;

list-style-image

Para utilizar una imagen como símbolo de elementos de una lista.

Valores:

- *url* (localización de la imagen);
- *none*;

list-style-position

Determina la forma de sangrado de las listas anidadas.

Valores:

- *inside*;
- *outside*;

[En este ejemplo puedes ver aplicadas estas propiedades.](#)

Observa en el ejemplo que estamos aplicando las propiedades sobre el selector "*ul*" y "*ol*", pudiendo aplicar también estilos a los elementos "*li*" de cada tipo de lista.

Márgenes y relleno. Atributos margin y padding

El **márgen** es el espacio que hay entre el borde del elemento y los elementos que tiene alrededor, es decir, es un espacio que está en el exterior del elemento al que se aplica.

El **relleno** o **padding** es el espacio que hay entre el elemento y su contenido, es decir, es un espacio que está en el interior del elemento.

Es muy importante saber que el *padding* se suma al ancho y alto del elemento.

Puedes ponerlo en práctica con una sencilla página con tres capas.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="ISO-8859-1">
    <title>Margin y Padding</title>
    <link href="estilos/estilo.css" rel="stylesheet">
</head>

<body>
    <div id="div1">Capa 1</div>
    <div id="div2">Capa 2</div>
    <div id="div3">Capa 3</div>
</body>

</html>
```

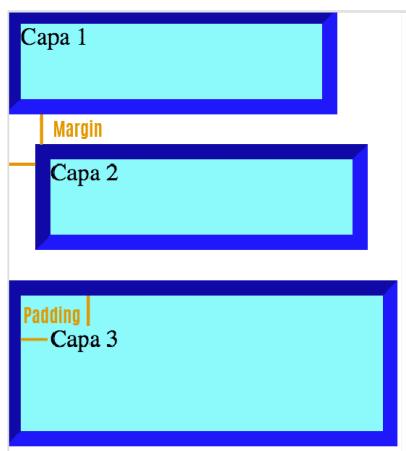
El CSS asociado:

```
@CHARSET "ISO-8859-1";

#div1, #div2, #div3 {
    width: 200px;
    height: 50px;
    border: inset 10px blue;
    background-color: cyan;
}
#div2 {
    margin: 20px;
}

#div3 {
    padding: 20px;
}
body {
    margin: 0px;
}
```

El resultado es este:



La segunda capa tiene márgenes y la tercera capa tiene relleno.

Especificación de tamaños mínimos

Hemos visto que cuando se establece el alto y ancho de un elemento en porcentaje, este se adapta al tamaño del dispositivo e incluso se redimensiona ajustándose al tamaño de una ventana del navegador. En ocasiones nos interesa que una capa tenga un tamaño mínimo pase lo que pase.

```
#todo {  
    min-width: 1000px;  
}
```

Para probarlo vamos a crear una página web muy simple con una única capa que se redimensiona y siempre se queda en el centro de la página:

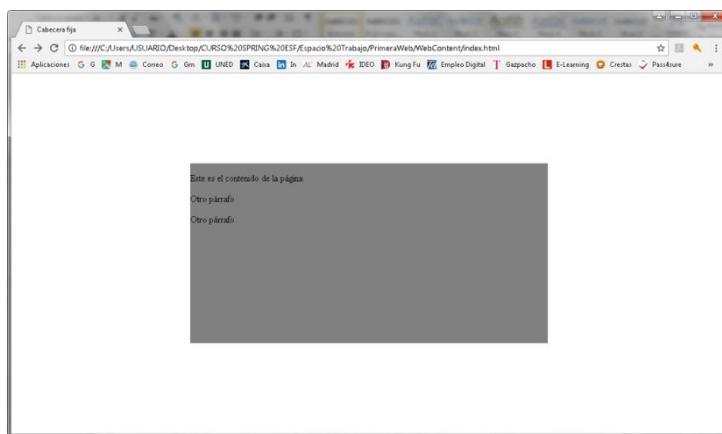
```
<!DOCTYPE html>  
<html>  
  
<head>  
    <meta charset="ISO-8859-1">  
    <title>Cabecera fija</title>  
    <link href="estilos/estilo.css" rel="stylesheet">  
</head>  
  
<body>  
    <div id="contenido">  
        <p>Este es el contenido de la página</p>  
        <p>Otro párrafo</p>  
        <p>Otro párrafo</p>  
    </div>  
</body>  
  
</html>
```

Y el CSS asociado:

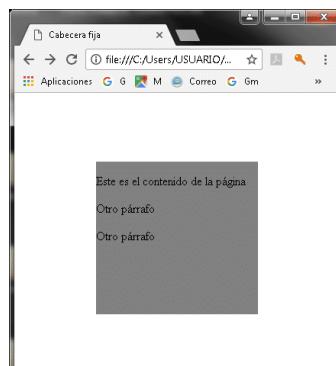
```
#contenido {  
    position: absolute;  
    width: 50%;  
    min-width: 200px;  
    height: 50%;  
    min-height: 200px;  
    top: 25%;  
    left: 25%;  
    background-color: gray;  
}
```

La capa ocupará un 50% de alto y un 50% de ancho, el otro 50% se reparte como margen, ya que dejamos un 25% arriba, un 25% abajo, y el 25% sobrante quedará a la derecha y abajo. Al redimensionar la ventana del navegador podrás apreciar cómo va cambiando la capa para adaptarse. Sin embargo, el tope está en los 200px de alto y ancho, en ningún caso la capa medirá menos que eso.

Este es el resultado:



Y cuando hacemos la ventana más pequeña:



Por mucho que reduzcas el tamaño de la ventana, la capa nunca será menor de 200px de alto y de ancho.

Posicionamiento de contenedores

Capas y posicionamiento

Para gestionar la posición de los elementos dentro de su contenedor vamos a trabajar con las siguientes propiedades CSS.

- **position:** determina el tipo de posicionamiento de los elementos en su contenedor. El contenedor de un elemento puede ser el cuerpo de la página (etiqueta `<body>`) o bien otro elemento (por ejemplo un `<div>`).
- **height:** altura del elemento.
- **width:** anchura del elemento.
- **top:** posición del elemento desde arriba.
- **left:** posición del elemento desde la izquierda.

Para comprender mejor el funcionamiento de estas propiedades vamos a ver algunos ejemplos.

Partimos de una página HTML muy sencilla con tres capas identificadas por un *id*.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="ISO-8859-1">
    <title>Los contenedores</title>
    <link href="estilos/estilo.css" rel="stylesheet" type="text/css">
</head>

<body>
    <div id="primero">Este es el primer DIV</div>
    <div id="segundo">Este es el segundo DIV</div>
    <div id="tercero">Este es el tercer DIV</div>
</body>

</html>
```

Y configuramos la hoja de estilos de la siguiente manera:

```
@CHARSET "ISO-8859-1";

#primero {
    position: relative;
    background-color: aqua;
    width: 300px;
    height: 100px;
}
```

```
#segundo {  
    position: relative;  
    background-color: gray;  
    width: 400px;  
    height: 100px;  
}  
  
#tercero {  
    position: relative;  
    background-color: aqua;  
    width: 500px;  
    height: 100px;  
    top: 100px; /* Con respecto al flujo del documento HTML */  
    left: 100px; /* Con respecto al flujo del documento HTML */  
}
```

Este es el resultado:

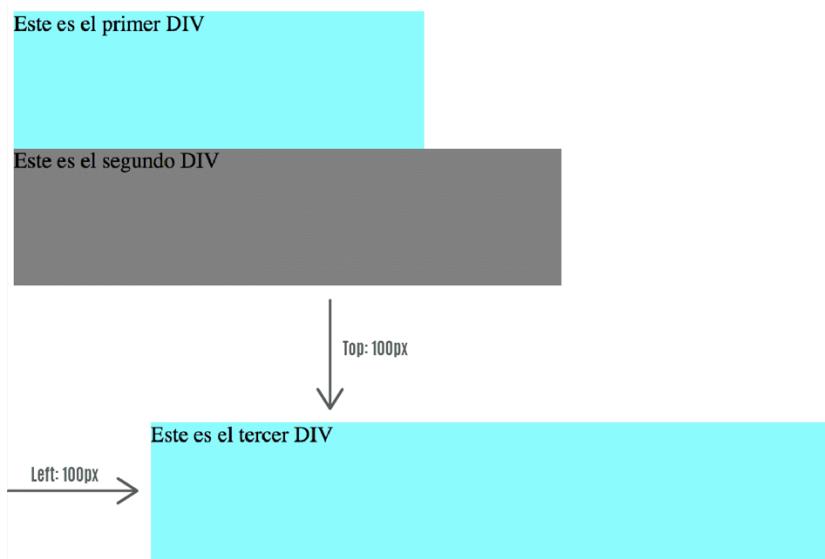
[Ejemplo HTML con tres capas identificadas por un id](#)

El ejemplo anterior utiliza capas situadas con posicionamiento relativo. Las propiedades *top* y *left* van con respecto al flujo natural que corresponde al elemento.

Position: relative

Una capa con posicionamiento relativo se sitúa según el flujo secuencial de elementos dentro del contenedor. Observa que las capas se han situado una detrás de otra.

IMPORTANTE: cuando el posicionamiento es relativo las propiedades *top* y *left* no van en relación a la esquina superior izquierda del documento, sino a partir de la posición natural que ocupa el elemento dentro del flujo secuencial HTML.



Ancho en píxel o porcentaje

Observa que en el ejemplo los atributos *height* y *width* de las capas (*<div>*) están establecidos en píxeles, lo que significa que tienen un tamaño fijo independientemente del tamaño del dispositivo donde se muestre la página. También es posible establecer los valores de alto y ancho en porcentaje para que los elementos se adapten al tamaño del dispositivo, así nos vamos acercando a lo que se denomina un diseño *responsive*.

Prueba a cambiar los anchos de las capas en 40%, 60% y 80% respectivamente para lograr que cambien en función del tamaño de la ventana donde se visualizan. Luego puedes abrir la página con un navegador e ir redimensionando la ventana para ver el efecto.

Position: absolute

Una capa con posicionamiento absoluto se sitúa dentro de su contenedor por coordenadas fijas (*top*, *left*) a partir de la esquina superior izquierda. Dentro de una capa con posicionamiento relativo puede haber otra con posicionamiento absoluto o viceversa.

Para ponerlo en práctica vamos a crear otra capa (etiqueta *<div>*) dentro de la capa cuyo *id* es *segundo*.

```
<div id="segundo">Este es el segundo DIV
    <div id="segundoB">Este es un DIV hijo</div>
</div>
```

Y añadiremos el siguiente código al fichero CSS.

```
#segundoB {
    position: absolute;
    background-color: fuchsia;
    width: 60%;
    height: 50px;
    top: 50px;
    left: 50px;
}
```

El contenedor de la capa *segundoB* es la capa *segundo* y las coordenadas y ancho establecidos van con respecto a la capa *segundo*.

A continuación tienes el ejemplo completo de HTML y CSS con el resultado: [Ejemplo position: absolute](#)

En el ejemplo, el contenedor de la capa fucsia es la capa gris y sus coordenadas van en relación a su esquina superior izquierda.

Position: static

Una capa con posicionamiento estático se coloca igual que una capa relativa, pero sus elementos hijos no serán posicionados según su esquina superior izquierda, sino según la esquina del contenedor de orden superior que no sea estático.

Para probarlo podemos hacer *static* el contenedor *segundo* para comprobar lo que ocurre con su contenedor hijo *segundoB*.

```
#segundo {  
    position: static;  
    background-color: gray;  
    width: 400px;  
    height: 100px;  
}
```

Las capas hijas de la capa *segundo* no se sitúan dentro de esta, podríamos decir que se escapan de su capa madre.

Este es el resultado:



Ahora las coordenadas de la capa fucsia van con respecto a la esquina superior izquierda del documento.

Este es el posicionamiento predeterminado. Puedes comprobar que si eliminas el atributo *position* funcionará igual.

Position: fixed

Funciona como el posicionamiento absoluto (*absolute*) con la diferencia de que la capa siempre estará visible aunque se utilice la barra de desplazamiento.

Vamos a ponerlo en práctica con este sencillo documento HTML que tiene una cabecera y un área de contenido.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="ISO-8859-1">
    <title>Cabecera fija</title>
    <link href="estilos/estilo.css" rel="stylesheet">
</head>

<body>
    <div id="cabecera">
        Esta es la cabecera de la página y quiero que esté siempre visible.
    </div>
    <div id="contenido">
        <p>Este es el contenido de la página</p>
        <p>Otro párrafo</p>
        <p>Otro párrafo</p>
    </div>
</body>

</html>
```

Queremos lograr que la capa cabecera siempre esté visible.

Y este es el CSS asociado a la página anterior:

```
@CHARSET "ISO-8859-1";

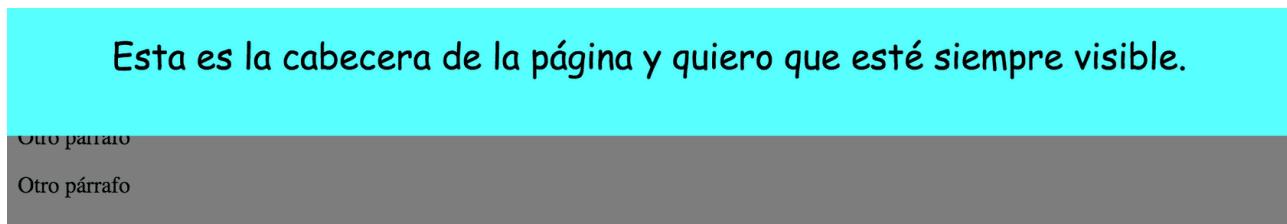
#cabecera {
    position: fixed;
    width: 100%;
    height: 75px;
    top: 0px;
    left: 0px;
    background-color: aqua;
    z-index: 2;
    text-align: center;
    font-family: "Comic sans MS";
    font-size: 25px;
    padding-top: 25px;
    /* Estos 15px se suman a la altura de del contenedor */
}
#contenido {
    position: absolute;
    width: 98%;
    top: 100px;
    left: 0px;
    height: 1300px;
    background-color: gray;
    z-index: 1;
    padding-left: 2%;
}
```

La capa *cabecera* tiene un posicionamiento fijo, lo que significa que siempre estará visible aunque el contenido sea muy grande.

A continuación puedes ver el código completo y el resultado:

Ejemplo HTML con cabecera y área de contenido

Y si utilizamos la barra de desplazamiento para ir hacia abajo, el resultado es que la capa de contenido se va escondiendo debajo de la capa de la cabecera, quedan superpuestas y el contenido en segundo plano.



Habrás observado que aparece un atributo CSS nuevo.

El **atributo *z-index*** se aplica cuando hay superposición de elementos para indicar el orden de superposición. Para nuestro ejemplo la cabecera está en primer plano y el contenido en segundo plano.

Position: inherit

El posicionamiento del elemento al que se aplica será el posicionamiento de su contenedor, es decir, el elemento padre.

En el siguiente [vídeo](#) te mostramos un ejemplo de aplicación del posicionamiento flotante.

Tipo de posicionamiento. Atributo *display*

El atributo *display* determina cómo se disponen los elementos a lo largo de la página.

display: block

Los elementos se disponen en bloque uno debajo de otro y ocupando todo el ancho, a no ser que se especifique un valor para la propiedad *width*. Solo tiene efecto si el posicionamiento es relativo o estático. Las etiquetas *<p>* y *<div>* tienen este tipo de posicionamiento por defecto, este es el motivo de que siempre ocupen todo el ancho, aunque su contenido sea una simple palabra.

```
<body>
<p id="parrafo">Hola mundo</p>
</body>
```

Los párrafos tienen por defecto posicionamiento en bloque.

```
#parrafo {  
    background-color: aqua;  
}
```

El resultado es el siguiente:

Ejemplo display: block

display: inline

Los elementos se disponen en línea uno a continuación de otro, el tamaño se ajusta al contenido, luego no tienen ningún efecto las propiedades *width* y *height*.

```
#parrafo {  
    background-color: aqua;  
    display: inline;  
}
```

Resultado:

Ejemplo display: inline

Párrafo dispuesto *inline* (en línea). No obedece a las especificaciones de alto y ancho.

Algunas etiquetas tienen un tipo de visualización *inline* por defecto, este es el caso de las etiquetas *<a>* (enlace) o **.

display: inline-block

Los elementos se disponen en línea y en bloque, uno a continuación de otro, el tamaño es el establecido por los atributos *height* y *width*.

```
<!DOCTYPE html>  
<html>  
  
<head>  
    <meta charset="ISO-8859-1">  
    <title>Ejemplo display: inline-block</title>  
    <link href="estilos/estilo.css" rel="stylesheet">  
</head>  
  
<body>  
    <div id="div1">Capa 1</div>  
    <div id="div2">Capa 2</div>  
    <div id="div3">Capa 3</div>
```

```
<div id="div4">Capa 4</div>
</body>

</html>
```

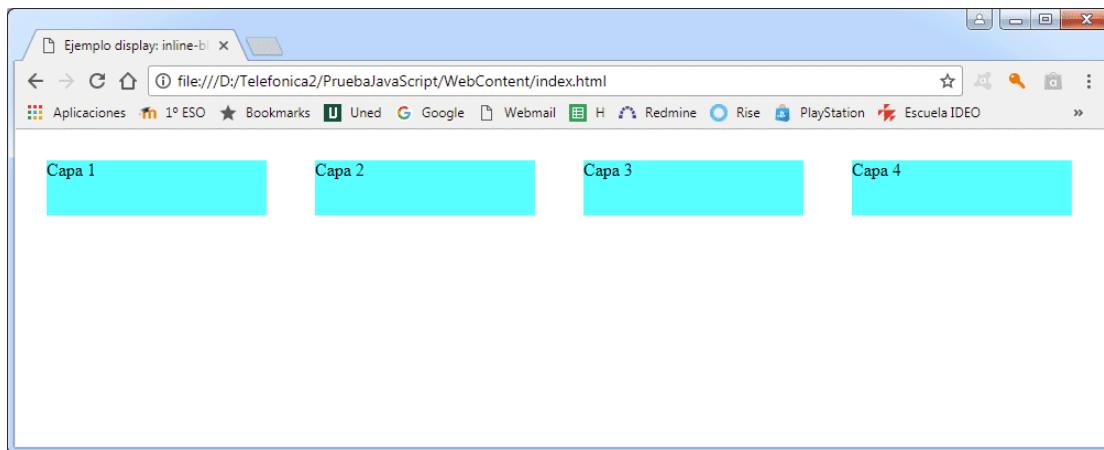
Página con cuatro capas que vamos a disponer en línea y en bloque.

```
#div1, #div2, #div3, #div4 {
    width: 200px;
    height: 50px;
    background-color: cyan;
    display: inline-block;
    margin: 20px;
}
```

Las capas se dispondrán en línea, una tras otra ocupando el espacio disponible, pero también en bloques de 200px por 50px respetando el tamaño especificado. El resultado es una página que se adapta al tamaño del dispositivo. El resultado son cuatro capas que se van distribuyendo una tras otra respetando el tamaño especificado. Una fila contendrá el número de capas que quepan según el ancho del contenedor.

Ejemplo display: inline-block

Dependiendo del tamaño de la ventana del navegador, cada capa se colocará como se ve en las imágenes. Pulsa sobre ellas para ampliarlas.



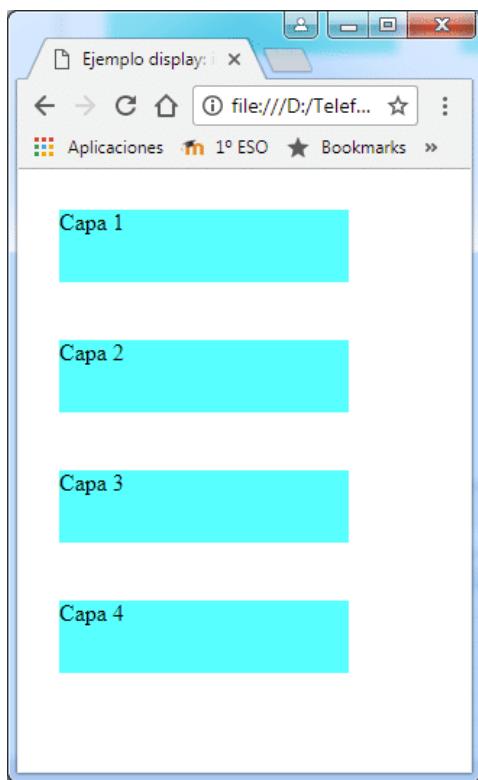
Cuatro capas en una fila.



Tres capas por fila.



Dos capas por fila.



Una capa por fila.

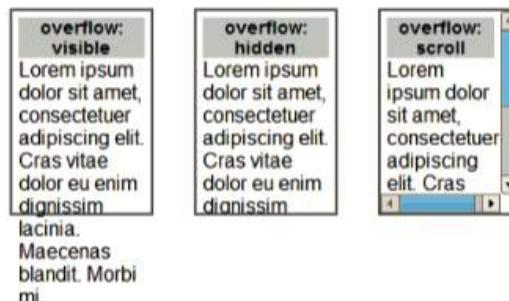
display: list-item

En bloque y en línea pero con viñetas.

Propiedad overflow

Normalmente las cajas HTML se ajustarán al contenido que tengan, pero en algunas ocasiones el contenido de un elemento no cabe dentro, dado que hemos definido un alto y ancho.

Con *overflow* podemos indicar el comportamiento que queremos que tenga el contenido.

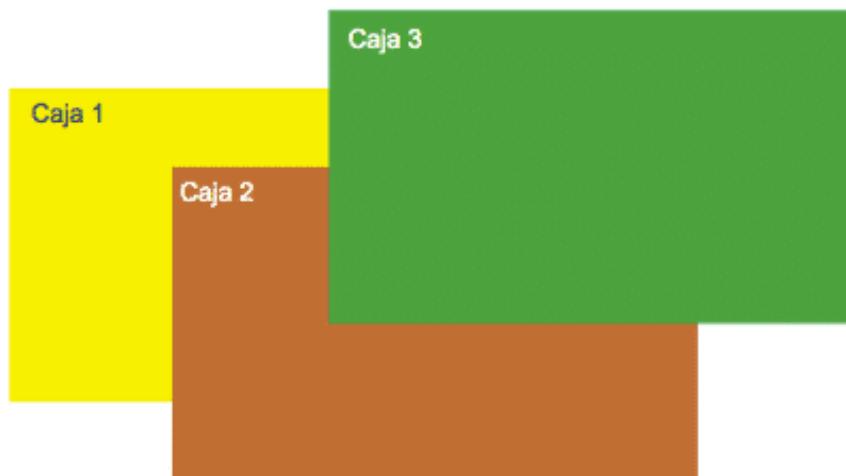


```
.caja1{
    overflow:visible;
}
.caja1{
    overflow:hidden;
}
.caja1{
    overflow:scroll;
}
```

Z-index

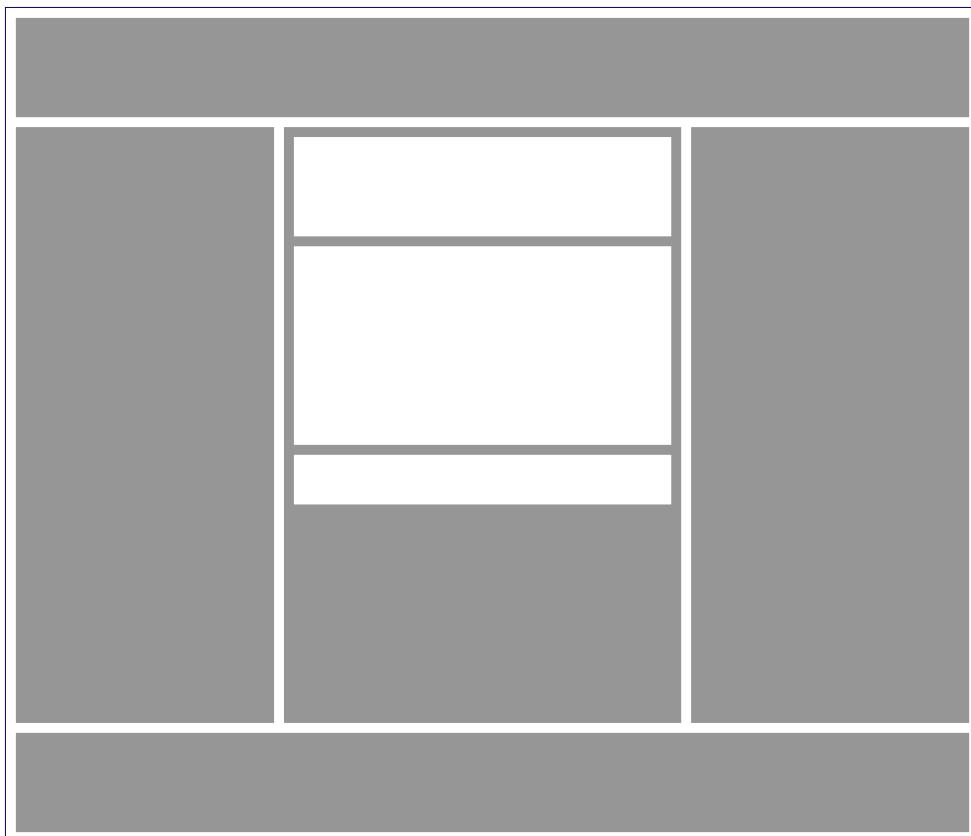
Además de la posición horizontal y vertical, podemos controlar el eje Z. Con la propiedad `z-index` podemos determinar el orden de las capas.

```
div {  
position: absolute;  
}  
#caja1 {  
z-index: 5;  
top: 1em;  
left: 8em;  
}  
#caja2 {  
z-index: 15;  
top: 5em;  
left: 5em;  
}  
#caja3 {  
z-index: 25;  
top: 2em;  
left: 2em;  
}
```



Ejemplo: estructura página HTML5

Veamos ahora un ejemplo práctico de cómo aplicaríamos las reglas CCS para crear la típica estructura de una página HTML5.



[Estructura HTML 5](#)

Otros efectos

Bordes, sombras y degradados

Los nuevos atributos de estilo de CSS3 te permiten crear bordes con sombra y rellenos degradados. Como son atributos muy nuevos, es posible que no te funcionen con todos los navegadores y si funcionan puede haber variaciones entre unos y otros.

Si empleas una versión muy antigua de navegador con seguridad no funcionará, pero no te dará problemas, simplemente el elemento se visualizará de la manera habitual, sin sombras, bordes redondeados, degradados, etc.

Bordes redondeados

Añade estos atributos a las capas del ejemplo anterior:

```
border-radius: 25px;  
padding: 10px;
```

Resultado:

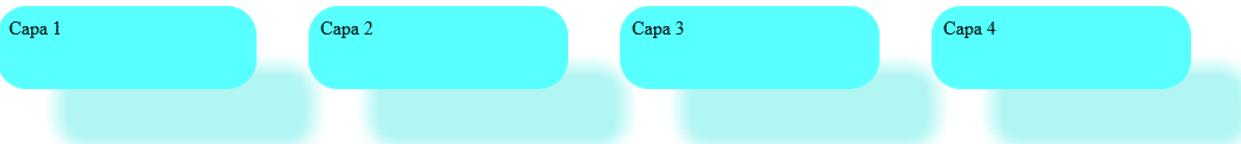


Sombras

```
box-shadow: 50px 50px 25px #A9F5F2;
```

Define una sombra en el elemento, los valores asignados son: posición de la sombra desde la derecha, posición de la sombra desde abajo, nivel de transparencia y color de la sombra.

Resultado:



Prueba a ir modificando los cuatro valores y comprueba los cambios.

Degrados

```
background: linear-gradient(left, red, blue, green);  
background: -webkit-linear-gradient(left, red, blue, green);
```

El fondo de las capas a las que se aplican está compuesto por un degradado de varios colores.

El primer parámetro es el punto de partida, estamos indicando que el degradado comience por la izquierda. El resto de argumentos son los puntos de color.

Este atributo de momento funciona con pocos navegadores, concretamente funciona con Mozilla Firefox. Observa que hemos puesto dos versiones (*linear-gradient* y *-webkit-linear-gradient*), la segunda versión es para compatibilizar con navegadores Google Chrome.

Este es el resultado:



Degrado lineal.

También puede hacerse en diagonal

```
background: linear-gradient(left top, red, blue, green);  
background: -webkit-linear-gradient(left top, red, blue, green);
```



En diagonal especificando ángulo

```
background: linear-gradient(50deg, red, blue, green);  
background: -webkit-linear-gradient(50deg, red, blue, green);
```

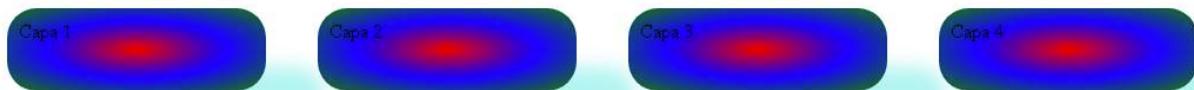
Degrado diagonal 50 grados.



Degradoado radial

Formando una elipse.

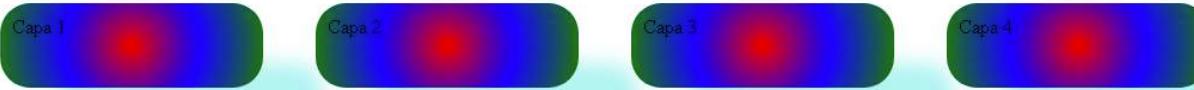
```
background: radial-gradient(red , blue, green);  
background: -webkit-radial-gradient(red , blue, green);
```



Degradoado en forma de elipse.

Círculo en lugar de elipse

```
background: radial-gradient(circle, red, blue, green);  
background: -webkit-radial-gradient(circle, red, blue, green);
```



Degradoado formando círculos.

Pseudoclases y pseudoelementos

Definición

Pseudoclases

Nos permiten dar algo de dinamismo a los elementos por medio de un evento. Se escriben detrás del selector, separadas por dos puntos.

```
selector:pseudoclase {  
    propiedad: valor;  
}
```

Puedes ampliar la información sobre las pseudoclases en este enlace.

https://www.w3schools.com/css/css_pseudo_classes.asp

Pseudoelementos

Es la posibilidad de aplicar normas de estilo ajenas a la estructura clásica de etiquetas, poniendo nuevas normas de aplicación (al primero, al último, etc.). Se escriben detrás del selector, separadas por dos puntos dobles (::).

```
selector::pseudoelemento {  
    propiedad: valor;  
}
```

Puedes ampliar la información sobre los pseudoelementos en este enlace.

https://www.w3schools.com/css/css_pseudo_elements.asp

Pseudoclases

Aplicado a los enlaces

- **:hover** - Cuando el puntero del ratón se posiciona encima del *link*.
- **:focus** - Selección del enlace usando el tabulador del teclado.
- **:visited** - Selecciona los enlaces que ya fueron visitados.
- **:active** - El momento en el que hacemos clic.

```
a:link {  
color:red;  
}
```

```
a:active {
color:green;
}

a:hover, a:focus {
color:blue;
}
a:visited {
color:yellow;
}
```

Aplicado a un *input* de formulario

- **:focus** - Cuando el cursor está dentro del campo.
- **:target** - Cuando vamos a una url con un *id* al final: <http://pagina.es#Objetivos>.
- **:enabled** - Selecciona los *input* que no tienen el atributo **disabled="true"**, es decir los que podemos editar.
- **:disabled** - Lo contrario que el anterior, los que no podemos editar.
- **:checked** - Para las *checkbox* y *radio* que están seleccionadas.

```
input:focus {
background-color: rgb(156, 153, 133);
}
input:disabled {
background-color: rgb(196, 109, 59);
}
input:checked {
cursor: crosshair;
}
input:indeterminate {
cursor: e-resize;
}
```

Para el árbol de etiquetas

- **:root** - Selecciona al elemento raíz del DOM. La etiqueta `<html>`.
- **:first-child** - Selecciona al primer hijo dentro de un elemento.
- **:last-child** - Selecciona el último hijo.
- **:nth-child(N)** - Selecciona elementos de valor de N:
 - (n) Todos los hijos.
 - (2n+1) Todos los hijos en posición impar.
 - (2n) Todos los hijos en posición par.
 - (nX) Todos los hijos a partir de X.
 - (-nX) Todos los hijos hasta X.
 - (X) El hijo en la posición X.

```
/*Selecciona a todos los p que sean primer hijo*/
p: first-child {
background-color: red;
}
```

```
/*Selecciona a cualquier elemento que sea primer hijo*/  
: first-child {  
background-color: yellow;  
}
```

Para el contenido

- **:empty** - Selecciona a todos los elementos que no tengan hijos o texto.
- **:not(X)** - Selecciona a todos los elementos que no posean una característica X.

Para el texto

- **:first-letter** - Selecciona la primera letra del texto.
- **:first-line** - Selecciona la primera línea del texto.

Puedes ver un pequeño ejemplo de alguna de estas pseudoclases.

Pseudoclases

Pseudoelementos

- **::before** - Para insertar un contenido (texto) antes de una etiqueta.
- **::after** - Para insertar un contenido (texto) después de una etiqueta.

```
p::before {  
content: "lo que queremos insertar antes";  
}  
p::after{  
content: "lo que queremos insertar despues";  
}
```

La propiedad "*content*" permite poner un texto predeterminado. En este caso se inserta el texto antes o después del elemento "*p*".

Pseudoelementos

Diseño responsive

Media queries

Las *media queries* se utilizan en la tecnología CSS para condicionar la aplicación de un grupo de atributos de diseño al cumplimiento de una condición, que tendrá que ser relativa al tipo de dispositivo, orientación, tipo de visualización, etc.

Vamos a ponerlo en práctica con este documento HTML:

```
<!doctype html>
<html lang="es">
<head>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width"/>
    <title>Contenido Responsive</title>
    <link rel="stylesheet" href="estilos/estilo.css"/>
</head>
<body>
    <section>
        <h2>fila1 columna1</h2>
        <p>trolololol</p>
    </section>
    <section>
        <article class="fila1">
            <h2>fila2 columna1</h2>
            <p>trolololol</p>
        </article>
        <article class="fila2">
            <h2>fila2 columna2</h2>
            <p>trolololol</p>
        </article>
    </section>
</body>
</html>
```

La etiqueta `<meta name="viewport" content="width=device-width"/>` representa el área visible del navegador. Es una etiqueta HTML5 que sirve para optimizar los sitios para móviles, ayudando a definir el ancho, alto y escala del área usada por el navegador para mostrar contenido.

Código CSS asociado con la página anterior:

```
@CHARSET "ISO-8859-1";
article {
    float:left;
    width:50%;
}

body {
    background:#C3E5F9;
    color:white;
    font-size:16px;
    font-family:Arial;
    text-shadow:1px 1px 0 black;
    margin:0;
}

section{
    background:#12A89D;
    margin:10px auto;
    overflow:hidden;
    padding:10px 0;
    text-align:center;
    width: 1000px;
}

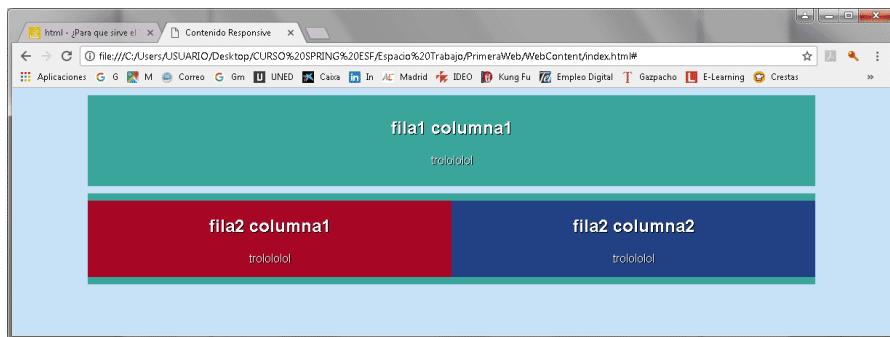
.fila1{
    background:#B30F2A;
}

.fila2{
    background:#1C4583;
}

@media screen and (max-width: 1000px) {
    section{
        width:100%;
    }
}

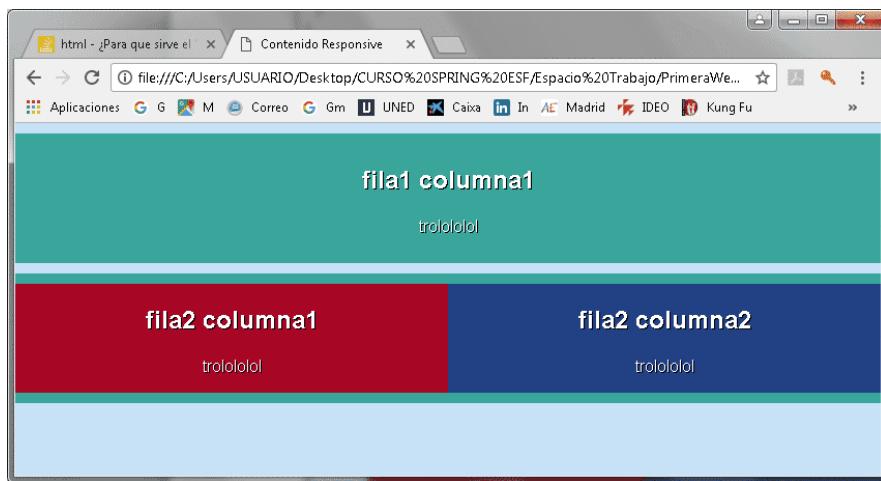
@media screen and (max-width: 700px) {
    article{
        width:100%;
    }
}
```

Hemos conseguido un diseño *responsive* que tiene tres comportamientos diferentes. Observa a continuación cómo se vería el resultado dependiendo del tamaño de la pantalla donde se visualiza.



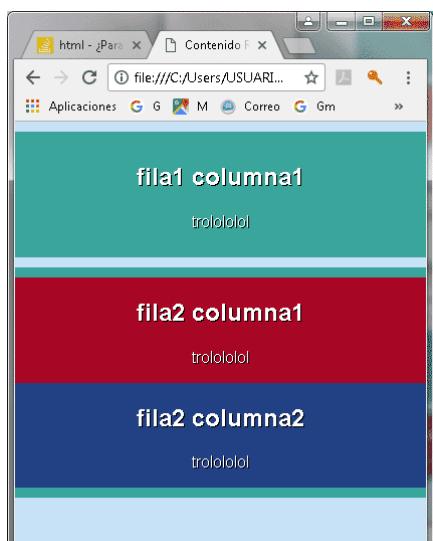
Diseño para PC

Para PC con ancho mayor o igual a 1000px deja unos márgenes a izquierda y derecha y los artículos se muestran en la misma fila.



Diseño para tabletas

Para pantallas entre 700 y 1000 pixels (normalmente tabletas) las capas ocupan un ancho del 100% de la pantalla para aprovechar mejor el tamaño.



Diseño para móvil

Para pantallas con ancho menor de 700px (normalmente móviles) los artículos se visualizan uno debajo de otro.

Las media query

Las *media query* están formadas por un **media type** y una o varias **condiciones**.

```
@media screen and (max-width: 700px) { }
```

En este caso el *media type* es *screen* y la condición de que el tamaño máximo sea 700px.

El *media type* puede ser **all**(todos), **screen** (pantalla), **speech** (lectores de pantalla) o **print** (impresión).

Podemos no especificar el *media type*, en cuyo caso será *all*.

```
@media (max-width: 700px) { }
```

Para todos los medios (*print* y *screen*) y tamaño del dispositivo menor o igual a 700px.

Podemos especificar varias condiciones:

```
@media (min-width: 700px) and (orientation: landscape) { }
```

Tamaño máximo sea 700px y la orientación horizontal.

```
@media (min-width: 700px) and (orientation: portrait) { }
```

Tamaño máximo sea 700px y la orientación vertical.

Otro ejemplo

Veamos un ejemplo CSS que cambia el tamaño de la fuente y el color de fondo en función del tamaño del dispositivo y es para pantalla o para impresión.

```
@CHARSET "ISO-8859-1";

body {
    background-color: aqua;
    font-size: 25px;
}

@media screen and (max-width: 700px) {
    body {
        background-color: gray;
        font-size: 20px;
    }
}
```

```
@media screen and (max-width: 300px) {  
    body {  
        background-color: gray;  
        font-size: 15px;  
    }  
}  
  
@media print {  
    body {  
        font-size: 50px;  
        font-family: "Comic sans MS";  
    }  
}
```

Ejemplo de aplicación

Veamos un ejemplo de adaptación de los elementos de una página con la estructura de HTML5. Podemos comprobar que dependiendo del tamaño del dispositivo los elementos cambian para adaptarse, ajustando su ancho o desapareciendo, como el caso de la adaptación a móvil.



Adaptación a móviles

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado. En esta unidad hemos visto cómo dotar de estilos nuestros documentos HTML, pudiendo realizar diseños atractivos y que aumente la usabilidad de nuestras páginas.

Es importante que entiendas que no podemos poner un ejemplo de aplicación de cada una de las reglas CSS, por lo que lo debes practicar y probar las combinaciones para conseguir el diseño deseado.

También debes tener en cuenta que hoy en día son muchos y muy variados los dispositivos encargados de visualizar el contenido, por lo que debes adaptar tus documentos HTML a múltiples tipos de pantalla, o lo que es lo mismo crear diseños *responsive*.

MP_0373. Lenguajes de marcas y sistemas de gestión de información

UF2. Lenguajes para la visualización de la información

2.4. Caso práctico



Índice

Objetivos	3
Creación de una página web	4
Preparando los archivos	4
Creando la maqueta inicial HTML5	4
El <i>head</i>	5
El <i>body</i>	6
El menú	10
Metiendo algo de contenido.....	14
Haciendo nuestra página responsive.....	17
Despedida	18

Objetivos

En esta sesión vamos a poner en práctica los conocimientos adquiridos a lo largo de la unidad.

Creación de una página web

Preparando los archivos

Para empezar nuestro proyecto, lo primero es crear los archivos y ordenarlos.

Aunque la página que vamos a crear no constará de muchos archivos, lo normal en un proyecto real es que tengamos que manejar una gran cantidad de ellos. Por eso, mantener una estructura de directorios adecuada es de suma importancia.

Archivos y carpetas

Aunque no existe una única forma de crear la estructura, en este caso aconsejamos separar los archivos HTML de los CSS con los estilos y de los de imágenes.

En un futuro podríamos querer crear funciones JavaScript u otros archivos, para los cuales también crearíamos su propia carpeta.



Estructura inicial de nuestra organización de archivos.

El archivo de inicio de nuestra página es index.html. Este nombre de archivo es el usado por defecto por los servidores para localizar el comienzo de la jerarquía del sitio web (la *home* o página de inicio).

Creando la maqueta inicial HTML5

El siguiente paso es crear la estructura principal de la página. En nuestro caso vamos a continuar con el esquema de la típica página web en formato blog.

Donde tenemos:

- Una cabecera (*head*) para poner el nombre del sitio, logos, etc.
- Un menú para insertar los diferentes enlaces a otras páginas del sitio.
- Una sección principal, donde pondremos el contenido principal.
- Un *aside* (sección de contenido relacionado) para los elementos secundarios como *banners*, *call to action*, listados, etc.

- Un *footer* (pie de página) donde insertaremos el *copyright* y otros elementos como menús secundarios, acceso a redes sociales, etc.

El *head*

Como ya hemos comentado, toda página se divide en un *head* (cabecera), donde pondremos el título, las llamadas a archivos externos, etc. y el *body* (cuerpo), donde irá el contenido visible de la página. En este caso veamos nuestro *head*.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi primera página web</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="keywords" content="Palabras clave para ayudar a saber el tema sobre el que trata la página" />
    <meta name="description" content="Descripción de la página" />

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

  </head>
  <body>

  </body>
</html>
```

En el código podemos ver cómo hemos insertado, además de la llamada al archivo CSS, etiquetas *meta* con información del documento como la codificación de caracteres, la escala inicial del documento, la descripción del contenido de la página, etc.

Aunque la *meta keywords* ya no es relevante para el posicionamiento en buscadores, la *description* sí es de utilidad para el SEO.

El **body**

Ahora insertaremos las etiquetas semánticas HTML5 para poner luego el contenido. En este caso empezamos el diseño pensando en la visualización para escritorio "ancho fijo", es decir le daremos un ancho para que se ajuste a resoluciones grandes, dejando márgenes a los lados si es necesario.

```
<body>

    <div id="contenedor">

        <header>
            <!-- Cabecera de la página-->
        </header>
        <nav>
            <!-- Menú de la página -->
        </nav>

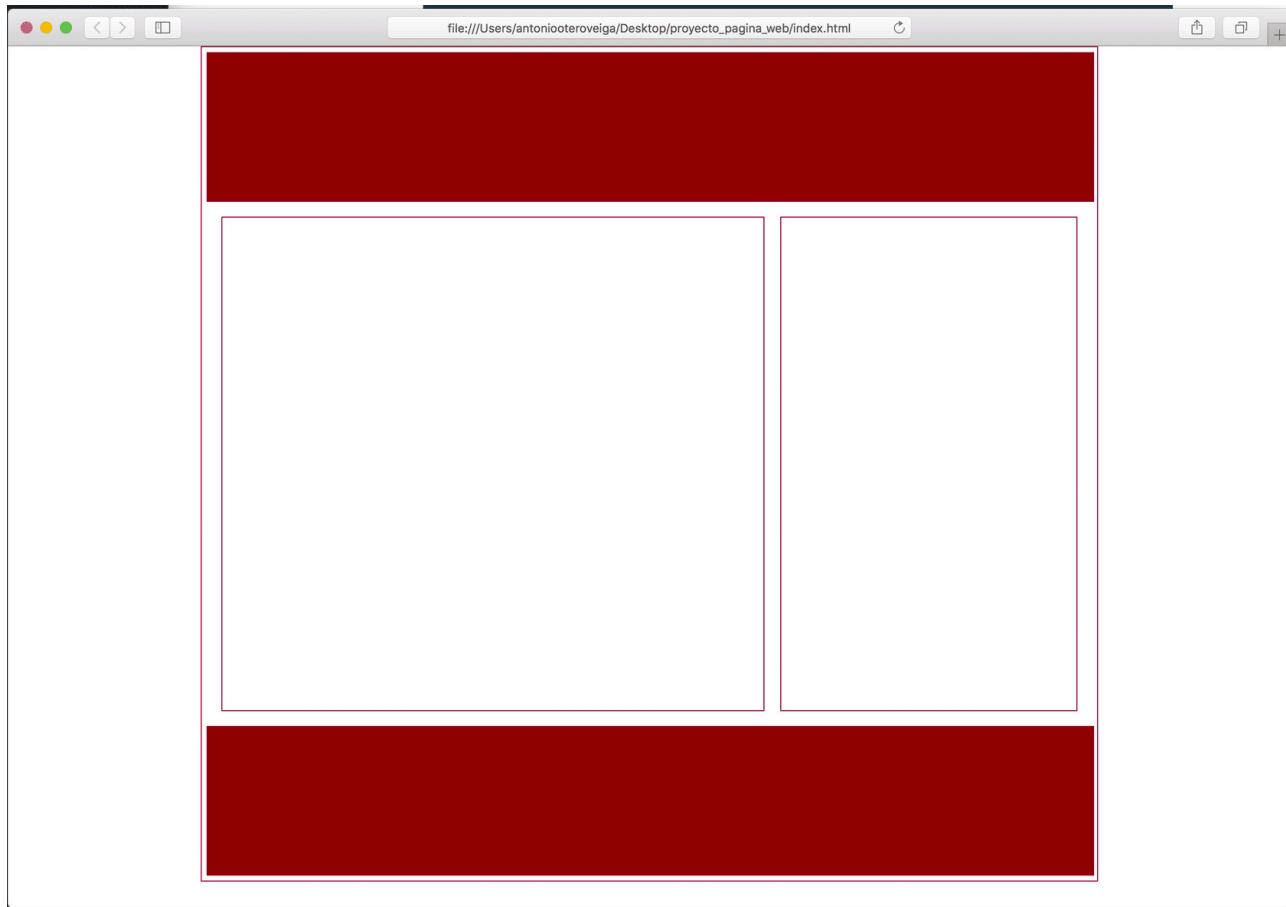
        <section>
            <!-- Contenido principal -->
        </section>
        <aside>
            <!-- información menos relevante de la página -->
        </aside>
        <footer>
            <!-- Pie de la página -->
        </footer>

    </div>

</body>
</html>
```

Aplicando CSS a los elementos principales

Ahora vamos a crear con los estilos CSS la estructura de nuestra página. Introduciremos bordes, fondos y altos a los elementos con la intención de que vayamos comprobando la estructura mientras la hacemos, aunque en algunos casos los eliminaremos cuando el documento esté terminado.



Veamos cómo quedaría la hoja de estilos.

```
@charset "UTF-8";
/* CSS Document */

* {
    margin:0px;
    padding:0px;
    font-family:Arial, Helvetica, sans-serif;
}

#contenedor{
    border: 1px solid #903;
    margin-left:auto;
    margin-right:auto;
    width:900px;
    background-color:#FFF;
    padding: 5px;
}

header {
    border: 1px solid #903;
    width:100%;
    margin-bottom: 15px;
    height: 150px;
    background-color:rgba(153,0,0,1);
```

```
        overflow: hidden;  
  
    }  
  
section{  
  
    border: 1px solid #903;  
    width: 550px;  
    float: left;  
    height: 500px;  
    margin-left: 15px;  
    margin-bottom: 15px;  
}  
  
  
aside{  
    border: 1px solid #903;  
    margin-right: 15px;  
    width: 300px;  
    float: right;  
    height: 500px;  
    margin-bottom: 15px;  
}  
  
  
footer {  
    border: 1px solid #903;  
    clear: both;  
    width: 100%;  
    height: 150px;  
  
    background-color: rgba(153,0,0,1);  
}
```

Archivo estilos.css.

Veamos ahora cada regla con más detalle.

```
*{  
  
    margin:0px;  
    padding:0px;  
    font-family:Arial, Helvetica, sans-serif;  
}
```

La primera regla que hemos incorporado está con el selector universal, dado que deseamos que se aplique a todos los elementos.

Ponemos los márgenes y *padding* a 0 para eliminar las herencias no deseadas incorporadas en párrafos, listas, etc.

Es importante entender que al eliminar estas herencias propias de los elementos HTML ganaremos en libertad a la hora de diseñar el documento, pero nos obliga a incorporar luego todos los estilos de los elementos según nuestra preferencia.

```
#contenedor{  
    border: 1px solid #903;  
    margin-left:auto;  
    margin-right:auto;  
    width:900px;  
    background-color:#FFF;  
    padding: 5px;  
}
```

Ya comentamos que hemos puesto todos los elementos en un contenedor para facilitar el diseño y hacerlo de "ancho fijo". Para este elemento hemos usado un selector de *id*.

Para centrar el elemento en la ventana del explorador los márgenes derecho y izquierdo los ponemos en *auto*, esto forzará el centrado del contenedor.

```
header {  
    border: 1px solid #903;  
    width:100%;  
    margin-bottom: 15px;  
    height: 150px;  
    background-color:rgba(153,0,0,1);  
    overflow: hidden;  
}
```

Mediante un selector de *tag* (etiqueta), damos estilo al *header*, de momento esta vacío.

En este caso se le ha puesto un alto e indicado que el contenido se adapte a su tamaño gracias al *overflow*.

```
section{  
    border: 1px solid #903;  
    width: 550px;  
    float: left;  
    height: 500px;  
    margin-left: 15px;  
    margin-bottom: 15px;  
}
```

Para el *section* le hemos puesto un ancho de 550px, también podríamos jugar con otras unidades de medida como los %.

Para la versión de escritorio se ha indicado que se coloque a la izquierda del documento poniendo *float:left*.

```
aside{  
    border: 1px solid #903;  
    margin-right: 15px;  
    width: 300px;  
    float: right;  
    height: 500px;  
    margin-bottom: 15px;  
}
```

El *aside* es similar al *section*, aunque con diferente tamaño y flotación. En este caso lo colocamos a la derecha y hemos puesto un tamaño de 300px.

Es importante dejar aire entre los elementos si el diseño nos lo permite. En el contenedor hemos puesto un ancho de 900px, y la suma del *aside* y *section* son (550px + 300px) 850px, por lo que hemos dejado entre los elementos un espacio de 50px. Esta medida dependerá del diseño que estés realizando.

```
footer {  
    border: 1px solid #903;  
    clear: both;  
    width: 100%;  
    height: 150px;  
  
    background-color: rgba(153,0,0,1);  
}
```

Para el *footer* le damos un tamaño del 100% del contenedor.

Es importante aplicar la regla *clear:both*, dado que al estar después de dos elementos flotantes, debe recuperar su posición sin flotación.

El menú

Un sitio web consta de varias páginas, por lo que resulta útil disponer de un menú que nos permita la navegación entre los diferentes documentos.

Para ello vamos a insertar una etiqueta *<nav>*, donde colocaremos los enlaces que facilitarán esta labor de navegación. Para el ejemplo que estamos elaborando hemos decidido que nuestro sitio constará de las siguientes páginas:

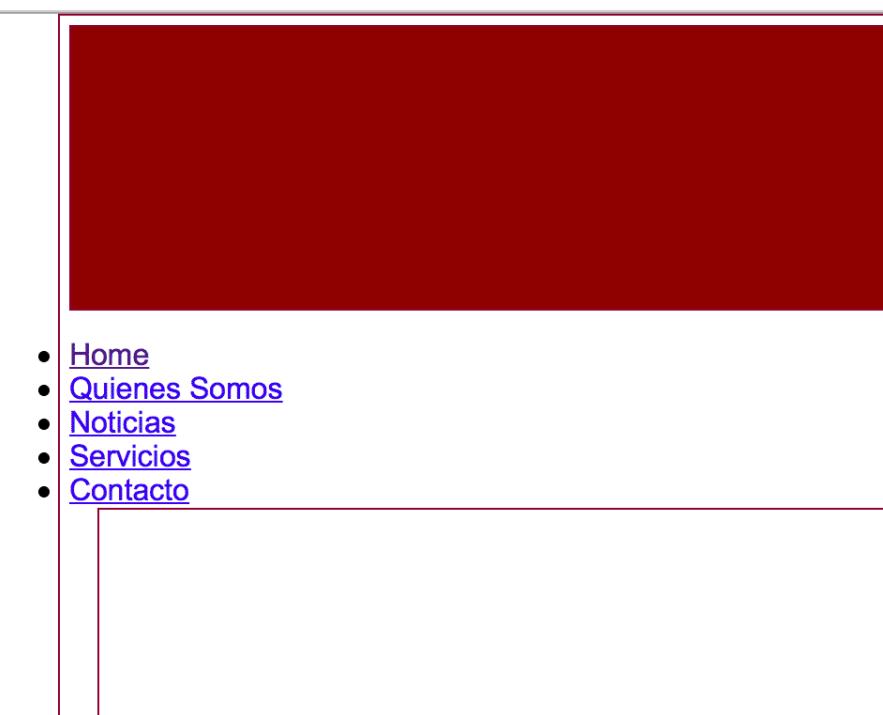
- Home (index.html).
- Quiénes somos (quienes.html).
- Noticias (noticias.html).

- Servicios (servicios.html).
- Contacto (contacto.html).

Primero insertamos en el HTML la etiqueta *nav* y una lista con los enlaces.

```
<nav>
    <!-- Menú de la página -->
    <ul>

        <li><a href="index.html">Home</a> </li>
        <li><a href="quienes.html">Quiénes somos</a> </li>
        <li><a href="noticias.html">Noticias</a> </li>
        <li><a href="servicios.html">Servicios</a> </li>
        <li><a href="contacto.html">Contacto</a> </li>
    </ul>
</nav>
```



Una vez tenemos la lista con nuestros enlaces, procederemos a aplicar los estilos para hacerlo más agradable.

```
nav{
    width: 100%;

}

nav ul{
    list-style: none;

}

nav ul li{
```

```
float: left;
width: 20%;
text-align: center;
}

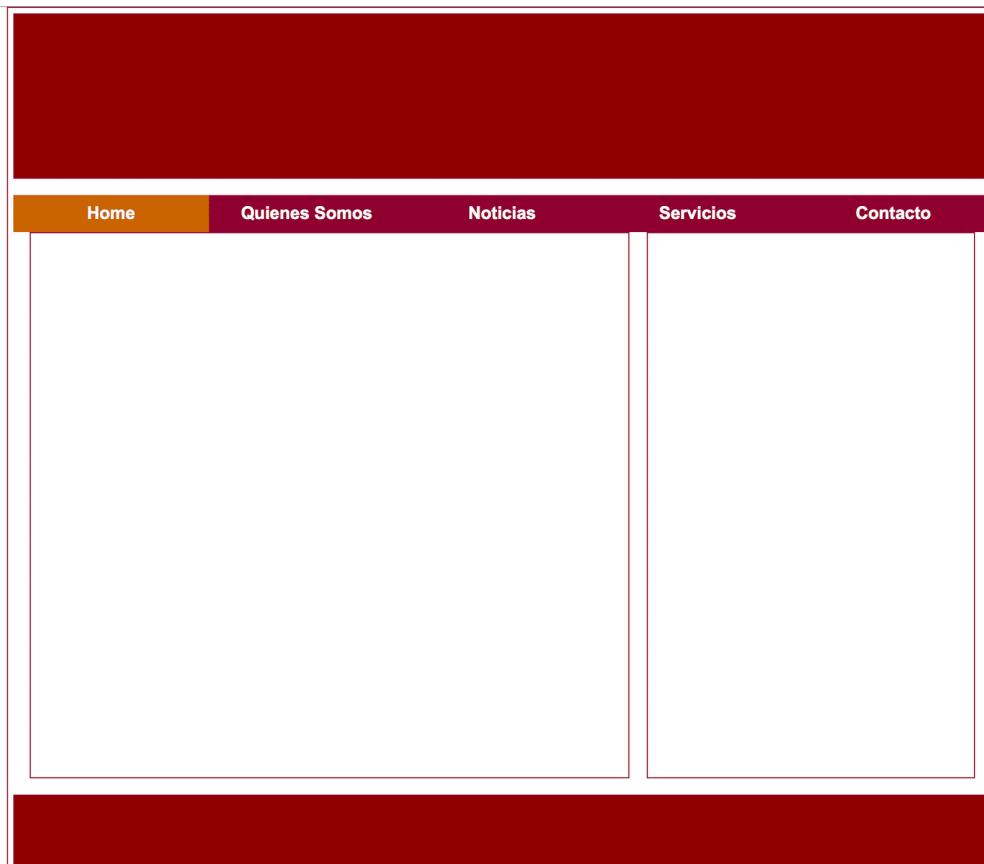
nav ul li a{

display: block;
padding-top: 8px;
padding-bottom: 8px;
font-weight: bold;
color: white;
background: #903;
text-decoration: none;

}

nav ul li a:hover{
background-color: chocolate;
}
```

Una vez aplicados los estilos, este sera su nuevo aspecto.



Examinemos con más detalle lo que hemos hecho.

```
nav{
    width: 100%;
```

}

Indicamos a la etiqueta `<nav>` que ocupe el 100% del contenedor.

```
nav ul{
    list-style: none;
}
```

Quitamos los elementos visuales que tienen las listas, como el punto.

Las listas también traen herencias como el `margin` y `padding`, no lo estamos eliminando aquí, pues ya lo hicimos en el selector universal (*), donde pusimos todos los `margin` y `padding` a 0.

```
nav ul li{
    float: left;
    width: 20%;
    text-align: center;
}
```

Dado que se trata de un menú horizontal, hacemos que los elementos `<i>` floten hacia la izquierda.

Dado que son 5 elementos en total, le hemos puesto un tamaño del 20% a cada uno para que cubran todo el espacio.

Al usar el `text-align` indicamos que el texto quede centrado dentro de cada etiqueta `<i>`.

```
nav ul li a{
    display: block;
    padding-top: 8px;
    padding-bottom: 8px;
    font-weight: bold;
    color: white;
    background: #903;
    text-decoration: none;
}
```

Para que el enlace ocupe toda la superficie del espacio reservado por la etiqueta `<i>`, convertimos la etiqueta `<a>` en bloque. De no hacerlo, no podría ocupar más que el propio texto.

Otra regla que es importante mencionar es la de `text-decoration:none`, donde quitamos herencias no deseadas, como el subrayado que traen los enlaces por defecto.

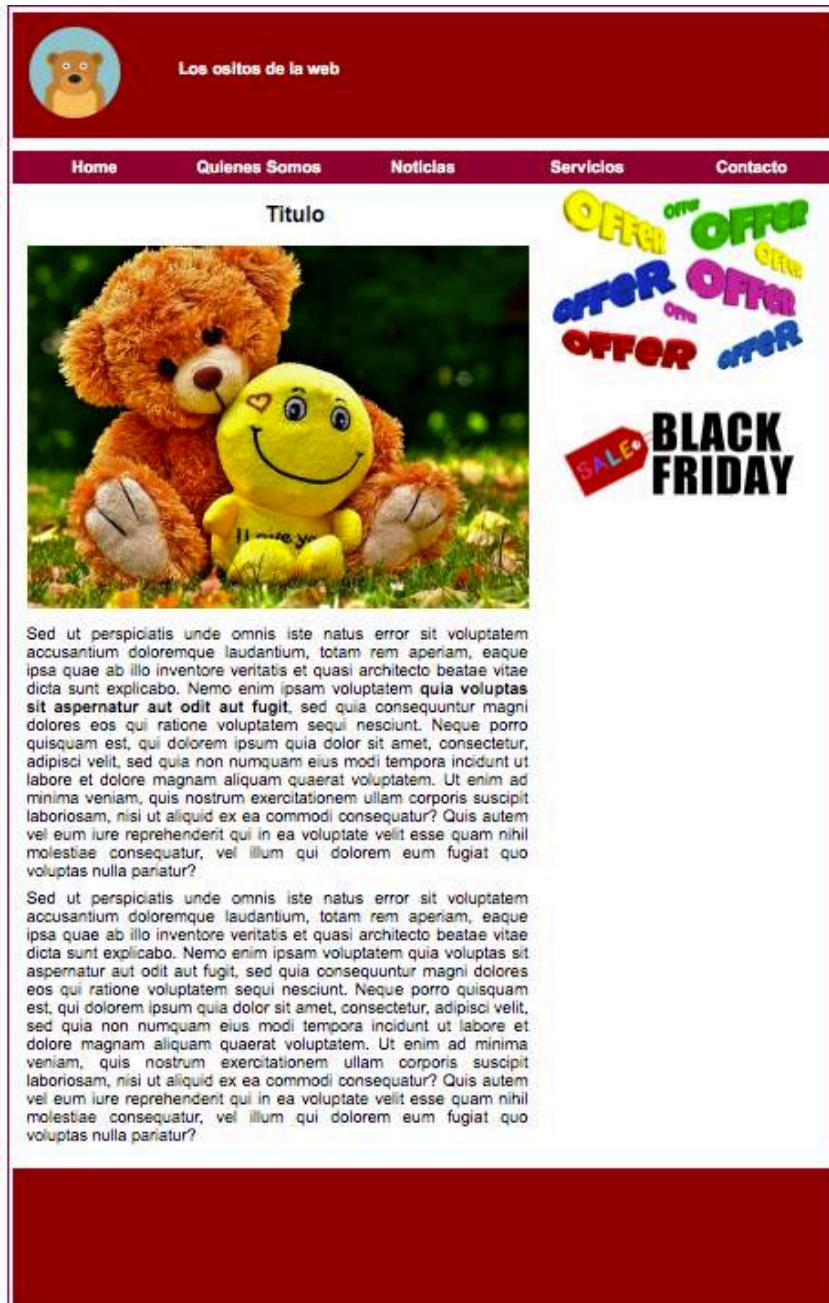
```
nav ul li a:hover{
    background-color: chocolate;
}
```

Para dotar de algo de dinamismo al menú le incluimos el comportamiento `:hover`. De esta manera, cuando el cursor del ratón se sitúe encima del elemento aplicará las reglas indicadas; en este caso un cambio del color de fondo del botón.

Mitiendo algo de contenido

Dado que resultaría algo tedioso realizar un proyecto real, vamos a dar algo de personalidad a nuestra página, aunque sea con algo de texto ficticio.

En el <header> vamos a incluir un texto y una imagen a modo de logo. Además meteremos texto en el apartado <section>, junto con alguna foto.



Llenando el `<header>`

Como ejemplo, en el `<header>` insertamos un logotipo y un texto de encabezado. Podemos ver que usamos la etiqueta `float` junto con los márgenes para situar los elementos.

Recordemos que si no tenemos el `overflow` en el estilo del `<header>`, este perdería su altura.

```
<header>
    <!-- Cabecera de la página-->
    <div class="logo">
        
    </div>
    <h3>
        Los ositos de la web
    </h3>
</header>
```

Contenido del `<header>` en el documento HTML.

```
header .logo{
float: left;
margin: 15px;
width: 100px;
}

header .logo img{
width: 100%;
}
```

Estilos para el `<header>` en el archivo CSS.

Llenando el `<section>`

Para llenar el `<section>` colocaremos una imagen y un par de párrafos. Como hemos quitado los estilos heredados tenemos que configurar los márgenes de las etiquetas `<p>`.

También se han quitado las etiquetas de borde y alto de `<section>` para que la altura de la página se ajuste al contenido.

```
<section>
    <!-- Contenido principal -->
    <h1>Tituto</h1>
    
    <p> Sed ut perspiciatis unde omnis iste natus error sit voluptatem
accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo
inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo
enim ipsam voluptatem <strong>quia voluptas sit aspernatur aut odit aut
fugit</strong>, sed quia consequuntur magni dolores eos qui ratione voluptatem
sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet,
consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut
labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam,
quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid
ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea
voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum
fugiat quo voluptas nulla pariatur?</p>
```

```
<p> Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium  
doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore  
veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam  
voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia  
consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque  
porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci  
velit, sed quia non numquam eius modi tempora incident ut labore et dolore  
magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum  
exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi  
consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit  
esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo  
voluptas nulla pariatur?</p>  
</section>
```

Contenido <section> en el documento HTML.

```
section h1{  
    width: 100%;  
    text-align: center;  
    margin: 20px;  
}  
  
section p{  
    text-align: justify;;  
    margin-bottom: 10px;  
}  
  
section img{  
    width: 100%;  
    margin-bottom: 15px;  
}
```

Estilos para el <section> en el archivo CSS.

Llenando el <aside>

En el caso de <aside> pondremos dos imágenes imitando dos *banners*.

```
<aside>  
    <!-- información menos relevante de la página -->  
    <a href="enlace.html"></a>  
    <a href="enlace.html"></a>  
</aside>
```

Contenido del <aside> en el documento HTML.

```
aside img{  
    width: 100%;  
    margin-bottom: 20px;  
}
```

Estilos para el <aside> en el archivo CSS.

Haciendo nuestra página responsive

Para hacer nuestra página *responsive* introduciremos en nuestro CSS las *media queries*, indicando los tamaños de las pantallas de los dispositivos.

En este caso adaptaremos la página para dispositivos móviles con un ancho máximo de 480px, y para tabletas con un ancho máximo de 700px.

```
@media screen and (max-width: 700px) {  
    section {  
        width: auto;  
        float: none;  
    }  
    aside {  
        width: auto;  
        float: none;  
    }  
}  
@media screen and (max-width: 480px) {  
    #contenedor{  
        width: 100%;  
        border:none;  
        margin: 0px;  
        padding: 0px;  
    }  
    header {  
        margin: 0;  
    }  
    header .logo{  
        margin: auto;  
        float: none;  
        width: 100px;  
    }  
    nav{  
        display: none;  
    }  
    header h3 {  
        display: none;  
    }  
    aside {  
        display: none;  
    }  
    section{  
        width: 100%;  
        margin: 0;  
        padding: 5px;  
    }  
}
```

Gracias a la propiedad de cascada podemos reemplazar los estilos, aplicando nuevas propiedades dependiendo del tamaño del dispositivo.

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

Una página web puede tener todo el contenido que queramos poner. En esta unidad hemos realizado paso a paso una pequeña muestra de lo que nos permite la tecnologías HTML y CSS.

Hemos hecho un recorrido desde la creación de la estructura básica de una página hasta la inserción de algo de contenido.

Ahora es importante que practiques realizando tus propias creaciones, combinado todas las herramientas comentadas en esta unidad.

Si quieres conocer más etiquetas HTML o reglas CSS, puedes encontrar cientos de manuales y libros por Internet, aunque debes estar atento que estén actualizados a HTML5.

Para conocer más etiquetas HTML y reglas CSS, puedes consultar W3Schools.

<https://www.w3schools.com/>

**MP_0373. Lenguajes de marcas y sistemas de
gestión de información**

**UF3. Lenguajes de almacenamiento y transmisión
de información**

3.1. Tipos de lenguajes



Índice

Objetivos	3
Introducción.....	4
Lenguajes para la transmisión de información	4
Metalenguajes.....	7
Características	7
Tipos de lenguajes	9
Lenguaje de marcas XML	9
Lenguaje de listas JSON	10
Diferencias entre JSON y XML	11
Despedida	13
Resumen.....	13

Objetivos

Los objetivos de esta unidad son los siguientes:

- Conocer los lenguajes para el almacenamiento y transmisión de información.
- Comprender las reglas de lenguaje de los metalenguajes.
- Analizar el lenguaje de marcas XML y de listas JSON.
- Establecer la necesidad de describir la información transmitida en los documentos.

Introducción

Lenguajes para la transmisión de información

En la actualidad los servicios y aplicaciones comparten información de forma asidua. La gran cantidad de servicios disponibles en redes como Internet da la posibilidad de que las aplicaciones comparten información entre ellas pero, siendo independientes unas de otras, necesitan un formato de representación de información común para realizar esta comunicación.

Si bien **HTML** está concebido para presentar la información, se limita a crear estructuras básicas de textos para facilitar su comprensión por el ser humano, no permite la transmisión de objetos o estructuras de datos que los sistemas informáticos comprendan mejor. Para este fin es donde entran los **lenguajes de marcas o de listas como XML y JSON**.



Fuente: Google Developers.

Pongamos un ejemplo

Una aplicación web como Wordpress (CMS, Content Management System o sistema gestor de contenidos) nos sirve para publicar información en un sitio web. Para ese fin posee herramientas de publicación.

Pero como ya sabemos, la cantidad de información que existe hoy en día en la red dificulta la localización de contenidos, por lo que necesitamos otras aplicaciones que indexen la información de los diferentes sitios web existentes. Los buscadores de Internet como Google o Bing tienen esa función.

Para facilitar esa tarea a los buscadores las aplicaciones web disponen de la posibilidad de generar un mapa del contenido, lo que facilitará al buscador la localización de la información relevante. Pero, **¿cómo conseguir que una aplicación web pueda comunicarse con otra totalmente diferente como el caso de un buscador?** Sencillamente creando un documento que contenga la estructura de la información publicada en un formato estandarizado, que las dos aplicaciones comprendan.

Este documento podríamos realizarlo a mano, o en el caso de nuestro ejemplo disponemos de múltiples *plugins* que nos facilitarán la tarea creando el *site-map* (mapa o directorio) de nuestro sitio web automáticamente.

The screenshot shows the configuration page for the XML Sitemap Generator plugin. At the top, it displays the result of the last ping, which started on June 6, 2014 at 18:16. It lists notifications sent to Google and Bing. On the right, there's a sidebar with links for support topics, a plugin homepage, and various donation and reporting options. The main area contains sections for basic and advanced options, including checkboxes for notifying Google, Bing, and robots.txt, and fields for memory and execution limits. A note about XSLT stylesheets and base URLs is also present. At the bottom, there's a link to the plugin's official site.

Result of the last ping, started on June 6, 2014 18:16.

The URL to your sitemap index file is: <http://www.arnebrachhold.de/sitemap.xml>.
Google was successfully notified about changes.
Bing was successfully notified about changes.
Notify Search Engines about [your sitemap](#) or [your main sitemap](#) and [all sub-sitemaps](#) now.
If you encounter any problems with your sitemap you can use the [debug function](#) to get more information.

About this Plugin:

- Plugin Homepage
- Suggest a Feature
- Help / FAQ
- Notify List
- Support Forum
- Report a Bug
- Donate with PayPal
- My Amazon Wish List

Basic Options

Update notification: [Learn more](#)

Notify Google about updates of your Blog
No registration required, but you can join the [Google Webmaster Tools](#) to check crawling statistics.

Notify Bing (formerly MSN Live Search) about updates of your Blog
No registration required, but you can join the [Bing Webmaster Tools](#) to check crawling statistics.

Add sitemap URL to the virtual robots.txt file.
The virtual robots.txt generated by WordPress is used. A real robots.txt file must NOT exist in the blog directory!

Advanced options: [Learn more](#)

Try to increase the memory limit to: (e.g. "4M", "16M")

Try to increase the execution time limit to: (in seconds, e.g. "60" or "0" for unlimited)

Include a XSLT stylesheet: (Full or relative URL to your .xsl file) Use default

Override the base URL of the sitemap:
Use this if your blog is in a sub-directory, but you want the sitemap be located in the root. Requires .htaccess modification. [Learn more](#)

Include sitemap in HTML format

Allow anonymous statistics (no personal information) [Learn more](#)

Sitemap Resources:

- Webmaster Tools
- Webmaster Blog
- Search Blog
- Webmaster Tools
- Webmaster Center Blog
- Sitemaps Protocol
- Official Sitemaps FAQ
- My Sitemaps FAQ

Recent Donations:

Plugin Google XML Sitemap. <https://es.wordpress.org/plugins/google-sitemap-generator/>

XML Sitemap

This is a XML Sitemap which is supposed to be processed by search engines which follow the XML Sitemap standard like Ask.com, Bing, Google and Yahoo. It was generated using the Blogging-Software [WordPress](#) and the [Google Sitemap Generator Plugin](#) by [Arne Brachhold](#). You can find more information about XML sitemaps on [sitemaps.org](#) and Google's [list of sitemap programs](#).

URL	Priority	Change frequency	Last modified (GMT)
http://localhost/wp_svn/2014/02/elit-elit-conque-diam-condimentum-vulputate-sociis/	20%	Monthly	2014-02-28 21:34
http://localhost/wp_svn/2014/02/non-odio-quisque-ac-tortor-maecenas-eros-ut-potenti-integer-mass/	20%	Monthly	2014-02-28 21:23
http://localhost/wp_svn/2014/02/libero-erat-nunc-ullamcorper-ornare-nec-quisque-eu-nisl-dis-iaculis-viverra-tristique/	20%	Monthly	2014-02-28 19:31
http://localhost/wp_svn/2014/02/porta-suspendisse-suscipit/	20%	Monthly	2014-02-28 13:05
http://localhost/wp_svn/2014/02/ante-ac-vestibulum-scelerisque-malesuada-duis-aueo-convallis-habitant-aliquam-duis-ut-s/	20%	Monthly	2014-02-28 12:55
http://localhost/wp_svn/2014/02/feugiat-cursu/	20%	Monthly	2014-02-28 09:46
http://localhost/wp_svn/2014/02/felis-neque-lacus-ut-sollicitudin-aliquam-sit-accumsan-metus-sociis-tortor-lacu/	20%	Monthly	2014-02-28 08:44
nullam-blandit-aliquam-ellit-ellit-quam-ipsum-imperdiet-mal/">http://localhost/wp_svn/2014/02/nullam-blandit-aliquam-ellit-ellit-quam-ipsum-imperdiet-mal/	20%	Monthly	2014-02-28 07:33
http://localhost/wp_svn/2014/02/sit-ultricies-consequat-ac-praesent-vivamus-nisl-tortor-imperdiet-iusto-euismod-vestibulum-eli/	20%	Monthly	2014-02-28 03:14
http://localhost/wp_svn/2014/02/vitae-gravida-placerat-leo-nulla-id-fringilla-at-sapien-rutrum-faucibus-ultrices-mauris-sit-vitae/	20%	Monthly	2014-02-28 02:22
http://localhost/wp_svn/2014/02/lacus-amet-interdum-amet-rutrum-volutpat-nisl-quam-iaculis-ut-l/	20%	Monthly	2014-02-28 02:15
http://localhost/wp_svn/2014/02/tristique-diam-in-sc/	20%	Monthly	2014-02-27 20:48
http://localhost/wp_svn/2014/02/sit-tortor-interdum-fringilla-dolor-consequat-nunc-lobortis-suspendi/	20%	Monthly	2014-02-27 17:48
http://localhost/wp_svn/2014/02/amet-turpis-sed-fermentum-vivamus-aliquam-ultrices-a-ac-facilisis-ut-pelleentesque-nunc-felis-auctor/	20%	Monthly	2014-04-23 19:04
http://localhost/wp_svn/2014/02/nisl-vestibulum-cursus-ma/	20%	Monthly	2014-02-27 11:24
http://localhost/wp_svn/2014/02/ultrices-felis-accumsan-aliquam-felis-sociis/	20%	Monthly	2014-02-27 10:20
http://localhost/wp_svn/2014/02/adipiscing-id-liqua-nulla-et-dolor-nisi-vivamus-libero-urna-ac/	20%	Monthly	2014-02-27 10:10

Sitemap como hoja de estilos para facilitar su visualización

```
<!--
sitemap-generator-url="http://www.arnebrachhold.de" sitemap-generator-version="4.0.7"
-->
<!-- generated-on="June 19, 2014 4:51 pm" -->
<?xml version="1.0" encoding="UTF-8"?>
<sitemapindex xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.sitemaps.org/schemas/sitemap/0.9" xsi:schemaLocation="http://www.sitemaps.org/schemas/sitemap/0.9 http://www.sitemaps.org/schemas/sitemap/0.9.xsd">
    <sitemap>
        <loc>http://localhost/wp_svn/sitemap-misc.xml</loc>
        <lastmod>2014-06-13T21:30:33+00:00</lastmod>
    </sitemap>
    <sitemap>
        <loc>http://localhost/wp_svn/sitemap-tax-post_tag.xml</loc>
        <lastmod>2014-06-13T21:30:33+00:00</lastmod>
    </sitemap>
    <sitemap>
        <loc>http://localhost/wp_svn/sitemap-tax-category.xml</loc>
        <lastmod>2014-06-13T21:30:33+00:00</lastmod>
    </sitemap>
    <sitemap>
        <loc>http://localhost/wp_svn/sitemap-externals.xml</loc>
        <lastmod>2014-06-13T21:30:33+00:00</lastmod>
    </sitemap>
    <sitemap>
        <loc>http://localhost/wp_svn/sitemap-pt-post-2014-06.xml</loc>
        <lastmod>2014-06-13T21:30:33+00:00</lastmod>
    </sitemap>
    <sitemap>
        <loc>http://localhost/wp_svn/sitemap-pt-post-2014-05.xml</loc>
        <lastmod>2014-05-17T20:30:20+00:00</lastmod>
    </sitemap>
</sitemapindex>
```

El XML con el contenido sin hoja de estilos

Metalenguajes

El metalenguaje de marcas extensible (eXtensible Markup Language) es sencillamente un conjunto de reglas.

Dicho conjunto de reglas identifica cómo definir etiquetas (marcado de texto) para dividir un documento en partes y subpartes individuales.

El ser humano ha creado **soluciones** para sus problemas desde que empezó a tener problemas de comunicación a larga distancia, como el caso de **T.A. Edison** o **Graham Bell** con el telégrafo y el teléfono (hoy en día atribuido a Antonio Meucci).

En la actualidad existen diferentes tecnologías que ayudan a compartir y presentar datos. Una de ellas es el metalenguaje XML (lenguaje de marcas), que utiliza la fuerza de sus cimientos de SGML (Standard Generalized Markup Language) para cubrir una serie de requisitos a la hora de identificar las piezas de un documento y poder compartirlo parcialmente o en su totalidad con cualquier software / usuario / lugar, tanto si el usuario está conectado a Internet, como si está trabajando con varios formatos de archivo de base de datos durante el desarrollo de un proyecto.

También ha ampliado mucho sus características respecto al uso en los servicios web o para el intercambio de datos en la capa de vista (XHTML).

Características

Su principal característica consiste en que puede especificar **reglas para etiquetas** definidas por él mismo, es decir, puede definir su propio lenguaje de marcado.

XML no es un lenguaje como tal, ya que no posee estructuras de control. Es por esto por lo que se le denomina metalenguaje.

XML es un metalenguaje, igual que HTML, pero se ha definido de forma que no está limitado a un vocabulario, industria o utilización en particular, siendo la personificación de la extensibilidad para Internet.

Se puede adaptar a todos los propósitos corporativos o privados a los que se quiera aplicar. Solo hay que respetar sus reglas.

XML es un metalenguaje muy sencillo de utilizar, sencillo de leer por humanos y máquinas, sencillo de crear y depurar, siendo adecuado para cualquier aplicación que utilice datos estructurados, como hojas de cálculo y bases de datos.

Otras características a destacar de XML son:

- Creación de sus propias etiquetas y atributos.
- Creación de reglas DTD (Definición de Tipo de Documento), que define las normas y estructura de la información del XML.
- La estructura y el diseño de la información son independientes.
- XML se almacena en formato de texto plano.
- Inclusión de metadatos, lo que ayuda al indexado de los documentos.
- Posibilidad de exportación a otros formatos (HTML, PDF, texto enriquecido RTF, voz, etc).
- Es de código abierto (www.w3c.org).
- Puede trabajar con cualquier conjunto de caracteres.
- Comparte información entre sistemas, por ejemplo, páginas web, bases de datos, etc.

XML crea archivos de texto para almacenar información estructurada, permitiendo una utilización más directa y mayor libertad a la hora de creación que SGML.

Tipos de lenguajes

Lenguaje de marcas XML

XML (*eXtensible Markup Language*) es un lenguaje de etiquetas; esto quiere decir que cada paquete de información está delimitado por dos etiquetas, como se hace también en el lenguaje HTML, pero XML separa el contenido de la presentación.

<*nombre*>*Julia*</*nombre*>

La etiqueta <*nombre*> identifica la palabra *Julia* como un nombre de persona, XML se centra en el significado del texto que encierra su etiqueta y no en la apariencia de cómo se muestre el texto en la web.

Una **etiqueta o tag** es una marca utilizada por el lenguaje para **delimitar a un elemento**. Los lenguajes de marcado usan su juego específico de marcas.

En lenguajes de marcado como HTML existen etiquetas o marcas que usan el carácter "<", el "/" o el carácter ">".

En HTML y en XML muchas de las marcas indican el comienzo de una instrucción, que termina con otra marca parecida que señala el fin del contenido de esa instrucción.

Las distintas utilidades que se plantean en la creación de software usando XML son:

- Muchos de los lenguajes de programación usan ficheros XML para intercambiar información. El uso de XML plantea un lenguaje estándar para el intercambio de información entre diferentes programas de manera fiable.
- Los lenguajes de programación usan ficheros XML para dar información adicional sobre la configuración del programa. Dicha información es usada por las API y los *frameworks* para instanciar objetos, crear reglas de navegación, definir procesos y establecer determinadas reglas en las actividades de dichas API y *frameworks*.

Ejemplo

Julia nació el **18.06.91** en **Navarra** con **DNI 02654312B**. Analizando este texto, tenemos datos entendibles por personas que necesitamos que un programa sea capaz de usar.

El ejemplo en formato en XML con sus etiquetas correspondientes sería:

```
<Datos>
<Persona>
<Nombre>Julia</Nombre>
<Fecha>18.06.91</Fecha>
<Ciudad>Navarra</Ciudad>
<DNI>02654312B</DNI>
</Persona>
</Datos>
```

Ejemplo de un XML.

Lenguaje de listas JSON

JSON es un **formato de datos basado en la estructura de objetos de JavaScript**.

Creado por Douglas Crockford, es muy parecido a la sintaxis de objetos usada por JavaScript, pero puede usarse independientemente para la transmisión o el almacenamiento de datos.

Este modelo de datos se está haciendo cada vez más popular porque hace énfasis en su contenido, reduciendo mucho la sintaxis para estructurar la información.

Los **archivos JSON son cadenas de texto que se deben "parsear"** (analizar sintácticamente) cuando se quiere acceder a los datos del documento. Esto no es un problema, pues todos los lenguajes actuales poseen métodos para realizar este *parseo*, como veremos en ejemplos en módulos posteriores.

Convertir una cadena de texto a un objeto se denomina *parsing*, mientras que convertir un objeto a una cadena se denomina *stringification*.

Ejemplo

En el siguiente ejemplo podemos ver la forma en la que se estructura la información con JSON.

Los elementos que definen su estructura son:

- {} - Identifica un objeto.
- [] - Array.
- : - Separador de *clave:valor* (se puede considerar como asignación =).
- "" - Dato o valor.
- , - Separador.

```
{  
    "squadName": "Super hero squad",  
    "homeTown": "Metro City",  
    "formed": 2016,  
    "secretBase": "Super tower",  
    "active": true,  
    "members": [  
        {  
            "name": "Molecule Man",  
            "age": 29,  
            "secretIdentity": "Dan Jukes",  
            "powers": [  
                "Radiation resistance",  
                "Turning tiny",  
                "Radiation blast"  
            ]  
        },  
        {  
            "name": "Madame Uppercut",  
            "age": 39,  
            "secretIdentity": "Jane Wilson",  
            "powers": [  
                "Million tonne punch",  
                "Damage resistance",  
                "Superhuman reflexes"  
            ]  
        },  
        {  
            "name": "Eternal Flame",  
            "age": 1000000,  
            "secretIdentity": "Unknown",  
            "powers": [  
                "Immortality",  
                "Heat Immunity",  
                "Inferno",  
                "Teleportation",  
                "Interdimensional travel"  
            ]  
        }  
    ]  
}
```

Ejemplo de código en JSON.

Diferencias entre JSON y XML

JSON y XML son tecnologías muy implantadas entre los sistemas informáticos y son reconocibles por la mayoría de los lenguajes de programación actuales.

Ambos lenguajes se pueden usar para la transmisión de información.

JSON es más liviano y rápido al poder estructurar los datos con menor número de caracteres. **XML es anterior y está muy implantado** en el mundo web, dada la facilidad que tiene para trabajar con tecnologías como AJAX (Asynchronous JavaScript And XML, técnica de desarrollo web para crear aplicaciones interactivas).

Similitudes

- JSON y XML son métodos de transmisión de información entre sistemas informáticos.
- Ambos son reconocibles por múltiples lenguajes de programación.
- Ambos admiten cambios de plataforma.
- Los dos son archivos de texto plano.

Diferencias

JSON

- Basado en *JavaScript Object Notation*.
- JSON está **orientado a objetos**.
- Admite definición de tipos de datos.
- Depende del lenguaje de programación para su validación.
- Es **más fácil de leer**.
- **Necesita ser parseado** por los lenguajes de programación.

XML

- **Lenguaje de marcado extensible**.
- Se basa en etiquetas.
- **No diferencia el tipo de datos**.
- Definición de reglas en el propio formato (DTD).
- **Necesita más caracteres** para estructurar los datos.
- **Mayor integración con AJAX**.

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

En esta unidad hemos aprendido la importancia que tiene la comunicación entre nuestras aplicaciones y el formato de documentos estandarizados para esa comunicación mediante XML y JSON.

Se ha realizado un pequeño análisis de las diferencias entre JSON y XML, siendo los dos sistemas de comunicación más usados en los lenguajes de programación actuales.

No existe un criterio concreto para la selección de cuál de los dos usar; siendo ambos lenguajes muy extendidos, tendremos que valorar los servicios utilizados para realizar dicha comunicación en cada caso.

**MP_0373. Lenguajes de marcas y sistemas de gestión
de información**

**UF3. Lenguajes de almacenamiento y transmisión de
información**

3.2. XML



Índice

Objetivos	3
XML: estructura y sintaxis	4
Partes de un documento XML	4
Jerarquía de elementos	6
Documentos bien formados	7
Herramientas	9
Modelo de documento DTD	10
DTD	10
Sintaxis	11
Elementos	12
Atributos	13
Entidades	15
Entidades internas	15
Entidades externas y parámetros	15
Notación	16
Modelo de documento XML schema	17
XML Schema	17
Definición	18
Elementos simples	18
Atributos y restricciones	19
Despedida	20
Resumen	20

Objetivos

Los objetivos de esta unidad son los siguientes:

- Identificar las tecnologías relacionadas con la definición de documentos XML.
- Analizar la estructura y sintaxis específica utilizada en la descripción.
- Crear descripciones de documentos XML.
- Utilizar descripciones en la elaboración y validación de documentos XML.
- Conocer el esquema XML (XSD, *XML Schema Definition*).

XML: estructura y sintaxis

Partes de un documento XML

Recuerda que XML no es un lenguaje de etiquetas, sino un conjunto de reglas para definir lenguajes de etiquetas, como XHTML.

Las etiquetas en XML se definen entre paréntesis angulares “< ... >”. Los documentos XML se componen de etiquetas y de contenido.

- El contenido está formado por los datos o textos sobre los que se aplican las marcas.
- Las etiquetas son la forma fundamental de los documentos de marcado.

Existen seis tipos de etiquetas:

Elementos y atributos

Son las partes en las que se divide el documento. Crean una estructura jerárquica para indicar qué elementos contienen otros elementos.

Se suele utilizar el parentesco para identificarlo (hijo de..., padre de..., hermano de...).

Los atributos son características de una etiqueta, pertenecen a la propia etiqueta que lo contiene y no son heredables.

```
<biblioteca>
  <libro>
    <titulo>La vida está en otra parte</titulo>
    <autor>Milan Kundera</autor>
    <fechaPublicacion año="1973"/>
  </libro>
  <libro>
    <titulo>Pantaleón y las visitadoras</titulo>
    <autor fechaNacimiento="28/03/1936">Mario Vargas Llosa</autor>
    <fechaPublicacion año="1973"/>
  </libro>
  <libro>
    <titulo>Conversación en la catedral</titulo>
    <autor fechaNacimiento="28/03/1936">Mario Vargas Llosa</autor>
    <fechaPublicacion año="1969"/>
  </libro>
</biblioteca>
```

Anidamiento de <libro>

Atributo

Referencia de entidad

Las referencias de entidad se utilizan para escribir en el contenido del documento caracteres que son difíciles de escribir o no están en el teclado. Para ello se utilizan 3 formas:

- Por el nombre de entidad. Siempre debe empezar con & y terminar con el punto y coma. Por ejemplo, el símbolo de libra se puede poner como £. Al procesar el documento que contenga esa expresión aparecerá el nombre de libra. La forma de escribirlo sería:
&nombre;
- Por el nombre decimal. Funciona de forma similar al anterior, pero en lugar de indicar el nombre del carácter se introduce su codificación en ASCII. La forma de escribirlo sería:
&#número;
- Por el valor hexadecimal. Funciona de forma similar, pero aquí se indica el número en formato hexadecimal.
&#xnúmero;

Comentarios

Los comentarios son elementos que no son ni contenido ni ningún elemento de marcado. Permiten hacer anotaciones para que sea más legible y entendible el documento XML sin procesar.

Para indicar un elemento se comienza con “<!--” y se terminan con “-->”. Un comentario no puede estar dentro de una etiqueta <...>.

<!-- Esto es un ejemplo y no tendrá repercusión cuando se procese el documento. -->

Instrucciones de procesamiento

Las instrucciones de procesamiento son etiquetas que proporcionan información, datos o instrucciones a una aplicación. Cuando se utiliza el documento estas instrucciones serán interpretadas por la aplicación que procesa el XML. Al igual que los comentarios, no forman parte de la presentación de XML.

Para escribir una instrucción de procesamiento se utiliza la siguiente forma:

<?Nombre instrucción ?>

El *nombre* identifica la aplicación que debe interpretar la instrucción. Los nombres que comienzan con XML se reservan para las instrucciones de normalización de XML.

CDATA

Las secciones **CDATA** identifican una sección en la que no se interpretarán los caracteres. Se utiliza para poder mostrar caracteres en el documento procesado que son expresiones o palabras reservadas en XML.

El formato para escribir una sección CDATA es:

<![CDATA[*caracteres reservados*]]>

Declaraciones de tipo de documento

Las declaraciones de tipo de documento (DTD) sirven para la descripción de la estructura y la sintaxis del documento. Aquí se definirán las etiquetas que se van a usar en la estructura del documento, qué elementos pueden incluir otros elementos, etc.

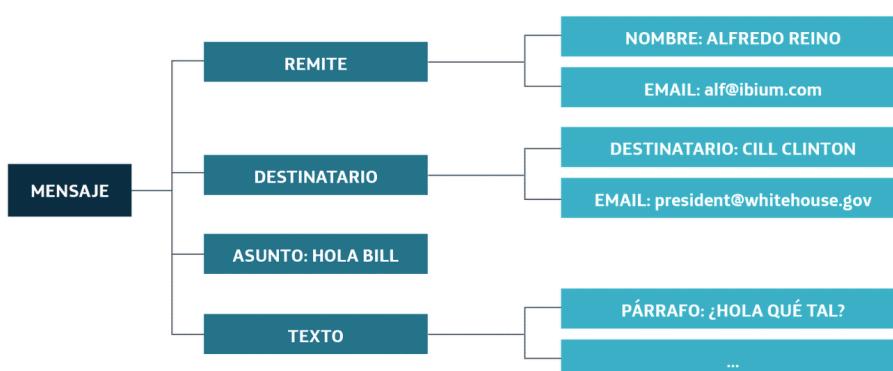
Una instrucción del DTD siempre tiene la forma <!Valor Instrucción>, donde:

- *Valor* es una palabra clave que permite identificar el elemento que se va describir.
- *Instrucción* es el código para el valor.

Más adelante veremos cómo escribir las instrucciones en la DTD del documento.

Jerarquía de elementos

El siguiente XML muestra la estructura jerárquica de elementos, representada también en la imagen.



```
<?xml version="1.0" encoding="UTF-7"?>
<!DOCTYPE mensaje SYSTEM "mensaje.dtd">
<mensaje>
    <remite>
        <nomb...>Alfredo Reino</nomb...>
```

```
<email>alf@ibium.com</email>
</remite>
<destinatario>
  <nombre>Bill Clinton</nombre>
  <email>president@whitehouse.gov</email>
</destinatario>
<asunto>Hola Bill</asunto>
<texto>
  <parrafo>¿Hola qué tal? ...</parrafo>
</texto>
</mensaje>
```

Documentos bien formados

Un documento bien formado es aquel que cumple estrictamente con las siguientes normas para los elementos.

- Los **elementos deben seguir una estructura jerárquica**. Cada elemento debe estar contenido dentro de otro, exceptuando el elemento raíz.
- Los **elementos deben estar alineados**; deben contener tabulaciones de forma que en la columna que tenemos la etiqueta de inicio de un elemento, debe estar también la de cierre. Esto hace legible los documentos sin procesar al ojo humano.
- Los **elementos no pueden estar superpuestos**. Un elemento que está contenido en otro elemento debe estar dentro de este elemento su etiqueta de inicio y final.
- Los **elementos deben estar cerrados**, es decir, todos los elementos deben tener su etiqueta de inicio y de cierre.
- Solo puede existir **un elemento raíz**, del que parten los demás.
- El **nombre de los elementos debe empezar siempre con un carácter no numérico** y nunca puede comenzar por XML o cualquier variación, es decir, no puede empezar por xml, Xml, XMI, xML, etc.
- Hay que **respetar las mayúsculas y minúsculas**. XML es un lenguaje *case sensitive* (sensible a mayúsculas y minúsculas). No es lo mismo la etiqueta *<Etiqueta>* que la etiqueta *<eTiqueta>*.
- Los **espacios y retornos de carro** solamente son tenidos en cuenta si aparecen en el valor de un atributo.
- **No se pueden utilizar de "manera normal" los caracteres** y palabras claves de XML, que son: &, <>, "".

Documento válido

La especificación XML en W3C establece que cualquier programa que procese un XML debe dejar de hacerlo si encuentra un error. Sin embargo, un documento HTML seguirá siendo procesado, aún con errores.

Un documento XML es válido si está bien formado y además cumple con las definiciones que se han mostrado en la DTD o en cualquier otro elemento de validación.

Para poder validar un documento XML podemos utilizar diferentes métodos:

- **DTD:** lo que realiza es una comprobación del cumplimiento del DTD definido.
- **Schema:** es un lenguaje más avanzado que la DTD y permite especificar la estructura y contenidos del documento.
- **Relax NG:** es un lenguaje de esquema basado en la gramática.
- **Schematron:** utiliza expresiones de acceso.

Pueden ser descritos al principio del documento o a través de un documento externo, cuyos enlaces se describen en el prólogo del XML. Ejemplo:

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

Validador XML

Editor web para validar XML. https://www.w3schools.com/xml/xml_validator.asp

Validador HTML

Editor web para validar HTML. <http://validator.w3.org/>

Ejemplo de documentos mal formados

```
<?xml: version="1.0"?>
<base_datos nombre="MiEmpresa", tipo="Oracle", propietario="user1">
    <tabla nombre="Sucursales", propietario="user1">
        <columna nombre="nSucursal", tipo="Integer">
        </columna>
    </tabla >
</base_datos>
<autor nombre="jdedef">
</autor>
```

Este documento está mal formado porque existen dos elementos raíz y no respeta las tabulaciones.

```
<?xml: version="1.0"?>
<base_datos nombre="MiEmpresa", tipo="Oracle", propietario="user1">
    <tabla nombre="Sucursales", propietario="user1">
        <columna nombre="nSucursal", tipo="Integer">
    </tabla>
</base_Datos>
    </coluMNa>
```

Es un documento mal formado porque no respeta el anidamiento ni las mayúsculas y minúsculas.

Herramientas

Para poder confeccionar un documento XML y HTML de manera correcta se pueden usar herramientas que revisan la sintaxis mientras se confecciona el documento.

Entre estas herramientas existen varios editores de XML que facilitan la tarea de crear documentos válidos y bien formados, pues advierten de los fallos de sintaxis de manera automática.

Las **funciones mínimas** que debería tener un editor de XML son:

- Ser capaz de leer la DTD del documento y presentar una lista *drop-down* con todos los elementos disponibles en la propia DTD de forma enumerada; de esa forma se evita incluir elementos no definidos.
- Avisar del olvido de etiquetas que son de uso obligatorio, incluso no permitiendo que se pueda continuar con la edición y salvaguarda del documento.

Algunos editores:

- **XML Pro** de Vervet Logic (open source).
- **XMLSpy** de Altova.
- **Liquid XML Studio** de Liquid Technologies.
- **<oXigen/>** XML Editor.
- **Turbo XML** de TIBCO (Plataforma de desarrollo integrado de XML).
- **XML Notepad** de Microsoft.
- **XMLwriter** de Wattle Software.

Modelo de documento DTD

DTD

Al igual que en SGML, en XML los DTD son **archivos de texto que encierran una definición formal** de un tipo de documento y, a la vez, tienen la función de **especificar la estructura lógica de un archivo XML**, proporcionando los nombres de los elementos, atributos y entidades que utiliza, además de **suministrar la información sobre cómo se pueden usar conjuntamente**.

Un DTD es un **conjunto de reglas sintácticas** para definir etiquetas.

Nos indica qué etiquetas se pueden usar en un documento, en qué orden deben aparecer, cuáles pueden aparecer dentro de otras, cuáles tienen atributos, etc.

Crear una definición del tipo de documento (DTD) es como crear nuestro propio lenguaje de marcado para una aplicación específica. Por ejemplo, podríamos crear un DTD que defina una tarjeta de visita. A partir de ese DTD, tendríamos una serie de elementos XML que nos permitirían definir tarjetas de visita.

Este lenguaje formal **permite a los procesadores analizar automáticamente un documento** e identificar de dónde viene cada elemento y cómo se relacionan entre ellos, para que los navegadores, las hojas de estilo, los motores de búsqueda, las bases de datos o las aplicaciones de impresión, entre otras, puedan utilizarlos.

El DTD del lenguaje XML es opcional. En tareas sencillas no es necesario construir una DTD, entonces se trataría de un documento "bien formado" (*well-formed*) y si lleva DTD será un documento "validado" (*valid*).

DTD especifica la clase de documento

- Describe un formato de datos.
- Usa un formato común de datos entre aplicaciones.
- Verifica los datos al intercambiarlos.
- Verifica un mismo conjunto de datos.

DTD describe

- **Elementos:** cuáles son las etiquetas permitidas y cuál es el contenido de cada etiqueta.
- **Estructura:** en qué orden van las etiquetas en el documento.
- **Anidamiento:** qué etiquetas van dentro de cuáles.

Sintaxis

El documento XML suele comenzar por una instrucción de procesamiento para el procesador de XML. En esta instrucción se indica el tipo de normalización que sigue el documento según el estándar XML y la codificación.

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

Después suele comenzar el DTD, que puede ser referenciado por un enlace o descrito manualmente en el propio documento. Para definir el DTD utilizamos la sintaxis:

```
<!DOCTYPE[  
    definiciones  
]>
```

Donde en *definiciones* se incluyen todas las declaraciones que puede contener el DTD. Si se referencia a un estándar externo se utiliza la palabra SYSTEM de la siguiente forma:

```
<!DOCTYPE etiqueta SYSTEM "documento.dtd">
```

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE ligaDeFutbol [  
    <!ELEMENT ligaDeFutbol (partido*)>  
    <!ELEMENT partido EMPTY>  
    <!ATTLIST partido localNombre CDATA #REQUIRED>  
    <!ATTLIST partido localGoles CDATA #REQUIRED>  
    <!ATTLIST partido visitanteNombre CDATA #REQUIRED>  
    <!ATTLIST partido visitanteGoles CDATA #REQUIRED>  
>  
  
<ligaDeFutbol>  
    <partido localNombre="Nottingham Presa" localGoles="0" visitanteNombre="Inter  
    de Mitente" visitanteGoles="1" />  
    <partido localNombre="Vodka Juniors" localGoles="3" visitanteNombre="Sparta  
    da Risa" visitanteGoles="3" />  
    <partido localNombre="Water de Munich" localGoles="4" visitanteNombre="Esteaua  
    es del grifo" visitanteGoles="2" />  
</ligaDeFutbol>
```

Ejemplo de XML con DTD.

Elementos

Define todos los elementos que podemos encontrar a lo largo del documento, es decir define las etiquetas del documento.

<!ELEMENT tag (tag1, tag2)>

Si queremos describir un elemento que contenga otro elemento pondremos entre paréntesis los elementos que se pueden introducir.

<!ELEMENT tag (tag1 | tag2, tag3 +, tag4 *)>

- Si está permitido que aparezca más de una vez pondremos el signo "+" detrás del nombre.
- Si un elemento es opcional pondremos "?".
- Si un elemento es opcional y puede repetirse pondremos "*".
- Con "|" indicamos que el elemento es opcional, pero tendremos que elegir entre uno de ellos.

Para los elementos que no contienen a otros elementos indicaremos si están vacíos (Empty), cualquier contenido (Any) o solo datos (#PCDATA).

Ejemplos:

```
<!ELEMENT texto #PCDATA>
<!ELEMENT salto EMPTY>
<!ELEMENT extra ANY>
```

<!ELEMENT elemento (#PCDATA, elemento1)>

También se permite el uso de elementos mixtos.

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

En el ejemplo anterior podemos ver que el DTD define las etiquetas que estarán dentro de la etiqueta <note>, no permitiendo el uso de las no indicadas (to, from, heading, body).

También definimos que cada una de las etiquetas contendrá datos (#PCDATA).

De este modo, si falta alguno de los elementos indicados o el contenido no es el indicado, el XML no se validará.

Saber más

En este enlace encontrarás más ejemplos de aplicación de ELEMENT.

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

Atributos

Los atributos permiten añadir información adicional a los elementos de un documento. La principal diferencia entre los elementos y los atributos es que los atributos no pueden ser anidados (no tienen hijos).

Se usan para añadir información corta, sencilla y desestructurada a una etiqueta.

```
<!ATTLIST Elemento NomAtributo Tipo Valpredeterminado>
```

Con los atributos podemos especificar cuál será el contenido adicional del elemento. Para definir el tipo del atributo que contendrá el elemento tenemos:

CDATA

Los atributos CDATA (*character data*) son los más sencillos, y pueden contener casi cualquier cosa. !CDATA hace que los analizadores lo ignoren.

ID

Nombre que identifica inequívocamente a la etiqueta.

IDREF

Sirve para referenciar una etiqueta que se ha definido en el ID.

ENTITY

(entities) Es el valor de una entidad o entidades separadas por coma.

NMTOKEN

(nmtokens) Se parecen a CDATA, pero solo aceptan denominar cosas con caracteres validos (letras, números, puntos, guiones, subrayados y los dos puntos).

Lista de nombres

Si se especifica una lista de nombres separados por comas indica que al usar el atributo se debe seleccionar uno de esa lista.

También podemos definir el tipo de valores predeterminado para los elementos:

#REQUIRED

Indica que el atributo es requerido siempre que se use la etiqueta.

#VALOR

Indica un valor que se usará por defecto si no se especifica el valor del atributo.

#IMPLIED

Indica que no se quiere que el atributo coja el valor por defecto.

#FIXED

Indica que el atributo siempre tiene el valor, aunque no se especifique.

```
<!ATTLIST curso
director CDATA #REQUIRED
horario (mañana | tarde | noche)
#IMPLIED
instalaciones (Exes | externas)
"Exes">
```

Ejemplo de aplicación ATTLIST.

Saber más

En este enlace encontrarás más ejemplos de ATTLIST.

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

Entidades

Las entidades permiten colocar trozos de contenido en cualquier lugar del documento haciendo referencia a su nombre. Para definir una entidad usamos la etiqueta `<!ENTITY`.

Existen diferentes entidades:

- Internas.
- Externas y parámetros.

Entidades internas

También llamadas **macros o constantes de texto**. Las entidades internas son las que se asocian a una cadena de caracteres. Se referencian únicamente desde el propio fichero, de ahí su nombre, internas.

```
<!ENTITY tag "Jaun Perez">
```

Son abreviaturas que se usan dentro del documento para simplificar. Cuando se procese el documento estas abreviaturas serán sustituidas por el nombre completo. Para definirlas se usa:

```
<!ENTITY tag "texto sustituyo">
```

Para referenciar la entidad dentro del documento usamos "**&nombre_entidad**", donde cada vez que usemos **&ISO** en el documento aparecerá el nombre completo.

```
<!ENTITY ISO "International Organization for Standardization">
```

Entidades externas y parámetros

Las **entidades externas** son similares a las internas, con la diferencia de que obtienen el contenido fuera del documento, pudiendo ser incluso imágenes o archivos de sonido. Se utiliza la palabra SYSTEM para indicar que es externo:

```
<!ENTITY entity-name SYSTEM "URI/URL">
```

Los **parámetros** solo se pueden referenciar en la DTD del documento. Se utiliza % antes del nombre para referenciar y se definen de la forma:

```
<!ENTITY %parametro "texto">
```

Saber más

En este enlace puedes ampliar la información sobre *<ENTITY>*.

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

Notación

Este tipo de atributo permite al autor declarar que su valor se ajusta a una notación declarada.

```
<!NOTATION nombre SYSTEM "notación">

<!ATTLIST mensaje fecha NOTATION (ISO-DATE / EUROPEAN-DATE) #REQUIRED>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE frutas [
    <!ELEMENT frutas (fruta)*>
    <!ELEMENT fruta EMPTY>
    <!ATTLIST fruta foto ENTITY #REQUIRED>

    <!ENTITY manzana SYSTEM "manzana.gif" NDATA gif>
    <!ENTITY naranja SYSTEM "naranja.gif" NDATA gif>

    <!NOTATION gif SYSTEM "image/gif">
]>

<frutas>
    <fruta foto="manzana"/>
    <fruta foto="naranja"/>
</frutas>
```

Ejemplo XML con DTD en el propio documento.

Modelo de documento XML schema

XML Schema

Un esquema, al contrario que un DTD, puede definir tipos de datos, lo cual es claramente beneficioso en el intercambio de datos, objetos o bases de datos.

Schema es más potente que el DTD clásico de XML, ya que permite más flexibilidad. Es un estándar recomendado por el W3C (Word Wide Web Consortium).

El DTD o declaración del tipo de documento y los esquemas proporcionan la **gramática** para una clase de documentos XML.

Estos mecanismos se utilizan para la llamada validación, tanto estructural como formal del documento, esto es, enviar un documento a un destinatario junto con las condiciones que deben cumplir los documentos.

En la actualidad los esquemas se utilizan en mayor medida que los DTD dentro de XML.

Un problema que presentan los DTD es que no siguen una **sintaxis XML**, sino una propia. Un grupo de desarrolladores ha propuesto una alternativa a los DTD, llamada esquemas ("schemas").

Básicamente, un esquema establece las reglas de un documento e indica qué etiquetas se pueden usar, sus atributos, las relaciones entre etiquetas, etc.

Ejemplo

En el siguiente ejemplo podemos ver que mediante el *Schema*, no solo definimos los elementos que contendrá nuestro documento, sino que también estamos definiendo el tipo de datos.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
```

```
<xs:element name="body" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Ejemplo de *XML Schema Definition* (XSD).

Definición

Al definir el *schema* es necesario escribir una serie de atributos:

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

Esto indica que los elementos utilizados corresponden al espacio de nombres (**namespace**) definido en esa dirección. No es el objetivo de este módulo describir los espacios de nombres, pero podemos decir que para que no haya ambigüedad entre las etiquetas descritas, siempre se define un espacio de nombres al que pertenecen.

Dentro de un espacio de nombres no puede haber dos etiquetas con el mismo nombre. Al igual que en la DTD, es posible hacer una referencia a un esquema o escribirlo manualmente en el prólogo del documento.

Para referenciarlo utilizamos los atributos **xsi** de **xmlns:xsi** y **xsi:schemalocation**. En este último pondríamos el archivo o dirección web donde esté definido el nuevo *schema*. Puedes ver un ejemplo a continuación.

```
xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"c:/archivo.xsd"
```

Elementos simples

Un elemento simple es el que contiene solo datos de contenido, no contiene ningún otro elemento. Se describe con la etiqueta:

```
<xs: element name="nombre_elemento" type="tipo_datos"/>
```

En *schema* cuando se define un elemento se debe poner tanto el nombre como el tipo de datos que contiene.

Se han definido varios tipos de datos, incluso pueden definirse otros tipos de datos nuevos descritos por el usuario.

Los más comunes son:

- xs:string
- xs:integer
- xs:boolean
- xs:date
- xs:time

El **elemento** puede incluir atributos o restricciones. Si no incluyera ninguna de estas utilizaríamos la forma de sintaxis abreviada que hemos visto; abre con "<" y cierra al final de la línea con "/>."

Si tuviera alguno de estos elementos se debe definir una nueva etiqueta, indicando que el elemento es simple:

```
<xs:element name="nombre_elemento">
    <xs:simpleType>
```

Atributos y restricciones

Para definir atributos se utiliza una sintaxis similar a la que tenemos para los elementos.

```
<xs:attribute name="nombre_atributo" type="tipo"/>
```

Se pueden poner valores por defecto y fijos con los atributos *fixed* y *default*, así como elegir si es requerido o no con el atributo *use*.

```
<xs:attribute name="nombre_atributo" type="tipo" use="required"
    default="valor"/>
<xs:attribute name="nombre_atributo" type="tipo" fixed="valor"/>
```

En cuanto a las restricciones, existen gran número de restricciones que se indican con una etiqueta. Las más utilizadas son la de valor máximo y mínimo, la longitud del campo de datos, o la elección entre una lista de valores de atributo:

```
<xs:restriction base="xs:string">
    <xs:length value=" numero de caracteres"/>
</xs:restriction>
<xs:restriction base="xs:string">
    <xs:length value="8"/>
</xs:restriction>
<xs:restriction base="xs:integer">
    <xs:minInclusive value="valor_maximo"/>
    <xs:maxInclusive value="valor_minimo"/>
</xs:restriction>
```

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

En esta unidad hemos aprendido:

- Las partes que conforman un documento XML y la jerarquía de sus elementos.
- La validación y/o definición de un documento mediante las reglas de las DTD.
- La validación de los documentos mediante *schema* para documentos en los que queremos insertar reglas más estrictas, como el tipo de datos de los diferentes nodos.

**MP_0373. Lenguajes de marcas y sistemas de gestión
de información**

**UF3. Lenguajes de almacenamiento y transmisión de
información**

3.3. JSON



Índice

Objetivos	3
JSON	4
¿Qué es JSON?	4
Sintaxis JSON	5
Tipos de datos	6
JSON frente a XML	6
Despedida	8
Resumen.....	8

Objetivos

Conocer la estructura y características de los documentos JSON.

JSON

¿Qué es JSON?

JSON es una notación que se basa en la forma que tiene JavaScript de guardar los datos en los objetos. Nos permite crear estructuras de datos de forma fácil y rápida. Y lo que es más importante, de forma óptima.

Sintaxis

Para crear un documento JSON debemos seguir las siguientes reglas:

- Los datos deben estar separados por comas.
- Los datos se pondrán con la estructura "*clave:valor*".
- Tanto las *claves* como los *valores* irán entre comillas.
- Los objetos JSON se definen por llaves "{}".
- Los *arrays* van con corchetes [].
- Tanto los objetos como los *arrays* pueden contener otros objetos o *arrays*.

```
{  
  "nombre": "Jose",  
  "edad": "20",  
  "genero": "masculino",  
  "email": "correodejose@dominio.com",  
  "localidad": "Madrid",  
  "telefono": "91000000",  
  "mando": "true",  
  "marco": "<iframe width=560 height=315 src=https://www.youtube.com/embed/QCEU-vba8mw frameborder=0 allowfullscreen></iframe>"  
}
```

Sintaxis JSON

La sintaxis JSON es igual a la de JavaScript. Los datos se ponen en pares, como **clave** y **valor**.

```
"nombre":"valor"
```

1. Para definir un objeto utilizamos las llaves {}.

```
{  
"nombre":"valor"  
}
```

2. Para definir varios elementos los separaremos por comas ",".

```
{  
"nombre":"Juan",  
"apellidos":"Gonzalez"  
}
```

3. También podemos poner varios objetos, igualmente los sepáramos por comas ",".

```
{  
"nombre":"Juan",  
"apellidos":"Gonzalez"  
,  
{  
"nombre":"María",  
"apellidos":"Alonso"  
,  
{  
"nombre":"José",  
"apellidos":"amohedo"  
}
```

4. JSON permite la definición de *arrays* con el uso de los corchetes [].

```
{  
"nombre":"juan",  
"edad":30,  
"coches":[ "Ford", "BMW", "Fiat" ]  
}
```

5. Los *arrays* pueden contener tanto valores como objetos.

Recuerda, un *array* es un conjunto de elementos organizados en memoria en posiciones contiguas. Todos los elementos son del mismo tipo y se puede acceder a ellos a partir de subíndices.

```
{  
    "nombre": "juan",  
    "edad": 30,  
    "coches": [  
        { "nombre": "Ford", "modelo": [ "Fiesta", "Focus", "Mustang" ] },  
        { "nombre": "BMW", "modelo": [ "320", "X3", "X5" ] },  
        { "nombre": "Fiat", "modelo": [ "500", "Panda" ] }  
    ]  
}
```

Tipos de datos

JSON deriva de JavaScript, pero solo en el modelo de datos, por lo que elementos como funciones o datos no definidos no están contemplados.

Los tipos de datos soportados por JSON son:

- **string:** cadenas de texto.
- **number:** datos numéricos.
- **object:** objetos JSON.
- **arrays:** arrays, vectores, etc.
- **boolean:** datos booleanos (true, false).
- **null:** valores nulos.

JSON frente a XML

Tanto JSON como XML se usan para intercambiar información entre ordenadores o programas, además de almacenar datos, como configuraciones de un servidor o aplicación, etc.

Inicialmente podemos pensar que JSON, al ser más breve a la hora de definirlo, es mejor que XML. Pero los dos están profundamente implantados en todos los lenguajes y sistemas informáticos.

JSON tiene una dependencia profunda de los lenguajes de programación, dado que necesita ser tratado para obtener la información, mientras que con XML podemos definir reglas y está más abierto a los sistemas.

Veamos un ejemplo con la misma información en los dos sistemas:

```
{"employees": [  
    { "firstName": "John", "lastName": "Doe" },  
    { "firstName": "Anna", "lastName": "Smith" },  
    { "firstName": "Peter", "lastName": "Jones" }  
]
```

Ejemplo JSON.

```
<employees>  
  <employee>  
    <firstName>John</firstName> <lastName>Doe</lastName>  
  </employee>  
  <employee>  
    <firstName>Anna</firstName> <lastName>Smith</lastName>  
  </employee>  
  <employee>  
    <firstName>Peter</firstName> <lastName>Jones</lastName>  
  </employee>  
</employees>
```

Ejemplo XML.

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

En esta unidad hemos aprendido:

- Estructura y sintaxis de los datos en JSON.
- Tipos de datos soportados por JSON.
- Algunos ejemplos de aplicación.

4.1. Validación



Índice

Objetivos	3
Introducción.....	4
Conceptos	4
Documentos bien formados.....	4
Documentos válidos.....	5
Definición de tipos de documentos	6
Métodos de validación XML	
Validar documentos con DTD.....	7
Validar documentos con esquemas (schema)	8
Validar XML con Relax NG	9
Validar con Schematron	10
Despedida	11
Resumen.....	11

Objetivos

Con esta unidad perseguimos los siguientes objetivos:

- **Recordar** las diferencias entre un documento "**bien formado**" y "**válido**".
- **Conocer** qué métodos sirven para **validar documentos XML**.

Introducción

Conceptos

Los **errores** en los documentos **XML** implican que los datos que se incluyen y las directivas van a provocar un mal funcionamiento de las aplicaciones desarrolladas.

La especificación XML en W3C establece que un programa debe dejar de procesar un documento XML si encuentra un error. La razón es que el software XML debe ser pequeño, rápido y compatible.

Sin embargo, los navegadores de HTML mostrarán los documentos aunque tengan errores. Los navegadores HTML tienen mucho código que está definido solamente para el tratamiento de los errores de HTML.

Teniendo en cuenta que la única forma estructurada de declarar válido un documento XML sería la validación, esta debería refrendar:

- **La corrección de los datos:** aunque los datos no se validan o corrigen en su contenido, sí permite la detección de datos incorrectos por formatos nulos o valores fuera de rango.
- **La integridad de los datos:** comprobar que toda la información obligatoria está en el documento es una de las ventajas de la validación.
- **Establecimiento de un protocolo en el documento:** usando la validación se comprueba que el emisor y receptor traducen y tratan el documento de la misma manera y lo interpretan igual.

Documentos bien formados

Un documento XML con sintaxis correcta se denomina "**Well Formed**" o **bien formado**.

Como ya hemos visto en la unidad anterior, un documento XML debe estar bien formado.

Repasemos las reglas de sintaxis básicas definidas para los lenguajes SGML:

- Los documentos XML deben tener **un elemento raíz** solamente.
- Los elementos XML deben tener una **etiqueta de apertura y otra de cierre**.
- Las etiquetas de XML distinguen entre mayúsculas y minúsculas, **son case-sensitive**.
- Los elementos XML deben anidarse dentro de la **jerarquía definida**.
- Los valores de **atributos XML deben ser indicados**.

Un documento **XML válido** se define en la especificación XML como un documento XML **bien formado**, que también se **ajusta a las reglas de una definición de tipo de documento (DTD)**. Los documentos XML bien formados simplemente marcan las páginas con etiquetas descriptivas.

Un documento **XML bien formado no necesita una DTD**, debe ajustarse a las reglas de sintaxis XML.

Si todas las etiquetas de un documento se forman correctamente y siguen las directrices XML, entonces un documento se considera como bien formado (pero puede no ser válido).

Documentos válidos

Un documento XML "bien formado" no significa que sea "válido". Un documento XML "válido" debe estar bien formado, ya que debe cumplir la sintaxis básica de formación de documentos bien formados de XML. Por lo tanto, además de estar bien formado, un documento deberá ajustarse a una definición de tipo de documento DTD o schema.

Según la Organización W3C, los documentos válidos son **los que validan contra una DTD**. Las reglas de validez significan que un documento **cumple con las restricciones establecidas** dentro de una DTD. Por lo tanto, las etiquetas o entidades deben estar en conformidad con las reglas y relaciones establecidas dentro de una DTD.

Sin embargo, **no hay control** sobre si una etiqueta o entidad es correcta. Así, una etiqueta de cabeza de primer nivel podría aplicarse a un objeto de cabeza de segundo nivel y ser válida, mientras que es incorrecta.

El énfasis en **documentos bien formados** se ha desarrollado dentro de la industria editorial, donde el uso de la información delimitada por el ángulo izquierdo y derecho se ha convertido en un problema. El énfasis en el documento bien formado permite definir, delimitar y anidar contenido para ser manejado dentro de programas que no son XML como tales, pero exhiben las características o potencial para estar bien formado.

Para que un XML sea válido, además de estar bien formado, el documento deberá ajustarse a una definición de tipo de documento DTD o schema.

Definición de tipos de documentos

Una definición de tipo de documento define las reglas y los elementos y atributos legales para un documento XML.

Existen varias definiciones de tipos de documentos diferentes que se pueden utilizar con XML:

DTD

Es la definición del tipo de documento original.

- Es el formato nativo y el más antiguo.
- Utiliza una sintaxis no-XML.
- Es el método más sencillo, pero presenta limitaciones.

XML Schema

Es una alternativa basada en XML para DTD.

- Evolución de la DTD descrita por el W3C.
- Utiliza sintaxis XML.
- Es más complejo y potente.

Relax NG

Es una nueva gramática de definición adoptada por la ISO.

- Muy intuitivo y más fácil de entender que el XSD.
- Puede utilizar sintaxis XML o una propia parecida a JSON.
- Convertido recientemente en un estándar ISO.

Schematron

Es un lenguaje de esquema estructural expresado en XML utilizando un pequeño número de elementos y XPath.

- Se basa en reglas en vez de en gramática.
- Convertido recientemente en un estándar ISO.
- Utiliza sintaxis XML, donde se definen reglas.

Una definición de tipo de documento define las reglas y los elementos y atributos legales para un documento XML.

Métodos de validación XML

Validar documentos con DTD

DTD define los elementos del documento, sus relaciones e información adicional que puede ser incluida, como atributos, entidades y/o notaciones. **Es el formato de esquema nativo**.

Este método resulta **el más sencillo de utilizar**, ya que lo que realiza es una comprobación del cumplimiento del DTD definido. El problema es que no soporta nuevas ampliaciones del XML. **Utiliza una sintaxis no-XML** para definir la estructura o modelo de contenido de un documento XML válido:

- Define todos los elementos.
- Define las relaciones entre los distintos elementos.
- Proporciona información adicional que puede ser incluida en el documento (atributos, entidades, notaciones).
- Aporta comentarios e instrucciones para su procesamiento y representación de los formatos de datos.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Recordatorio</heading>
    <body>¡No te olvides de nuestra reunión de esta semana!</body>
</note>
```

En este XML vemos como el DOCTYPE referencia al documento DTD para su validación.

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

Este es el fichero DTD con las reglas para validar el documento.

Validador de documentos XML

La validación se realiza frente a cualquier esquema XML o DTD declarado dentro del documento XML. <https://www.xmlvalidation.com/>

Validar documentos con esquemas (schema)

Recordemos que los **esquemas XML** son un lenguaje más avanzado que DTD y permiten especificar mucho mejor el **sistema de datos**. Una vez validado es posible expresar la estructura y contenidos del documento.

Un esquema XML se utiliza para describir la estructura de un documento XML especificando los elementos válidos que pueden ocurrir en un documento, el orden en el que pueden ocurrir y expresando restricciones sobre ciertos aspectos de estos elementos.

Un esquema XML contiene: **vocabulario** (elementos y atributos), **contenido** (estructura y relaciones) y **tipos de datos**.

```
<?xml version="1.0" encoding="UTF-8"?>
<addresses xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation='test.xsd'>
    <address>
        <name>Joe Tester</name>
        <street>Baker street 5</street>
        <wrongExtraField/>
    </address>
</addresses>
```

Fichero XML.

```
<xss:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
    <xss:element name="addresses">
        <xss:complexType>
            <xss:sequence>
                <xss:element ref="address" minOccurs='1' maxOccurs='unbounded' />
            </xss:sequence>
        </xss:complexType>
    </xss:element>

    <xss:element name="address">
        <xss:complexType>
            <xss:sequence>
                <xss:element ref="name" minOccurs='0' maxOccurs='1' />
                <xss:element ref="street" minOccurs='0' maxOccurs='1' />
            </xss:sequence>
        </xss:complexType>
    </xss:element>

    <xss:element name="name" type='xs:string' />
    <xss:element name="street" type='xs:string' />
</xss:schema>
```

Fichero schema.

Validar XML con Relax NG

Relax NG es un lenguaje de esquema basado en la gramática y es más sencillo y fácil de entender que un esquema XML, ya que tiene un alto poder expresivo.

Las características clave de Relax NG son:

- Es **muy simple**.
- Es **fácil de aprender**.
- Tiene una sintaxis XML y una **sintaxis compacta no XML**.
- **No cambia** el conjunto de información de un documento XML.
- Soporta espacios de **nombres XML**.
- Trata atributos uniformemente con elementos en la medida de lo posible.
- Tiene un soporte sin restricciones para contenido no ordenado.
- Tiene un soporte sin restricciones para contenido mixto.
- Tiene una sólida base teórica.
- Puede asociarse con un lenguaje de datos distinto (por ejemplo, los tipos de datos del esquema W3C XML).

```
<addressBook>
  <card>
    <name>John Smith</name>
    <email>js@example.com</email>
  </card>
  <card>
    <name>Fred Bloggs</name>
    <email>fb@example.net</email>
  </card>
</addressBook>
```

Fichero XML.

```
<element name="addressBook" xmlns="http://relaxng.org/ns/structure/1.0">
  <zeroOrMore>
    <element name="card">
      <element name="name">
        <text/>
      </element>
      <element name="email">
        <text/>
      </element>
    </zeroOrMore>
  </element>
```

Relax NG.

RELAX NG Tutorial

Tutorial en inglés. <http://relaxng.org/tutorial-20011203.html>

Validar con Schematron

Schematron es un lenguaje basado en XML para validar documentos de instancia XML. Se utiliza para hacer afirmaciones sobre datos en un documento XML y también se utiliza para expresar las reglas operativas y de negocio. *Schematron* es una norma ISO.

Se basa en afirmaciones en vez de en gramática y utiliza expresiones de acceso.

En concreto, se basa en una serie de reglas y utiliza expresiones de acceso para definir lo que se permite en un documento XML, haciendo que si se cumplen sea un documento "válido". Este método es el más flexible en cuanto a la descripción de estructuras relacionales y se suele utilizar junto a otros lenguajes, como el anterior Relax NG.

El componente *Schematron* utiliza la implementación de ISO *Schematron*. Es una implementación basada en XSLT. Las reglas *Schematron* se ejecutan a través de cuatro *pipelines* de XSLT, lo que genera una XSLT final que se utilizará como base para ejecutar la aserción contra el documento XML.

El componente se escribe de forma que las reglas de *Schematron* se carguen al inicio del punto final (solo una vez) para minimizar la sobrecarga de instancia de un objeto de plantillas Java, que representa las reglas.

Tutorial

En este enlace encontrarás documentación sobre *Schematron*. <http://schematron.com/>

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

En esta unidad hemos repasado la validación para documentos XML, las reglas y métodos que se utilizan para validar documentos XML.

Un documento bien formado no es lo mismo que un documento validado. Para que un documento XML esté bien formado debe cumplir las reglas de los lenguajes SGML:

- Solo debe contener caracteres Unicode legales correctamente codificados. Ninguno de los caracteres de sintaxis especiales como < y & deben usarse, excepto cuando se realizan sus funciones de marcado-delineación.
- Las etiquetas de "inicio", "fin" y "elemento vacío" que delimitan los elementos se deben anidar correctamente, sin que falte ninguno y sin superponerse. Las etiquetas de "inicio" y de "fin" deben coincidir exactamente.
- Los nombres de las etiquetas no pueden contener ninguno de los caracteres: "# \$% & '() * +, /; <=>? @ [\] ^ ` { } ~, ni un carácter de espacio, y no puede comenzar con -, o un dígito numérico.

Un documento XML "válido" debe estar bien formado, ya que debe cumplir la sintaxis básica de formación de documento bien formado de XML. Pero además deberá ajustarse a una definición de tipo de documento DTD o *schema*.

4.2. Transformación de contenidos



Índice

Objetivos	3
Introducción.....	4
Concepto	4
Lenguajes de transformación	4
XLS	7
Lenguajes de transformación: XSL	7
Beneficios XSL.....	8
Hojas de estilo	8
XPATH	11
Concepto	11
Características	11
Elementos de XPath	12
Los predicados	13
Test.....	14
Ejes	15
Funciones	16
XSLT	18
XSL Transformation o XSTL.....	18
De XML a XML.....	18
De XML a xHTML	21
Procesador XSLT	22
Funciones XSLT.....	23
Elemento <xsl:template>.....	23
Elemento <xsl:value-of>	24
Elemento <xsl:for-each>	24
Elemento <xsl:sort>.....	25
Elemento <xsl:if>	26
Elemento <xsl:choose>.....	26
Elemento <xsl:apply-templates>.....	27
Despedida	29
Resumen.....	29

Objetivos

Con esta unidad perseguimos los siguientes objetivos:

- Conocer los conceptos de transformación de documentos XML.
- Conocer cuáles son los lenguajes de transformación que hay en el mercado como estándares.
- Conocer el lenguaje de transformación XSL.
- Conocer el lenguaje de transformación XPath.
- Aprender a realizar transformaciones con los estilos XSLT.

Introducción

Concepto

Los documentos XML proporcionan información estructurada pero que no tiene formato de salida. Eso quiere decir que, si queremos presentar esa información con un formato determinado, deberíamos transformar el documento en otro al que se le aplique el formato deseado.

Las tecnologías que podemos usar **para realizar esta transformación** son las siguientes:

- **XSLT:** (eXtensible Stylesheet Language Transformation) sirve para definir cuál será el modo de transformar un documento XML en otro.
- **XSL-FO:** (eXtensible Stylesheet Language - Formatting Object) se utiliza para transformar XML en un formato legible e imprimible por una persona, por ejemplo en un documento PDF.
- **Xpath:** permite el acceso a los componentes de un documento XML.

Las transformaciones posibles son:

- **Desde XML hacia XML:** usado para transformar documentos XML en otros con información añadida, borrada u organizada de otra forma.
- **Desde XML hacia otros formatos:**
 - **XML hacia XHTML:** a través de estilos y los lenguajes XPath y XSLT.
 - **XML hacia formatos estándares, como PDF, SVG, RTF u otros:** a través del lenguaje de transformación XSL-FO.

Lenguajes de transformación

Los documentos XML definen datos cuando se usan como contenedores de estructuras complejas de datos.

En su sintaxis no hay manera de definir el formato de salida de datos, por lo que se deben usar otros estándares para que dichos datos puedan ser transformados de una manera formateada.

Alguna vez puede ser necesario cambiar el árbol de datos de un XML para transformarlo en otro que tenga los mismos datos, pero con una estructura distinta.

Los **lenguajes de transformación de XML** son muchos. Veamos algunos ejemplos:

XSLT

Es un lenguaje para transformar documentos XML en otros documentos XML o en otros formatos, como HTML para páginas web, texto plano, o XSL Formatting Objects.

XQuery

Es un lenguaje de consulta y programación funcional que transforma las consultas y colecciones de datos estructurados y no estructurados, por lo general en forma de XML, en texto y con extensiones específicas del proveedor para otros formatos de datos (JSON, binario, etc.).

XProc

Es una recomendación W3C para definir un lenguaje de transformación XML que define tuberías XML.

XML document transform

Un lenguaje de transformación XML de Microsoft. Es un lenguaje de programación diseñado específicamente para transformar una entrada de un documento XML en una salida de documento que satisface alguna meta específica.

STX

Es un lenguaje de transformación XML que pretende tener alta velocidad, bajo consumo de memoria y ser una alternativa a la versión 1.0 y 2.0. de XSLT.

XML script

Permite la creación, el almacenamiento y la manipulación de variables y datos durante el procesamiento. XML es un lenguaje de marcado para documentos que contienen tanto el contenido (es decir, texto, imágenes, etc.) y alguna indicación de lo que desempeña el papel de contenido (por ejemplo, si está en un encabezado de sección o un pie de página, etc.).

FXT

Es una herramienta de transformación XML funcional, implementado en el estándar ML.

XDuce

Es un lenguaje con una sintaxis sencilla. Está escrito en ML.

C~~D~~uce

Extiende XDuce a usos de propósito general como lenguaje de programación funcional.

XACT

Es un sistema basado en Java para las transformaciones XML de programación. Como características notables incluye plantillas XML como valores inmutables y un análisis estático para garantizar la seguridad de tipos utilizando los tipos de esquema XML.

XFun

Es un lenguaje funcional para definir transformaciones entre los árboles de datos XML.

XStream

Es un lenguaje de transformación funcional para documentos XML basados en CAML. Las transformaciones se pueden producir de manera masiva.

HaXml

Es una biblioteca y una colección de herramientas para escribir transformaciones XML en Haskell. Su enfoque es muy consistente y potente.

Scala

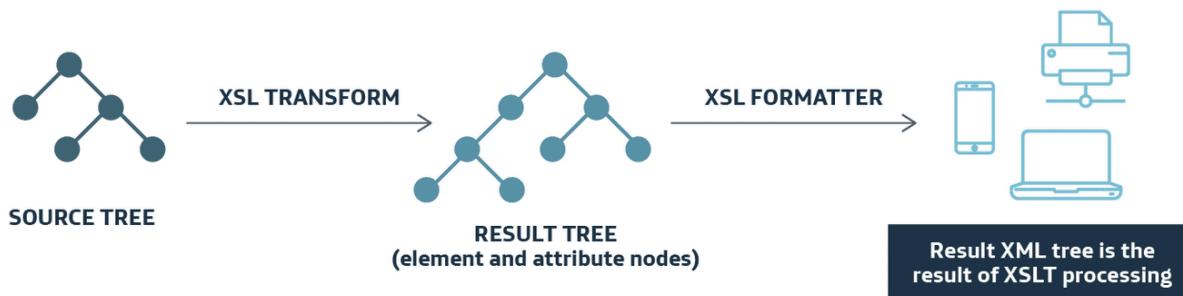
Un lenguaje funcional y orientado a objetos, de uso general, con un apoyo específico para la transformación de XML en forma de XML, coincidencia de patrones, literales y expresiones, junto con las bibliotecas XML estándar.

XLS

Lenguajes de transformación: XSL

Una hoja de estilo XSL especifica la presentación de una clase de documentos XML mediante la descripción de cómo una instancia de la clase se transforma en un documento XML que utiliza el vocabulario de formato.

XSL es un Lenguaje Extensible de Hojas de Estilo, cuyo objetivo principal es mostrar cómo debería estar estructurado el contenido, cómo debería ser diseñado el contenido de origen y cómo debería ser paginado en un medio de presentación, como puede ser una ventana de un navegador web, un dispositivo móvil, un conjunto de páginas de un catálogo, un informe o un libro.



En la construcción del árbol el proceso de transformación también añade la información necesaria para el formato de salida de dicho árbol.

Un **procesador XSL** de hoja de estilo acepta un documento XML y una hoja de estilo XSL y **produce la presentación de dicho contenido** de código XML que se pretendía por el diseñador de ese estilo.

Hay dos aspectos de este proceso de presentación:

- En primer lugar, la construcción de un árbol de resultados desde el árbol de código fuente XML.
- En segundo lugar, interpretar el árbol de resultados para producir resultados con formato adecuado para su presentación en una pantalla, en papel, o en otros medios.

El primer aspecto se denomina "transformación árbol" y el segundo se llama "formateo". El proceso de formateo es realizado por el formateador, que puede ser simplemente un motor de representación dentro de un navegador.

Beneficios XSL

A diferencia del caso de HTML, los nombres de los elementos de XML no tienen presentación semántica intrínseca.

A falta de una hoja de estilo, **un procesador no podría saber cómo representar el contenido de un documento XML** que no sea como una cadena de caracteres no diferenciado.

XSL proporciona un modelo integral y un vocabulario para escribir hojas de estilo usando la sintaxis XML.

Otras ventajas de XSL son:

- Mantener un formato de salida de un documento independientemente de su estructura y de cualquier otra especificación.
- Reutilizar el mismo estereotipo de formato para crear diversos documentos.
- Favorecer la automatización de tareas, ya que es compatible con lenguajes de programación, especialmente con ECMAScript.

Hojas de estilo

Del mismo modo que aplicamos estilos a los documentos HTML, con los documentos de datos XML podemos hacer lo mismo si queremos mejorar su visualización en un explorador. Siendo fácil de aprender y utilizar, su desventaja es que solo sirve para visualizar documentos en un navegador.



Para usar estilos CSS en un documento XML añadimos la siguiente línea:

```
<?xmlstylesheet type="text/css" href="fichero.css" ?>
```

Fichero.css es el nombre del archivo CSS que contiene las reglas de formato.

El fichero con los estilos CSS seguirá las mismas normas que aprendimos en la visualización de documentos, aplicando las diferentes reglas a las etiquetas.

Ejemplo

Vamos a aplicar estilos a un documento XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<?xmlstylesheet type="text/css" href="estilos.css" ?>
<alumnos>
    <curso>
        <alumno>
            <nombre>Juan</nombre>
            <nota>9</nota>
            <faltas>1</faltas>
        </alumno>
        <alumno>
            <nombre>Luisa</nombre>
            <nota>10</nota>
            <faltas>0</faltas>
        </alumno>
        <alumno>
            <nombre>Sergio</nombre>
            <nota>5</nota>
            <faltas>7</faltas>
        </alumno>
        <alumno>
            <nombre>María</nombre>
            <nota>8</nota>
            <faltas>3</faltas>
        </alumno>
    </curso>
</alumnos>
```

Documento XML.

En el ejemplo, además de ver la estructura de datos típica en los documentos XML, podemos observar cómo se realiza la llamada a un archivo CSS:

```
<?xmlstylesheet type="text/css" href="estilos.css" ?>
```

En este archivo aplicamos las reglas para definir su visualización en el navegador.

```
alumno{
    padding: 10px;
    display: block;
    overflow: hidden;
}

alumno:nth-child(2n){
    background-color: #ccc;
}
```

```
nombre, nota, faltas{  
    font-family: arial;  
    font-size: 18px;  
    width: 80px;  
    border-right: 2px solid #aaa;  
    display: block;  
    float: left;  
    text-align: center;  
}  
  
nota{  
    font-size: 22px;  
    color:green;  
}  
  
faltas{  
    font-size: 22px;  
    color:red;  
}
```

Archivo *estilos.css*.

El resultado al abrir el documento en un navegador, después aplicar las reglas CSS, será:



				Juan	9	1	
				Luisa	10	0	
				Sergio	5	7	
				María	8	3	

XPATH

Concepto

XPath debe su nombre al uso de una notación de caminos o *path*, similar al de las URL para **navegar a través de la estructura** jerárquica de un documento XML.

XPath es un lenguaje de direccionamiento de partes o piezas de un documento XML, diseñado para ser **utilizado tanto por XSLT como por XPointer**. También tiene la capacidad de manipulación de cadenas de caracteres, números y booleanos.

XPath utiliza una sintaxis compacta, **no XML**, para facilitar el uso de XPath de los valores de los atributos en las URI y en XML.

XPath opera sobre la estructura lógica de un documento XML, en lugar de operar en su sintaxis. XPath es un elemento importante en el estándar XSLT.

XPath navega por los nodos que un XML genera. Estos nodos suelen ser de elementos, de atributos y de texto.



La forma de acceder a los nodos del árbol generado es a partir de expresiones. Una expresión XPath es un predicado que se aplica sobre una estructura de árbol correspondiente a la jerarquía de los datos de un documento XML y devuelve todo lo que encaja con ese predicado.

```
/tienda/lista[1]
```

Selecciona el primer elemento *lista* que es hijo del elemento *tienda*.

Características

XPath utiliza expresiones de ruta para seleccionar nodos o conjuntos de nodos en un documento XML. Estas expresiones de ruta se parecen mucho a las expresiones que se utilizan en el sistema de archivos de un ordenador.

Una ruta de ubicación se puede utilizar como una expresión. La expresión devuelve el conjunto de nodos seleccionados por la ruta.

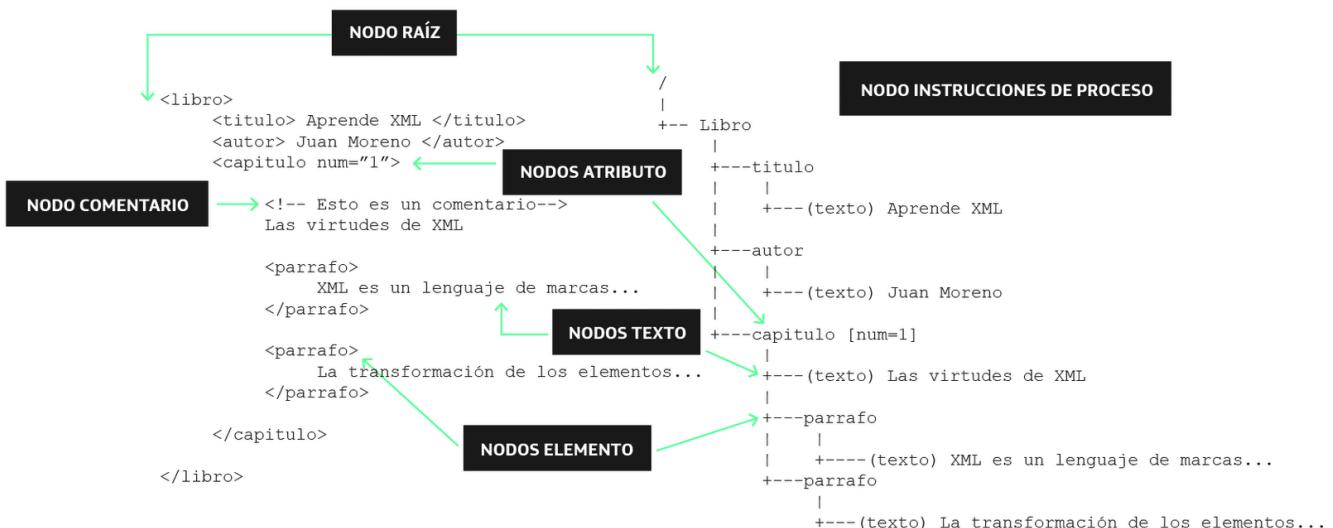
El operador "**|**" calcula la unión de los operandos, que deben ser conjuntos de nodos. Los predicados se utilizan para filtrar expresiones, de la misma manera que se utilizan en las rutas de ubicación.

- XPath es una sintaxis para definir partes de un documento XML.
 - XPath utiliza expresiones de ruta para navegar en documentos XML.
 - XPath contiene una biblioteca de funciones estándar.
 - XPath es un elemento importante en el estándar XSLT.
 - XPath es una recomendación del W3C.

Elementos de XPath

Vamos a estudiar los elementos con los que trabaja XPath a nivel de objetos. En el ejemplo vemos la creación de árbol y definición de los elementos.

Puedes ampliar la información sobre [XML Path Language \(XPath\)](#).



Nodo comentario

Se pueden integrar comentarios dentro del documento de la siguiente manera:
<!--esto es un comentario-->

En el árbol también se generan nodos para cada nodo con comentarios.

Nodos atributo

Cada nodo elemento puede tener o tiene asociado un conjunto de nodos atributo; el elemento es el padre de cada uno de estos nodos atributo.

Sin embargo, un **nodo atributo no es un hijo de su elemento padre**, por lo que no se consideran hijos del elemento al que están asociados, sino etiquetas añadidas al nodo

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.

Nodos texto

Son aquellos caracteres del documento que no están marcados con ninguna etiqueta y están agrupados en un nodo. Este tipo de nodos no tiene hijos. Cada carácter dentro de una sección CDATA se trata como dato de carácter. Los caracteres dentro de comentarios, instrucciones de procesamiento y valores de atributos, no producen nodos de texto.

Nodos elemento

Son cada uno de los elementos del documento XML. Todos ellos tienen un elemento padre (excepto el nodo raíz). Pueden tener identificadores únicos. Para ello es necesario que un atributo esté definido en una DTD o en un fichero *schema XML* asociado. Esto nos permite referenciar dicho elemento de forma mucho más directa.

Un documento no puede tener dos elementos con el mismo identificador único.

Nodo Raíz

Es el nodo que contiene toda la información XML. No debemos confundirlo con el elemento raíz del XML, pues en XPath se coloca justo antes de este, por lo que lo contiene. XPath lo identifica con "/".

En el ejemplo el elemento raíz está justo antes del nodo </libro>.

Nodo instrucciones proceso

Hay un nodo instrucción de procesamiento para cada una de las instrucciones, con la excepción de aquellas que se produzcan dentro de la declaración de tipo de documento.

En el árbol se generan nodos para cada nodo con comentarios y con instrucciones de proceso. Al contenido de estos nodos se puede acceder con la propiedad *string-value*.

La declaración XML no es una instrucción de procesamiento, por lo tanto, no hay ningún nodo instrucción de procesamiento correspondiente a la declaración XML.

Los predicados

Los predicados usados en XPath filtran un conjunto de nodos con respecto a un eje para producir un nuevo conjunto de nodos.

Un predicado se utiliza para filtrar una secuencia de nodos aplicando una prueba específica. La expresión de predicado se incluye entre corchetes y se enlaza con el último nodo de una expresión de ruta de acceso.

Existen dos tipos de elementos que podemos incluir en un predicado:

- Test.
- Ejes.

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.

```
//parrafo/*
```

Selecciona todos los nodos (principales) descendientes de *parrafo*.

```
//parrafo/node ()
```

Selecciona todos los nodos descendientes de *parrafo* (de cualquier tipo: texto, comentarios...).

Solo aplicables a ejes de contenido:

- **text()**, devuelve cualquier nodo de tipo texto.
- **comment()**, devuelve cualquier nodo de tipo comentario.
- **processing-instruction()**, devuelve cualquier tipo de instrucción de proceso.

```
//parrafo/text ()
```

Selecciona el texto de todos los nodos *parrafo*.

```
//parrafo//text ()
```

Selecciona todos los textos dependientes de todos los nodos *parrafo*.

Una doble barra: **//** (en su forma larga **descendant::**) selecciona TODOS los nodos que descienden del conjunto.

Ejes

Son los que permiten seleccionar el conjunto de nodos dentro del nodo contexto que cumple un patrón, que es el que se define en la expresión. Pueden ser o no de contenido.

ancestor: devuelve todos los nodos de los que el nodo contexto es descendiente.

```
//ancestor::parrafo/ancestor::*
```

Devuelve todos los hijos que tiene el nodo *parrafo*.

```
//*[@@href]/ancestor::capitulo
```

Devuelve los nodos *capitulo* que tienen hijos con el atributo *href*.

child: es el eje utilizado por defecto para devolver los hijos de un nodo. Su forma habitual es la barra, "/", también puede ponerse /*child*:::

```
/libro/titulo
```

Devuelve los nodos *titulo* de *libro* (versión abreviada con una /).

```
/libro/child::titulo
```

Devuelve los nodos *titulo* de *libro* (sin abreviar).

attribute: permite seleccionar los atributos que deseemos. Es el único eje que no es de contenido.

```
/libro/capitulo/@ref
```

Selecciona el atributo *ref* que tengan las etiquetas *capitulo*.

```
/libro/capitulo[@ref]/*
```

Selecciona todos los nodos hijos que tengan como atributo en la etiqueta *capitulo* el atributo *ref*.

```
/libro/capitulo[@autor='Juan']/*
```

Selecciona todos los nodos hijos que tengan como atributo en la etiqueta *capitulo* el atributo *ref* con el valor "Juan".

descendant: permite seleccionar todos los nodos que descienden del conjunto de nodos contexto.

Se corresponde con la doble barra, "//", aunque se puede usar *descendant*:::

```
/libro//parrafo
```

Selecciona todos los nodos descendientes de *parrafo*, no solo los hijos directos.

self: se refiere al nodo contexto y se corresponde con el punto ":".

```
./parrafo
```

Devuelve los nodos *parrafo* descendientes del nodo contexto.

parent: selecciona los nodos padre. Para referirnos a él usamos los dos puntos, "..".

```
../capítulo
```

Devuelve los nodos *capítulo* descendientes del nodo padre.

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

Funciones

Las funciones predeterminadas para XPath pueden incluirse en las expresiones.

Cada función en la biblioteca de funciones se especifica utilizando un prototipo **que proporciona el tipo de retorno, el nombre de la función y el tipo de argumentos**.

Si un tipo de argumento es seguido por un signo de interrogación el argumento es opcional, de lo contrario el argumento es requerido.

Las funciones que podemos utilizar son:

- **boolean()**, al aplicarla sobre un conjunto de nodos devuelve *true* si no está vacío.
- **not()**, al aplicarla sobre un predicado devuelve *true* si el predicado es falso, y *false* 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 el nombre de un nodo.
- **local-name()**, devuelve el nombre del nodo actual o del primer nodo de un conjunto de nodos.
- **namespace-uri()**, devuelve la 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.
- **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 el número de caracteres que tiene una cadena de caracteres.
- **sum()**, devuelve la suma de los valores numéricos de cada nodo en un conjunto de nodos determinado.

Funciones XPath

Si deseas saber más sobre las funciones de XPath puedes visitar este enlace.

<https://www.w3.org/TR/1999/REC-xpath-19991116/#corelib>

XSLT

XSLT (XSL Transformation) es el lenguaje de hojas de estilo recomendado para manejar las transformaciones de documentos XML en cualquiera de sus versiones.

XSLT es mucho más sofisticado que CSS, ya que con XSLT se pueden agregar y/o eliminar elementos en la representación y también atributos desde o hacia el archivo de salida.

También a través de XSLT **se pueden reorganizar y ordenar elementos**, realizar pruebas y tomar decisiones sobre qué elementos ocultar y mostrar, ya que cuenta con una parte procedimental en la que se pueden tomar decisiones.

XSLT utiliza el lenguaje XPath para encontrar información en un documento XML.



De XML a XML

Para poder transformar ficheros XML que tienen un formato especificado en otro fichero XML diferente, debemos usar una hoja de estilo XSLT que se puede aplicar para traducir los datos de una gramática XML a otra.

Aunque más adelante profundizaremos en los diferentes métodos que tenemos a nuestra disposición gracias a XSLT, veamos un ejemplo de transformación de un XML a otro. En el siguiente ejemplo podemos ver que el XML está estructurado con la información almacenada en atributos.

El objetivo de este ejemplo es crear un nuevo XML con la información en etiquetas, a excepción del nombre de la tienda, que se dejará como atributo.

La estructura original de cada objeto es:

```
<item type="tienda" nombre="amazon" codigo="1102" producto="tablet 7 pulgadas" precio="28.875"/>
```

Y se quiere conseguir esta otra estructura para cada elemento tienda:

```
<tienda marca="amazon">
  <identificacion>tablet 7 pulgadas</identificacion>
  <codigo>1102</codigo>
  <precio>28.875</precio>
</tienda>
```

```
<?xml version="1.0"?>
<productos>
  <item type="tienda" nombre="amazon" codigo="1102" producto="tablet 7 pulgadas"
precio="28.875"/>
  <item type="tienda" nombre="tiendaPC" codigo="1104" producto="tablet 10
pulgadas" precio="92.250"/>
  <item type="tienda" nombre="tiendaPC" codigo="1105" producto="tablet 14
pulgadas" precio="20.313"/>
</productos>
```

XML origen.

Para realizar la transformación creamos un XML con las etiquetas de procesado, que localizan los nodos en el documento original mediante XPath y lo transforman.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
  <portfolio>
    <xsl:for-each select="productos/item[@type='tienda']">
      <tienda>
        <xsl:attribute identificacion="marca">
          <xsl:value-of select="@nombre"/>
        </xsl:attribute>
        <identificacion><xsl:value-of select="@producto"/></identificacion>
        <codigo><xsl:value-of select="@codigo"/></codigo>
        <precio><xsl:value-of select="@precio"/></precio>
      </tienda>
    </xsl:for-each>
  </portfolio>
</xsl:template>
</xsl:stylesheet>
```

XML traductor.

Es pronto aún para entender toda la sintaxis del ejemplo de traducción, pero vamos a comentar algunas de las líneas más importantes para comprender su funcionamiento.

- Dado que se trata de un documento XML, lo declaramos como tal:
<?xml version="1.0"?>

- Tenemos que definir que el documento XML se trata de una hoja de estilos XSL:
`<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`
- Con la siguiente línea indicamos el formato del documento de salida. Este es un elemento de nivel superior y siempre debe ser un hijo de `<xsl:stylesheet>` o de `<xsl:transform>`.

Los atributos que usamos en este caso son: *method*, donde ponemos que será otro XML; e *indent*, donde indicamos que será "indentada" según jerarquía.

`<xsl:output method="xml" indent="yes"/>`

- Ahora se representa que se trata de una plantilla con una serie de acciones que se realizarán si el patrón de la plantilla encaja (*match*) con el árbol de nodos del XML: `<xsl:template match="/">`
- A partir de aquí es donde empezamos a definir la transformación que sufrirá nuestro documento. Insertando en la plantilla las etiquetas de la nueva estructura e indicando cuál será su contenido, realizando un rastreo en el XML original.
... `<portfolio> <xsl:for-each select="productos/item[@type='tienda']">`
`<tienda> <xsl:attribute identificacion="marca"> ...`

Podemos comprobar que la sintaxis de la etiqueta `xsl:for-each` realiza una iteración aplicando las transformaciones sobre los elementos nodo encontrados por medio de *select*, donde hemos colocado el prefijo de **XPath**.

```
<?xml version="1.0"?>
<portfolio>
  <tienda marca="amazon">
    <identificacion>tablet 7 pulgadas</identificacion>
    <codigo>1102</codigo>
    <precio>28.875</precio>
  </tienda>
  <tienda marca="tiendaPC">
    <identificacion>tablet 10  pulgadas</identificacion>
    <codigo>1104</codigo>
    <precio>92.250</precio>
  </tienda>
  <tienda marca="tiendaPC">
    <identificacion>tablet 14 pulgadas</identificacion>
    <codigo>1105</codigo>
    <precio>20.313</precio>
  </tienda>
</portfolio>
```

Resultado final de la transformación.

De XML a xHTML

Gracias a XSLT no solo podemos realizar transformaciones entre documentos XML. Como veremos, podemos crear muchos tipos de documentos, entre ellos documentos xHTML.

Veamos ahora un ejemplo similar al anterior, pero en este caso transformando un documento con una estructura nativa de XML a un formato web como el xHTML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<libro>
    <titulo>La conjura de los necios</titulo>
    <descripcion>Cuerpo del documento</descripcion>
</libro>
```

XML con los datos a transformar.

Como vemos en el ejemplo anterior, tenemos la información de un libro almacenado en un documento XML. Aunque, como ya aprendimos, podemos darle estilos CSS al documento para facilitar su visualización en los exploradores, esto no es aconsejable por temas relacionados con SEO o por la compatibilidad con otros procesos web.

Para solucionarlo creamos el siguiente XSLT, indicando la estructura típica de un HTML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:template match="libro">
        <html>
            <head>
                <title><xsl:apply-templates select="titulo" mode="head"/></title>
            </head>
            <body>
                <xsl:apply-templates/>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="titulo" mode="head">
        <xsl:value-of select="text()"/>
    </xsl:template>
    <xsl:template match="titulo">
        <h1><xsl:value-of select="text()"/></h1>
    </xsl:template>
    <xsl:template match="descripcion">
        <h3><xsl:value-of select="text()"/></h3>
    </xsl:template>
</xsl:stylesheet>
```

Documento XML traductor.

Al igual que el ejemplo donde convertimos un XML a otro XML, vamos transformando el documento.

```
<html>
  <head>
    <meta content="text/html; charset=UTF-8" http-equiv="Content-Type">
    <title> La conjura de los necios </title>
  </head>
  <body>
    <h1> La conjura de los necios </h1>
    <h3>Cuerpo del documento</h3>
  </body>
</html>
```

Documento xHTML resultante.

Como resultado obtenemos un documento xHTML completamente normalizado.



Vista del documento xHTML en un explorador.

Procesador XSLT

Los procesadores XSLT son aplicaciones capaces de procesar documentos XML y realizar transformaciones mediante hojas XSLT.

Cuando se procesa el documento se genera una representación del mismo como un árbol de nodos y lo va recorriendo nodo a nodo realizando las transformaciones.

El nodo que se está procesando en cada momento se denomina **nodo contexto**.

Este tipo de aplicaciones las podemos encontrar en lenguajes de servidor (PHP, ASP, .NET, JSP, etc.) y en entornos de cliente, como los exploradores web, o lenguajes como JavaScript.

XSL Transformation Tool (XSLT)

Procesador on-line de XSLT. <http://online-toolz.com/tools/xslt-transformation.php>

Pasos para la transformación

- Se analiza la hoja de transformación y se convierte en un árbol.
- El documento XML es procesado y convertido en una estructura de árbol.

- El procesador XSLT se posiciona en la raíz del documento XML; en este punto el procesador se encuentra en el contexto original.
- Los elementos que no formen parte del espacio de nombres de prefijo *xsl* son procesados sin transformaciones.
- El procesador aplica cada regla nodo a nodo.

Funciones XSLT

Los elementos *<xsl:stylesheet>* y *<xsl:transform>* son elementos similares que sirven para definir el elemento raíz de la hoja de transformaciones.

<xsl:stylesheet>

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

<xsl:transform>

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Como atributos obligatorios tiene **version**, que indica la versión del documento XSLT y **xmlns**, que marca el espacio de nombres del documento.

Para obtener acceso a las características, a los elementos y a los atributos de XSLT debemos declarar el espacio de nombres XSLT en la parte superior del documento.

El *xmlns xsl = "http://www.w3.org/1999/XSL/Transform"* apunta al espacio de nombres oficial del W3C XSLT. Si usamos este espacio de nombres se debe incluir el atributo *version = "1.0"*.

Esta etiqueta no reemplaza a la que define el tipo de documento XML:

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

Elemento *<xsl:template>*

El elemento *<xsl:template>* se utiliza para **crear plantillas para transformaciones**.

El atributo **match** se utiliza para asociar una plantilla con un elemento XML. También sirve para definir una plantilla para todo el documento XML.

El valor del atributo *match* se construye a través de una expresión XPath, es decir, *match = "/"* define el documento completo.

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

Elemento `<xsl:value-of>`

El elemento `<xsl:value-of>` puede utilizarse para extraer el valor de un elemento XML y añadirlo al flujo de salida de la transformación.

Usando el mismo ejemplo que hemos usado anteriormente podríamos incluir esta instrucción.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <body>
      <h2>Colección de libros</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
          <th>Titulo</th>
          <th>Autor</th>
        </tr>
        <xsl:for-each select="libreria">
          <tr>
            <td><xsl:value-of select="/libro/titulo"/></td>
            <td><xsl:value-of select="/libro/autor"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Elemento `<xsl:for-each>`

El elemento XSL `<xsl:for-each>` se puede utilizar para seleccionar cada elemento XML de un conjunto de nodos especificado recorriendo todos los nodos del tipo que indiquemos en el elemento.

Usando el mismo ejemplo que hemos usado anteriormente podríamos incluir este elemento.

También podemos filtrar la salida del archivo XML añadiendo un criterio al atributo `select` en el elemento `<xsl:for-each>`.

```
<xsl: for-each select = "libreria / libro [autor= 'Miguel de Cervantes']">
```

Los operadores de filtros legales son:

- `=` Igual.
- `!=` Distinto.
- `<<` Menor que.
- `>>` Mayor que.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <body>
      <h2>Colección de libros</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
          <th>Titulo</th>
          <th>Autor</th>
        </tr>
        <xsl:for-each select="libreria">
          <tr>
            <td><xsl:value-of select="/libro/titulo"/></td>
            <td><xsl:value-of select="/libro/autor"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Ejemplo de documento de transformación a xHTML.

Elemento *<xsl:sort>*

Podemos usar el elemento *<xsl:sort>* para organizar la salida por un filtrado específico. Por ejemplo, por el autor del libro:

```
<xsl:sort select="autor"/>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <body>
      <h2>Colección de libros</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
          <th>Titulo</th>
          <th>Autor</th>
        </tr>
        <xsl: for-each select = "libreria/libro">
          <xsl:sort select="autor"/>
          <tr>
            <td><xsl:value-of select="/libro/titulo"/></td>
            <td><xsl:value-of select="/libro/autor"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

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

Elemento *<xsl:if>*

Podemos usar una estructura condicional para poder filtrar qué elementos se renderizan o transforman y cuáles no. La sintaxis de este elemento es:

```
<xsl:if test="expression">
...alguna salida si la expresión es cierta...
</xsl:if>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <body>
      <h2>Colección de libros</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
          <th>Titulo</th>
          <th>Autor</th>
        </tr>
        <xsl: for-each select = "libreria/libro">
          <xsl:if test="precio > 10">
            <tr>
              <td><xsl:value-of select="titulo"/></td>
              <td><xsl:value-of select="autor"/></td>
              <td><xsl:value-of select="precio"/></td>
            </tr>
          </xsl:if>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Elemento *<xsl:choose>*

El elemento *<xsl:choose>* se utiliza junto con *<xsl:when>* y *<xsl:otherwise>* para expresar múltiples pruebas condicionales. La sintaxis de este elemento es:

```
<xsl:choose>
  <xsl:when test="expression">
```

```
... alguna salida ...
</xsl:when>
<xsl:otherwise>
    ... alguna salida ...
</xsl:otherwise>
</xsl:choose>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
        <body>
            <h2>Colección de libros</h2>
            <table border="1">
                <tr bgcolor="#9acd32">
                    <th>Titulo</th>
                    <th>Autor</th>
                </tr>
                <xsl:for-each select="libreria/libro">
                    <tr>
                        <td><xsl:value-of select="titulo"/></td>
                        <xsl:choose>
                            <xsl:when test="precio > 10">
                                <td bgcolor="#ff0fff">
                                    <xsl:value-of select="autor"/></td>
                            </xsl:when>
                            <xsl:otherwise>
                                <td><xsl:value-of select="autor"/></td>
                            </xsl:otherwise>
                        </xsl:choose>
                    </tr>
                </xsl:for-each>
            </table>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

Elemento *<xsl:apply-templates>*

El elemento *<xsl:apply-templates>* aplica una plantilla al elemento actual o a los nodos secundarios del elemento actual.

Si añadimos un atributo *select* al elemento *<xsl:apply-templates>* procesará solamente el elemento secundario que coincida con el valor del atributo. Podemos usar el atributo *select* para especificar el orden en el que se procesan los nodos secundarios.

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

```
<xsl:template match="/">
  <html>
    <body>
      <h2>Colección de libros</h2>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
<xsl:template match="libro">
  <p>
    <xsl:apply-templates select="titulo"/>
    <xsl:apply-templates select="autor"/>
  </p>
</xsl:template>
<xsl:template match="titulo">
  Title: <span style="color:#ff0000">
    <xsl:value-of select=". "/></span>
    <br />
</xsl:template>
<xsl:template match="autor">
  Artist: <span style="color:#00ff00">
    <xsl:value-of select=". "/></span>
    <br />
</xsl:template>
</xsl:stylesheet>
```

Elementos XSLT

Si quieres aprender más elementos, funciones o atributos referentes a XSLT, puedes visitar este enlace. <https://developer.mozilla.org/es/docs/Web/XSLT/Elementos>

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

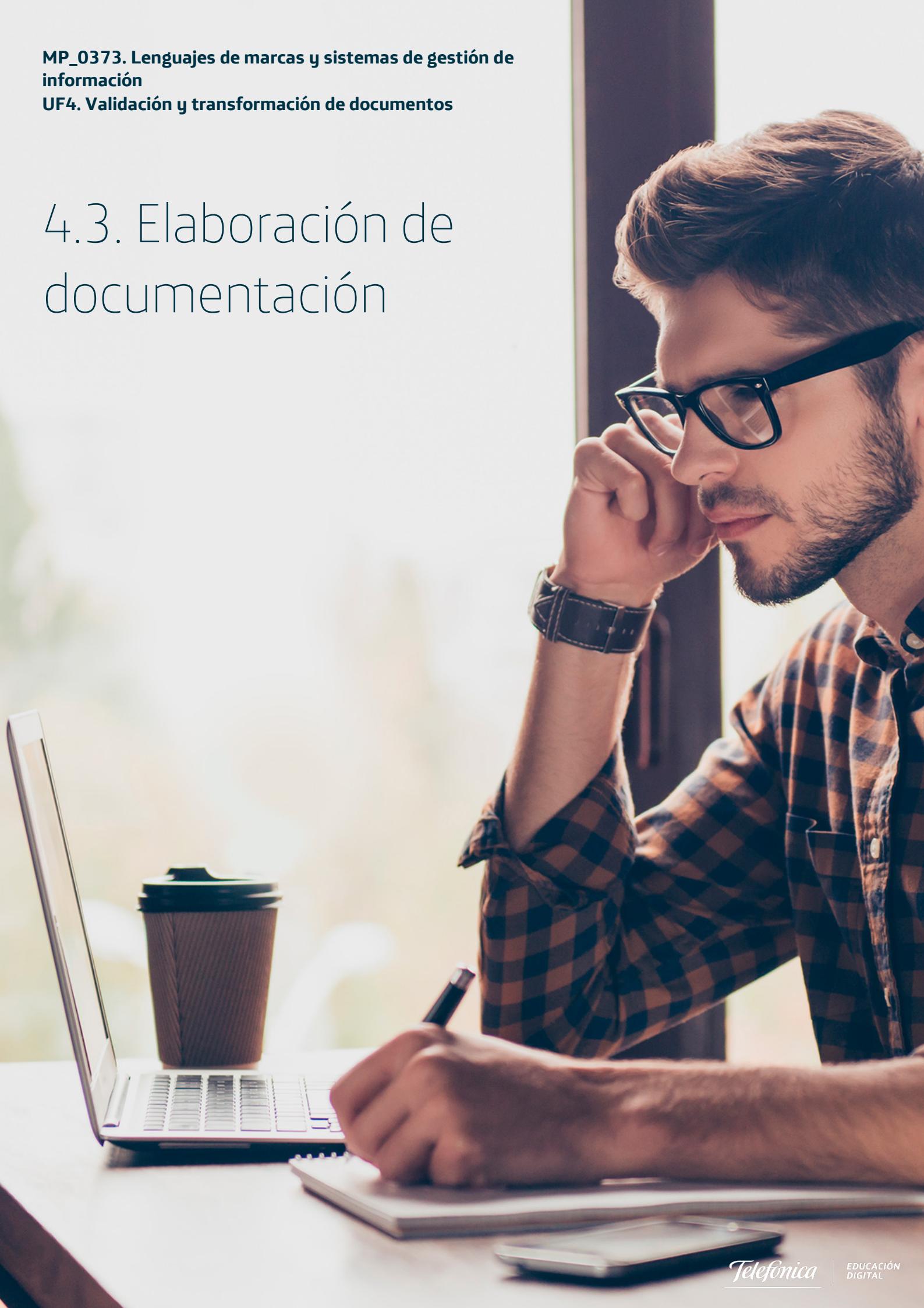
En esta lección hemos aprendido cómo los documentos XML definen datos cuando se usan como contenedores de estructuras complejas de datos. En su sintaxis no hay manera de definir el formato de salida de estos datos, por lo que se deben usar otros estándares para presentarlos formateados.

En ocasiones también es necesario cambiar el árbol de datos de un XML para transformarlo en otro que tenga los mismos datos pero con una estructura distinta.

XML permite crear documentos con datos pero sin un formato determinado. Mediante las transformaciones podemos obtener otros documentos que contienen partes del documento original. También podemos construir documentos con formato XHTML y cambiar los *schemas* de un XML a otro distinto. Estos nuevos documentos generados son el producto de las transformaciones.

Uno de los estándares de transformación de documentos XML en otros documentos XML o en formatos diferentes es XSL, que a través del lenguaje de transformación XSLT permite manejar la estructura de datos de los documentos entrantes para generar documentos con una estructura específica.

4.3. Elaboración de documentación



Índice

Objetivos	3
Introducción.....	4
XML para elaborar documentación	4
Documentación en la realización de software.....	4
Base de datos y plantillas XSL	4
Transformación a documentos	5
Generación de noticias	5
Formatting objects XSL-FO	7
XSL-FO.....	7
Estructura.....	7
Ejemplo de página inicial XSL-FO.....	8
Jerarquía de elementos XSL-FO.....	9
Páginas.....	10
Secuencias de páginas	11
Tipos de contenido	11
Cajas.....	12
Contenedor.....	12
Bloque.....	12
Línea.....	13
Inclusiones.....	13
Ejemplo.....	14
Imágenes	15
Listas	15
Enlaces	16
Despedida	17
Documentación y ejemplos.....	17
Resumen.....	17

Objetivos

En esta lección perseguimos los siguientes objetivos:

- Conocer los usos para la generación de documentación en base a XML.
- Conocer el lenguaje Formatting Objects XSL-FO.
- Manejar la sintaxis de XSL-FO.

Introducción

XML para elaborar documentación

El uso de XML para elaborar documentos es una forma de creación de documentación bastante estandarizada en muchos contextos y entidades.

Los escenarios más comunes en este tipo de configuraciones son:

- Generación de documentación en la **realización de software**.
- Generación de documentos con origen en **base de datos y plantillas XSL**.
- Transformación a **documentos electrónicos** en base a plantillas.
- Generación de noticias en base a **XMLNews**.

Documentación en la realización de software

En muchos sistemas de generación de código fuente se permite trasvasar la información del proyecto y de dicho código a formato XML estandarizado, de forma que la documentación de dicho código siga unas estructuras prefijadas.

Las aplicaciones **actualmente necesitan de mucha documentación**, explicando los procesos que realizan o sus métodos. Para ello los entornos de desarrollo disponen de herramientas que facilitan la creación de estos textos de forma ordenada. Normalmente usan los comentarios escritos por los desarrolladores, que con anotaciones especiales generan esta documentación.

Un ejemplo es la creación de documentación de aplicaciones como Javadoc.

Uso de JavaDoc

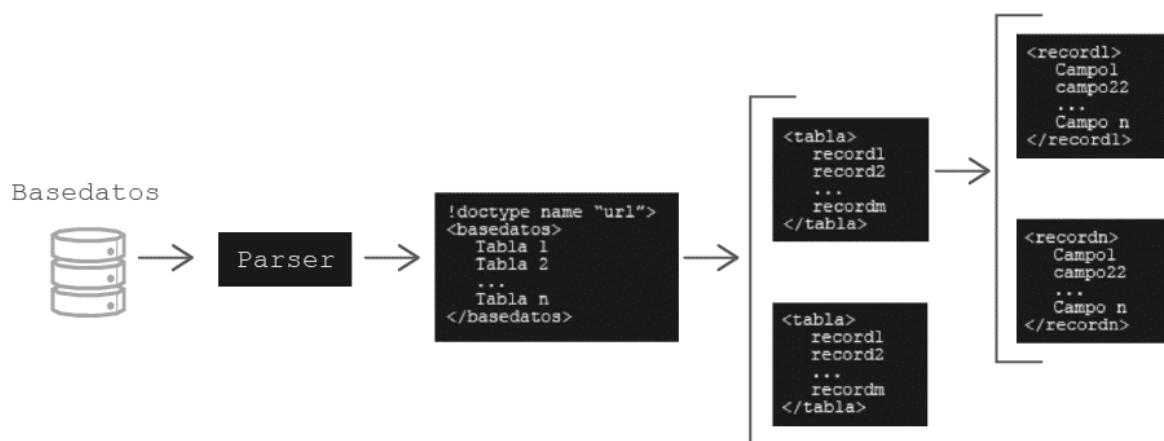
Generador de documentación con Java.

Base de datos y plantillas XSL

La técnica para el **traspaso desde un modelo relacional**se basará en el desarrollo de un *parser*, que transformará cada registro de la base de datos en un nodo de un elemento XML.

Una **base de datos consiste en un conjunto de tablas**, que a su vez constan de registros, que a su vez se componen de campos.

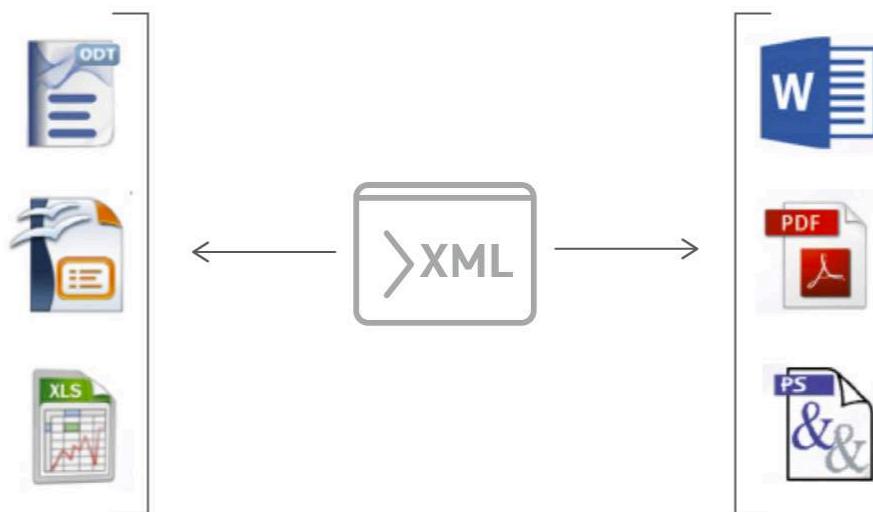
Podemos modelar la base de datos con un nodo de documento y sus nodos de elementos, que serán el reflejo de cada registro y un conjunto de campos para cada registro.



Transformación a documentos

El uso de plantillas en las transformaciones desde documentos XML hacia otros formatos de documentos permite traspasar la información en formatos de lectura o impresión.

El uso de **XSL-FO** permite obtener documentos imprimibles y con un formato legible amigable.



Generación de noticias

XMLNews es una especificación para el intercambio de noticias y otra información.

El uso de este estándar facilita, tanto a los productores de noticias como a los consumidores de noticias, producir, recibir y archivar cualquier tipo de información de noticias a través de diferente hardware, software y lenguajes de programación.

```
<?xml version="1.0" encoding="UTF-8"?>
<nitf>
  <head>
    <title>Colombia Earthquake</title>
  </head>
  <body>
    <headline>
      <h1>143 Dead in Colombia Earthquake</h1>
    </headline>
    <byline>
      <bytag>By Jared Kotler, Associated Press Writer</bytag>
    </byline>
    <dateline>
      <location>Bogota, Colombia</location>
      <date>Monday January 25 1999 7:28 ET</date>
    </dateline>
  </body>
</nitf>
```

Formatting objects XSL-FO

XSL-FO

Formatting Objects (FO) funciona de manera que, aplicado a un fichero XML, nos proporciona otro fichero con la información de este renderizada a algún formato de visualización impresa.

Sus principales características son:

- Es un **lenguaje de marcado de documentos XML** que es usado para generar archivos PDF u otros formatos de impresión y visualización.
- Es **parte de un conjunto de tecnologías W3C** diseñadas para la transformación de datos XML hacia otros formatos.
- La representación de datos a través de ficheros XML permite que puedan ser trasladados a otros formatos como el HTML, WML, etc.
- La principal diferencia entre un FO y otros XSLT es que **FO está pensado para generar directamente una renderización**, aunque ambos transformen documentos XML en otros documentos.
- Los formatos destino más conocidos pueden ser **PDF, PostScript, texto, word, etc.**
- XSL-FO, formalmente llamado XSL, se convirtió en una recomendación del W3C el 15 de octubre de 2001. Las otras partes de XSL son XSLT y XPath. La versión 1.1 de XSL-FO fue publicada en 2006.

Estructura

Los ficheros FO comienzan como un documento XML, con su declaración clásica.

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

El espacio de nombres se define de esta forma:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns:fo="http://www.w3.org/XSL/Format/1.0"
  result-ns="fo">
```

Las propiedades especifican cómo será la renderización.

```
<fo:block font-family="Times, Times New Roman">...</fo:block>
```

Esta es la forma de indicar los tipos de letra que deseamos que tenga el bloque (se comienza por la primera y se continúa hasta encontrar una disponible), siendo *font-family* una propiedad bien conocida por los usuarios de CSS.

Resumen de objetos de formato XSL-FO

Resumen del contenido y las propiedades de XSL Formatting Objects.

Ejemplo de página inicial XSL-FO

En este ejemplo podemos ver los elementos principales de un documento XSL-FO, vamos a comentar algunas de sus declaraciones.

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
        <fo:simple-page-master master-name="inicio">
            <fo:region-body/>
        </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="inicio">
        <fo:flow flow-name="xsl-region-body">
            <fo:block>Hola Mundo!!</fo:block>
        </fo:flow>
    </fo:page-sequence>
</fo:root>
```

El elemento *<fo: root>* es el elemento raíz; declara el espacio de nombres para XSL-FO.

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <!-- El contenido del documento va en este nivel-->
</fo:root>
```

El elemento *<fo: layout-master-set>* contiene una o más plantillas de página.

```
<fo:layout-master-set>
    <!-- Las plantillas que se usen irán a este nivel. -->
</fo:layout-master-set>
```

Cada elemento *<fo: simple-page-master>* contiene una plantilla de página única. Cada plantilla debe tener un nombre único (*master-name*).

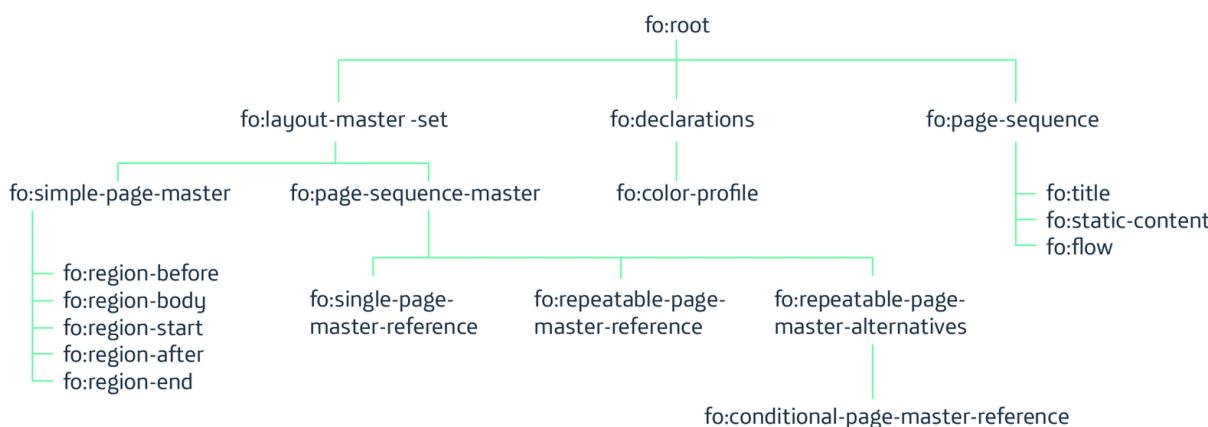
```
<fo:simple-page-master master-name="plantilla4">
    <!-- Cada plantilla única va a este nivel -->
</fo:simple-page-master>
```

Uno o más elementos *<fo: page-sequence>* describen el contenido de la página. El atributo *master-reference* se refiere a la plantilla de página simple con el mismo nombre.

```
<fo:page-sequence master-reference=" plantilla4 ">
    <!-- La página de contenidos va a este nivel-->
</fo:page-sequence>
```

Jerarquía de elementos XSL-FO

Veamos la jerarquía de los elementos en un documento XSL-FO.



`fo:root`

El elemento `<fo: root>` es el elemento raíz y también declara el espacio de nombres para XSL-FO.

`<fo:layout-master-set>`

Define la geometría y secuencia de las páginas. Sus descendientes son:

- `<fo:page-masters>`. Describen las subdivisiones de la página y su geometría.
- `<fo:page-sequence-masters>`. Describen la secuencia en la que un conjunto de páginas maestras serán usadas para generar las páginas tras aplicar el formato. La página maestra de la secuencia interna puede especificarse mediante una referencia a una *single-page-master* o como una repetición de una o más *page-masters*:
 - `<fo:single-page-master-reference>`. Secuencia que consiste en una *single-page-master*.
 - `<fo:repeatable-page-master-reference>`. Especifica la repetición de una *single-page-master*.
 - `<fo:repeatable-page-master-alternatives>`. Especifica la repetición de un conjunto de *page-masters*.

`<fo:page-sequences>`

Para cada una solo es utilizada una secuencia de páginas, ya definidas en el `<fo:page-sequence-masters>` que satisfaga las condiciones dadas en el `page-masters`.

Sus descendientes son:

- `<fo:flows>` y `<fo:static-content>` que proporcionan el contenido que se pondrá en las páginas.

`fo:simple-page-master`

Cada elemento `<fo: simple-page-master>` contiene una plantilla de página única. Cada plantilla debe tener un nombre único (*master-name*).

<fo: page-sequence>

Uno o más elementos <fo: page-sequence> describen el contenido de la página. El atributo *master-reference* se refiere a la plantilla de página simple con el mismo nombre.

<fo: region-body>

Establece cómo será la apariencia de la región en la que se encuentra el contenido actual de la página.

Páginas

Las páginas se representan con <fo:simple-page-master>, las cuales son hijas de un <fo:layout-master-set>.

Para definir el aspecto de cada una de las páginas tenemos que indicar los tamaños de las regiones anterior, posterior, cuerpo, inicio y fin. Las etiquetas que se usan respectivamente son:

- <fo:region-before> - Región anterior.
- <fo:region-after> - Región posterior.
- <fo:region-body> - Cuerpo.
- <fo:region-start> - Inicio.
- <fo:region-end> - Fin.

```
<fo:simple-page-master master-name="HelloWorld">
  <fo:region-before extent="20mm"/>
  <fo:region-after extent="20mm"/>
  <fo:region-body margin-top="10mm" margin-bottom="10mm"/>
</fo:simple-page-master>
```

Ejemplo de regiones de impresión que definen áreas en la página.

Estas, a su vez, pueden ser llenadas de contenido usando <fo:flow> o <fo:static-content>.

La página es la unidad de impresión y tiene las siguientes áreas:

- Ancho de la página: *page-width*.
- Alto de la página: *page-height*.
- Márgenes: opcional, que se subdividen en cuatro regiones:
 - Arriba: *margin-top*.
 - Abajo: *margin-bottom*.
 - Izquierda: *margin-left*.
 - Derecha: *margin-right*.

```
<fo:simple-page-master master-name="HelloWorld"
  page-height="297mm"
  page-width="210mm"
  margin-left="12mm"
  margin-right="12mm"
  margin-top="20mm"
  margin-bottom="20mm">
```

Ejemplo.

Secuencias de páginas

La secuencia de páginas es el segundo hijo de `<fo:layout-master-set>` y se denomina con la marca `<fo:page-sequence-master>`.

El elemento `<fo:page-sequence-master>` solo contiene un atributo, que determina el nombre de referencia (*master-name*). Las secuencias de página se definen por los elementos incluidos:

- `<fo:repeatable-page-master-reference>`.
- `<fo:repetible-page-master-alternatives>`.
- `<fo:page-master-reference>`.

```
<xsl:template match="/">
  <fo:root>
    <fo:layout-master-set>
      <fo:simple-page-master master-name="PageMaster.Content-right"
      ....
      </fo:simple-page-master>
      <fo:page-sequence-master master-name="Part-Pages">
        </fo:page-sequence-master>
        <fo:page-sequence-master master-name="Content-Pages">
          ....
          </fo:page-sequence-master>
    ...
  ...
```

Secuencia de páginas complejas

Ejemplo de aplicación de secuencia de páginas.

Tipos de contenido

Existen dos tipos de contenido que podemos insertar; **dinámico y estático**.

Contenido dinámico (*flows*)

`<fo:flow>` inserta contenido dinámico en el documento. Puede contener otros elementos como `<fo:block>`, `<fo:display-graphic>`, `<fo:display-link>`, `<fo:display-rule>` y otros elementos colocados al mismo nivel del elemento.

`<fo:flow>` contiene el atributo llamado *flow-name*, cuyos valores pueden ser: *xsl-body*, *xsl-after*, *xsl-before*, *xsl-start* y *xsl-end*, e indican en qué lugar de la página se coloca el contenido.

```
<fo:flow flow-name="region-after">
  <fo:block>Creado por Pepe Illo</fo:block>
</fo:flow>
```

Ejemplo `fo:flow`.

Contenido estático (*static content*)

<**fo:static-content**> incluye contenido estático en las páginas e internamente tiene los mismos contenidos que <**fo:flow**>. Cuando se usan los contenidos estáticos, **deben aparecer antes de los elementos dinámicos** al definir la secuencia de elementos de la página.

```
<fo:static-content flow-name="region-after">
  <fo:block>(C) Telefónica Educación Digital 2018</fo:block>
</fo:static-content>
```

Ejemplo *fo:static-content*.

Cajas

XSL-FO utiliza **cajas rectangulares (áreas)** para mostrar la salida.

Estos elementos están divididos formalmente en **cuatro tipos de áreas rectangulares**: contenedores, bloques, líneas e "inclusiones" (*inline-areas*).

Contenedor

Es la estructura de más alto nivel. Usando coordenadas se colocará en el área que lo contiene. Los contenedores pueden anidarse como secuencias de bloques y de espacios a visualizar.

Por ejemplo, una página de un texto o documento funcionará como un contenedor que incluye otros cinco contenedores: la cabecera, el cuerpo, el pie y los márgenes derecho e izquierdo.

Los FO que producen contenedores son: ***fo:region-body*, *fo:region-before*, *fo:region-after*, *fo:region-start* y *fo:region-end***.

Bloque

Sirve para crear una estructura bloque, que se podría asemejar a un párrafo o un apartado de una lista. Los bloques pueden ser incluidos de forma secuencial dentro del contenedor que los incluye. Los elementos que pueden incluirse dentro de un bloque son: líneas, espacios a visualizar, una imagen y otros bloques (incluir ruptura de línea).

Los FO que producen bloques son: ***fo:block*, *fo:display-graphic*, *fo:display-link*, *fo:display-rule* y *fo:list-block***.

```
<fo:block>
  <fo:inline font-style="italic">
    Esto está en letra italica
  </fo:inline>
  Página: <fo:page-number/>
  <fo:inline>
    ¡Esto no lo está!
  </fo:inline>
</fo:block>
```

Los **<fo:block>** pueden estar contenidos en **<fo:flow>**, **<fo:static-content>** u otros **<fo:block>**. Ellos pueden contener a otros **<fo:block>**, así como **<fo:display-graphic>** y **<fo:display-rule>** (ambos a nivel de bloque), y también **<fo:inline>** y **<fo:page-number>** (a nivel de inclusiones). También pueden tener texto plano.

Línea

Son las líneas de texto dentro de un bloque. Pueden contener inclusiones, así como inclusiones de espacios (*inline areas* e *inline spaces*, respectivamente). No existen FO que se correspondan con las líneas, sino que el motor de evaluación del formateo va generándolas conforme divide líneas debido, por ejemplo, al tipo de justificación elegida.

Inclusiones

Son partes de una línea, como por ejemplo cada carácter, la referencia a una nota al pie o una expresión matemática. Cada inclusión puede contener otras inclusiones o inclusiones de espacio.

Es un área rectangular que puede contener texto u otras inclusiones. Se corresponde con inclusiones:

- **fo:bidi-override.**
- **fo:character.**
- **fo:first-line-marker.**
- **fo:inline-graphic.**
- **fo:inline-include-container.**
- **fo:inline-rule.**
- **fo:inline-sequence.**
- **fo:list-item-body.**
- **fo:list-item-label.**
- **fo:page-number.**
- **fo:page-number-citation.**

Algunas se corresponden con la versión de sus compañeras a nivel de bloque, tales como **fo:inline-rule** o **fo:inline-graphic**. Otras ya las hemos utilizado previamente, como **fo:list-item-body**, **fo:list-item-label** y **fo:page-number**.

Ejemplo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- =====
      Hola.fo
      Joaquin Bravo Montero
      (c) Programación en Castellano
      http://www.programacion.net
      ===== -->

<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

  <fo:layout-master-set>

    <fo:simple-page-master master-name="simple"
      page-height="29.7cm"
      page-width="21cm"
      margin-top="1cm"
      margin-bottom="2cm"
      margin-left="2.5cm"
      margin-right="2.5cm">
      <fo:region-body margin-top="3cm"/>
      <fo:region-before extent="3cm"/>
      <fo:region-after extent="1.5cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-name="simple">

    <fo:flow flow-name="xsl-region-body">

      <fo:block font-size="18pt"
        font-family="sans-serif"
        line-height="24pt"
        space-after.optimum="15pt"
        text-align="center"
        padding-top="3pt">
        Mi primer XSL-FO
      </fo:block>

      <fo:block font-size="12pt"
        font-family="sans-serif"
        line-height="15pt"
        space-after.optimum="3pt"
        text-align="justify">
        Hola este es mi primer XSL-FO.
      </fo:block>

    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Siquieres ver más información de este ejemplo, puede ir a la fuente en: [Programación.net](http://www.programacion.net).

Imágenes

El contenido que normalmente vamos a incluir es de tipo texto, aunque también se pueden incluir imágenes, casi de la misma forma que en HTML se hace con ``.

En este ejemplo vemos cómo insertar un gráfico en un bloque. Las propiedades que incluye son: `href` (ubicación del fichero gráfico), `min-height`, `min-width`, `max-height`, `max-width`, `scale-to-fit`.

```
<fo:block>
  <fo:inline>
    <fo:external-graphic src="xml/Img/image1.bmp"/>
  </fo:inline>
  <fo:inline>
    <fo:external-graphic src="xml/Img/image1.bmp" content-width="26mm"/>
  </fo:inline>
  <fo:inline>
    <fo:external-graphic src="xml/Img/image1.bmp" content-width="25mm" content-
height="20mm"/>
  </fo:inline>
</fo:block>
```

Listas

Un `<fo:list-block>` es el bloque que genera listas, pueden contener bien una serie de `<fo:list-item>`, o bien pares de `<fo:list-item-label>` y `<fo:list-item-body>`, pero no ambas cosas a la vez.

Tienen tres propiedades de especial relevancia:

- ***provisional-label-separation***: se le dan tres valores de distancia separados por punto y coma, indicando valor máximo, mínimo y óptimo, respectivamente.
- ***provisional-distance-between-starts***: distancia desde el comienzo del eje de la etiqueta y el comienzo del eje del cuerpo del elemento.
- ***space-between-list-rows***: distancia entre los sucesivos elementos: máximo, mínimo y óptimo.

```
<fo:list-block>
  <fo:list-item>
    <fo:list-item-label>*</fo:list-item-label>
    <fo:list-item-body>Primer elemento de la lista</fo:list-item-label>
  </fo:list-item>
  <fo:list-item>
    <fo:list-item-label>*</fo:list-item-label>
    <fo:list-item-body>Segundo elemento de la lista</fo:list-item-label>
  </fo:list-item>
</fo:list-block>
```

Ejemplo de lista.

```
<fo:list-block>
  <fo:list-item-label>*</fo:list-item-label>
  <fo:list-item-body>Primer elemento de la lista</fo:list-item-label>
  <fo:list-item-label>*</fo:list-item-label>
  <fo:list-item-body>Segundo elemento de la lista</fo:list-item-label>
</fo:list-block>
```

Ejemplo de lista.

Enlaces

Un enlace tiene sentido en la versión electrónica de dicho documento. Para introducir enlaces se usa la etiqueta `<fo:simple-link>`, que puede actuar como un elemento de bloque o de inclusión, según su contenido.

Para controlar su comportamiento se le han asignado seis atributos: ***external-destination***, ***internal-destination***, ***indicate-destination***, ***show-destination***, ***space-above-destination-block*** y ***space-above-destination-start***.

- Un enlace a un **documento remoto** se especifica asignándole su *URI* a *external-destination*.
- Un enlace a un **recurso del propio documento** se indica asignándole el valor del identificador de dicho recurso a la propiedad *internal-destination*.
- Un enlace **no puede tener estos dos atributos a la vez**.

`<fo:show-destination>` acepta los valores *replace* (por defecto) o *new*, que permite cargar el enlace indicado en la misma ventana o en otra distinta.

Por último, `<fo:space-above-destination-start>` y `<fo:space-above-destination-block>` permiten indicar si cuando se cargue el documento destino se va a poner el punto exacto indicado por el enlace en la parte superior del navegador, o por el contrario se va a desplazar hacia abajo (no dejando espacio vacío, sino con el contenido que lo preceda).

Despedida

Documentación y ejemplos

La elaboración de documentos con XSL:FO, como has podido comprobar, tiene una gran cantidad de opciones y puede ser una herramienta muy poderosa para incorporar en nuestras aplicaciones.

A continuación encontrarás enlaces con ejemplos y documentación para que profundices en el tema.

Ejemplos XSL-FO

Página con un listado de ejemplos, para ver el código y el resultado.

Más ejemplos de aplicación con XSL-FO

Página de archivo con ejemplos de uso de XSL-FO.

Manual XSL-FO

Manual en inglés de XSL-FO.

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

XML es un estándar muy abierto que permite almacenar información de cualquier tipo y con una estructura en árbol. En muchas tecnologías el uso de documentos XML, tanto para la salida como para la entrada de datos, es de uso común y está estandarizado.

Dentro de XSL se encuentra incluido el estándar para poder generar documentos electrónicos imprimibles o electrónicos con formato legible. Está aceptado por la W3C y es el XSL-FO. En este tema hemos visto su sintaxis y estructura.

4.4. Utilización de herramientas de procesamiento: DOM y SAX



Índice

Objetivos	3
Tratamiento de XML desde JAVA	4
SAX.....	5
Definición.....	5
Características	6
Funcionamiento.....	6
DOM.....	8
Definición.....	8
Características	9
Funcionamiento.....	10
Despedida	14
Resumen.....	14

Objetivos

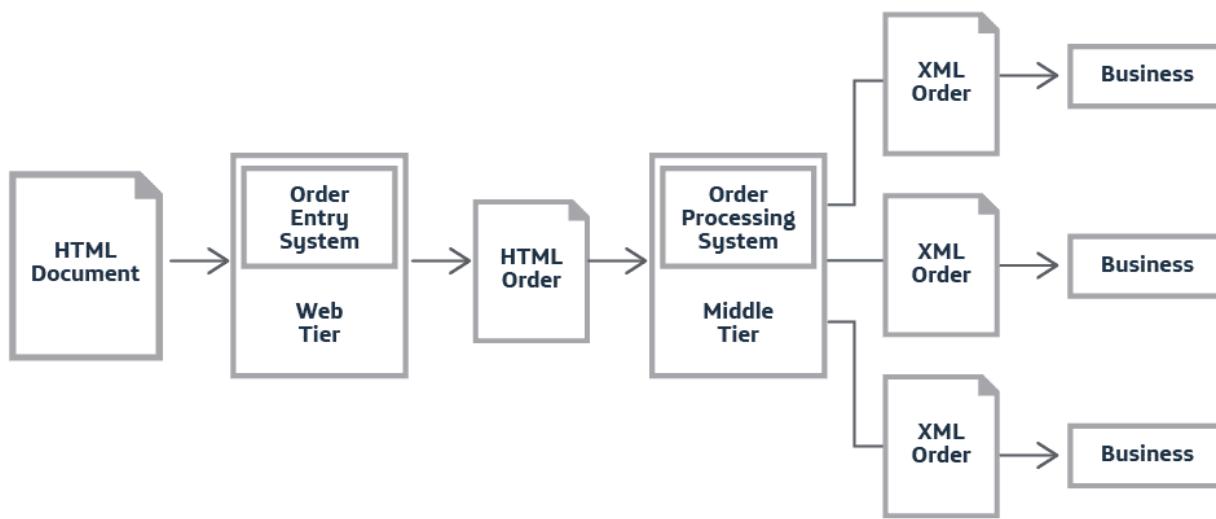
En esta lección perseguimos los siguientes objetivos:

- Conocer cómo se puede acceder y manejar los datos insertados en documentos de marcas.
- Conocer el estándar SAX, su definición, funcionamiento, características y ventajas.
- Conocer el estándar DOM, su definición, funcionamiento, características y ventajas.

Tratamiento de XML desde JAVA

Ya hemos visto que XML es una solución para el intercambio de datos entre distintas plataformas. Es por ello que podemos utilizar XML para almacenar y publicar datos de una forma estructurada.

En determinadas ocasiones puede que sea necesario procesar, validar y transformar documentos XML.



El lenguaje Java proporciona librerías para procesar documentos XML. El conjunto de estas librerías se conoce como el API JAXP.

Actualmente existen tres modelos para llevar a cabo dicho procesamiento:

- Simple API for XML (SAX).
- Document Object Model (DOM).
- Streaming API for XML (StAX).

SAX

Definición

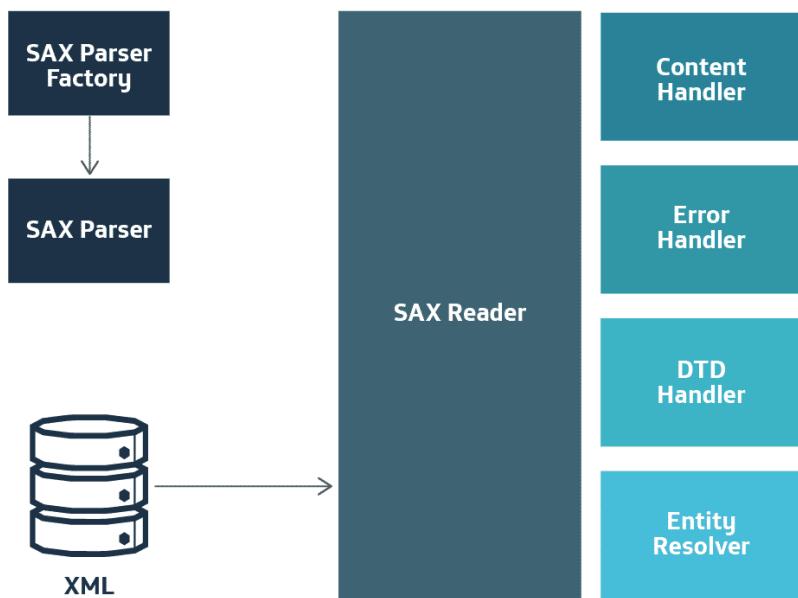
SAX, API para XML, es un estándar público desarrollado para el análisis de documentos XML basado en eventos.

SAX define una interfaz abstracta mediante programación que modela el conjunto de información XML (*infoset*) a través de una secuencia lineal de llamadas a métodos. Por lo tanto podemos indicar que:

- La API SAX es una API orientada a eventos.
- A diferencia del DOM, no conlleva la generación de estructuras internas.
- La mayoría de los procesadores soportan esta API.
- La programación de SAX se realiza en Java.

SAX está definido en los siguientes paquetes:

- *org.xml.sax*.
- *org.xml.sax.ext*.
- *org.xml.sax.helpers*.
- *javax.xml.parsers*.



Características

Las características revisadas de este estándar indican que SAX es el más apropiado para la lectura y validación de documentos en los que solo sea necesario procesarlo una sola vez, ya que SAX no crea un modelo y lo mantiene en memoria para su posterior uso.

El analizador que se use **debe leer el fichero XML secuencialmente**, cada vez que reconoce una etiqueta o nodo lo notifica a la aplicación que lo está ejecutando a través de la llamada a un método de la interfaz.

- El procesamiento se realiza como un *stream* de eventos.
- Los eventos fuerzan la ejecución de métodos de la interfaz, que el programador debe implementar en sus clases.
- Los eventos se producen de forma anidada, como los métodos que se invocan.
- SAX no genera una estructura en memoria del documento, por lo que es más eficiente que DOM.
- Permite la validación de un documento XML.

En el caso de tener archivos de gran tamaño es bastante eficiente, pues lee el documento sin ocupar gran cantidad de memoria.

SAX sirve por lo tanto para labores de validación de documentos XML.

Funcionamiento

SAX lee el documento secuencialmente de principio a fin, sin cargarlo en memoria, de forma que cuando encuentra un elemento se encarga de lanzar su evento asociado, que producirá la llamada a un método de la interfaz.

Cuando el evento es lanzado, puede ser capturado para realizar una función determinada. Esta API está definida en el paquete: *javax.xml.parsers*.



Para poder capturar estos eventos y programar las acciones correspondientes de análisis y otras operaciones que se deseen, se debe implementar un manejador de eventos en el lenguaje de programación del sistema de proceso que use el XML.

Un **manejador es una clase con una serie de métodos** y cada método se ejecutará cuando el analizador capture su evento asociado. **Estos eventos se producen al leer un documento** (al

comienzo del documento, apertura o cierre de un elemento, al encontrar una instrucción de proceso o un comentario, etc.).

Los métodos utilizados por SAX para leer un documento son:

- *startDocument*.
- *endDocument*.
- *startElement*.
- *endElement*.
- *characters*.

```
package xmlSax;

import java.util.Arrays;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class EjemploSax extends DefaultHandler{

    public static void main(String[] args) throws Exception{
        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser saxParser = factory.newSAXParser();
        saxParser.parse("alumno.xml", new EjemploSax());
    }

    @Override
    public void startDocument() throws SAXException {
        System.out.println("Empezamos a leer empleados.xml");
    }

    @Override
    public void endDocument() throws SAXException {
        System.out.println("Terminamos de leer empleados.xml");
    }

    @Override
    public void startElement(String uri, String localName, String qName,
                           Attributes at) throws SAXException {
        System.out.println("Procesando " + qName);
    }

    @Override
    public void endElement(String uri, String localName, String qName)
                           throws SAXException {
        System.out.println("Fin " + qName);
    }

    @Override
    public void characters(char[] ch, int start, int length)
                           throws SAXException {
        String contenido = new String(ch, start, length);
        System.out.println(contenido);
    }
}
```

Ejemplo de lectura de un documento XML desde SAX en Java.

DOM

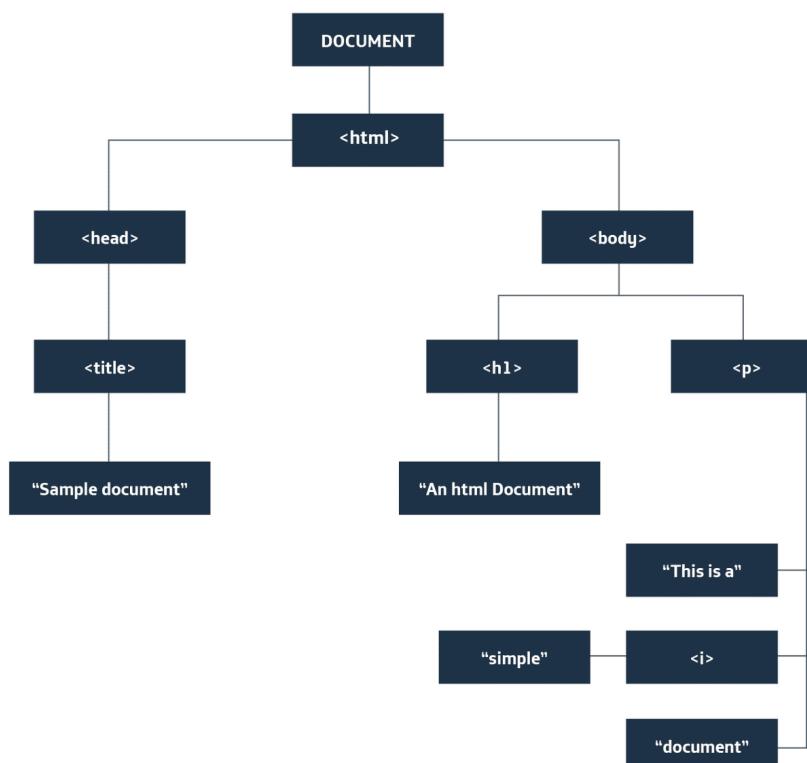
Definición

Al procesar un documento XML a través de DOM se genera lo que es denominado árbol jerárquico en memoria. Este árbol contiene toda la información del fragmento/documento XML en cuestión. Cada elemento del documento pasa a ser un nodo del árbol, por lo que en DOM todo es un nodo.

La API de "**Document Object Model**" (**DOM**) es un estándar que se define a través de un conjunto de interfaces que describen una estructura abstracta para documentos XML y HTML. **DOM crea el documento XML o HTML entero** en memoria con una estructura tipo árbol. Cada elemento del documento XML o HTML se representa con un nodo (*DOMNode*).

En la imagen siguiente se puede visualizar la estructura generada por un documento HTML. DOM está definido en los paquetes Java *org.w3c.dom, javax.xml.transform.dom y javax.xml.parsers*.

El árbol jerárquico de información en memoria permite que a través del manejador pueda manipularse la información: crear o eliminar información de un nodo en cualquier punto del árbol, acceder o cambiar su contenido y mover la herencia de los nodos.



Árbol DOM.

DOM de W3C utiliza una referencia doble para referirse a los objetos del documento. Todo lo que hay en el modelo de objeto es un:

- Nodo.
- Tipo de objeto con nombre.

La implementación del DOM tiene varios niveles. El W3C ha establecido tres niveles de soporte DOM:

Nivel 0

Se utiliza para manipular HTML.

Nivel 1

Permite acceder a todas las partes del cuerpo de un documento XML. No permite acceder a las DTD. Es la recomendación actual.

Nivel 2

Además de las capacidades del nivel 1, permite acceder a DTD, hojas de estilo y espacios de nombres. Aún está en desarrollo.

La principal ventaja de este mecanismo es que toda la información XML se encuentra en memoria. Esta residencia en memoria hace que los datos puedan ser navegables por el programa DOM, permitiendo regresar y manipular información ya procesada, lo que no es posible con SAX.

Por extensión, **DOM también se usa para la manipulación de documentos XHTML y HTML**. La estructura de información de un DOM es a través de nodos, de forma que:

- El documento en sí es un nodo.
- Todos los elementos XML o HTML son elementos nodos.
- Todos los atributos XML o HTML son atributos nodos.
- El texto dentro de los elementos XML o HTML son textos nodos.
- Los comentarios son comentarios nodos.

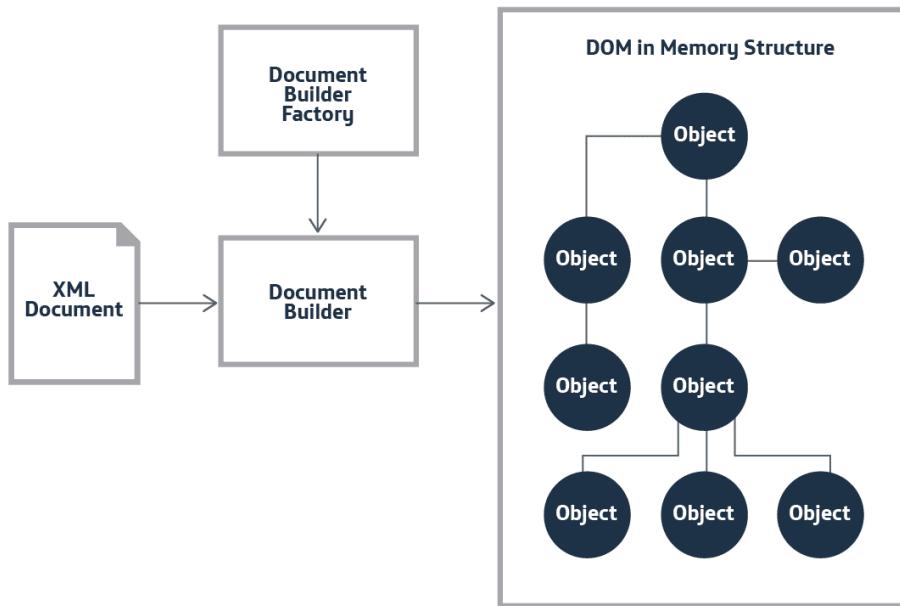
Características

DOM posee las siguientes características:

- Genera una estructura de árbol en memoria al procesar el documento.
- Se puede usar para generar documentos XML.
- Es recurso intensivo.
- Requiere más tiempo y recursos para procesar documentos.
- No es recomendable para grandes documentos.

DOM está definido en los siguientes paquetes:

- org.w3c.dom.
- javax.xml.parsers.



Arquitectura DOM.

Funcionamiento

Para los programadores, trabajar con DOM les permite disponer de un control muy preciso sobre toda la estructura del documento XML.

La API de DOM proporciona funciones que nos permiten añadir, eliminar, modificar y reemplazar cualquier nodo de cualquier documento de forma sencilla. Los objetos del DOM tienen propiedades y la API nos ofrece métodos para poder interactuar con esos elementos.

Los tipos de nodos en el árbol son:

- **Document:** el nodo raíz de todos los documentos HTML y XML. Todos los demás nodos cuelgan del árbol que genera al cargarse en el navegador.
- **DocumentType:** contiene la representación DTD que se ha utilizado en la página (mediante el DOCTYPE).
- **Element:** representa a un elemento del lenguaje de guion, en el caso de HTML serían los elementos propios que se crean con una etiqueta de apertura y otra de cierre. Es el único nodo que puede tener tanto nodos hijos como atributos.
- **Attr:** representa el atributo y su valor, propios de los elementos.
- **Text:** almacena el contenido del texto incluido en un elemento.
- **CDataSection:** es el nodo que representa una sección de tipo `<![CDATA[]]>`.
- **Comment:** representa un comentario.

Necesitamos conocer los métodos que nos permiten navegar entre nodos, conocer las propiedades del nodo y demás. Para ello, dependiendo del **nivel de DOM** al que tengamos acceso a través de su implementación en el navegador, podremos utilizarlos.

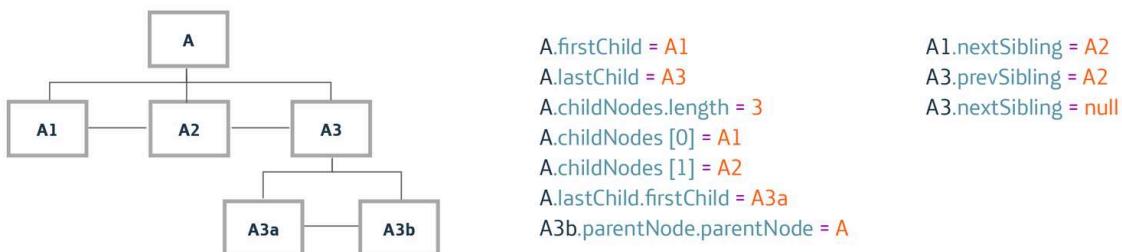
Incluimos un listado de algunos métodos y atributos que nos proporciona el estándar:

Nivel de documento

- `createElement`: crea un elemento nuevo.
- `createTextNode`: crea un nodo de texto.
- `createComment`: crea un nodo comentario.
- `createAttribute`: crea un nodo atributo.
- `getElementsByTagName`: devuelve un listado de nodos con un `tagname` en orden.

Nivel de nodo

- `get` y `set` de `NodeName`: nombre del nodo.
- `get` y `set` de `nodeValue`: valor del nodo.
- `get` y `set` de `NodeType`: tipo del nodo.
- `get` y `set` de `ParentNode`: padre del nodo actual.
- `get` y `set` de `ChildNodes`: una lista de nodos hijo.
- `get` y `set` de `FirstChild`: primer hijo del nodo.
- `get` y `set` de `FastChild`: último hijo del nodo.
- `get` y `set` de `PreviousSibling`: nodo precedente.
- `get` y `set` de `NextSibling`: nodo siguiente.
- `get` y `set` de `Attributes`: lista de atributos del nodo.



Métodos de jerarquía DOM.

Ejemplo

Veamos un ejemplo de cómo crear un archivo XML y convertirlo en una clase de Java.

```
package xmlDom;

import java.io.File;
import java.io.FileWriter;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
```

```
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Text;

public class Principal {

    public static void main(String[] args) throws Exception{

        String nombreArchivo = "probando.xml";
        try{

            File archivo = new File(nombreArchivo);
            FileWriter escribir = new FileWriter(archivo);
            String cabecera = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<empleados></empleados>";

            escribir.write(cabecera);
            escribir.close();
            }catch (Exception e) {
                System.out.println("Error al crear archivo");
            }

        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document document = db.parse(nombreArchivo);

        Element raiz = document.getDocumentElement();

        Element newEmp = document.createElement("empleado");

        Element newId = document.createElement("id");
        Text IDText = document.createTextNode("00133");
        newId.appendChild(IDText);

        Element newNombre = document.createElement("nombre");
        Text nombreText = document.createTextNode("Luis Sanchez");
        newNombre.appendChild(nombreText);

        Element newDir = document.createElement("direccion");

        Element newCalle = document.createElement("calle");
        Text calleText = document.createTextNode("Castellana");
        newCalle.appendChild(calleText);

        Element newCiudad = document.createElement("ciudad");
        Text ciudadText = document.createTextNode("Madrid");
        newCiudad.appendChild(ciudadText);

        Element newCp = document.createElement("codigo_postal");
        Text cpText = document.createTextNode("28027");
        newCp.appendChild(cpText);

        newDir.appendChild(newCalle);
        newDir.appendChild(newCiudad);
        newDir.appendChild(newCp);
    }
}
```

```
Element newEmail = document.createElement("email");
Text emailText = document.createTextNode("luis@yahoo.es");
newEmail.appendChild(emailText);

Element newDpto = document.createElement("dpto");
Text dptoText = document.createTextNode("Compras");
newDpto.appendChild(dptoText);

Element newSueldo = document.createElement("sueldo");
Text sueldoText = document.createTextNode("60000");
newSueldo.appendChild(sueldoText);

newEmp.appendChild(newId);
newEmp.appendChild(newNombre);
newEmp.appendChild(newDir);
newEmp.appendChild(newEmail);
newEmp.appendChild(newDpto);
newEmp.appendChild(newSueldo);

raiz.appendChild(newEmp);

TransformerFactory tf = TransformerFactory.newInstance();
Transformer transformer = tf.newTransformer();

DOMSource source = new DOMSource(document);
StreamResult resultado = new StreamResult(nombreArchivo);

transformer.transform(source, resultado);

}

}
```

Código de la clase para la creación de un XML mediante DOM en Java.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<empleados>
    <empleado>
        <id>00133</id>
        <nombre>Luis Sanchez</nombre>
        <direccion>
            <calle>Castellana</calle>
            <ciudad>Madrid</ciudad>
            <codigo_postal>28027</codigo_postal>
        </direccion>
        <email>luis@yahoo.es</email>
        <dpto>Compras</dpto>
        <sueldo>60000</sueldo>
    </empleado>
</empleados>
```

El XML resultante.

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado. El análisis y modificación de documentos de marcas como XML y HTML es necesario para que los sistemas de procesamiento que los usan puedan manejarlos y entender sus datos.

Muchos *frameworks* usan documentos XML para emitir directivas de proceso o definir su propia configuración. Poder entender y analizar estos documentos es vital para el funcionamiento de dichos *frameworks*.

En este tema hemos visto los conceptos más básicos de cómo **trabajar con documentos XML utilizando las API de SAX y DOM**.

Las **ventajas** de procesar documentos XML **utilizando el modelo SAX son:**

- Permite procesar documentos de cualquier tamaño.
- Es útil cuando solo queremos procesar parte del documento.
- Es muy simple y rápido.
- No consume mucha memoria ni necesita mucho procesador.

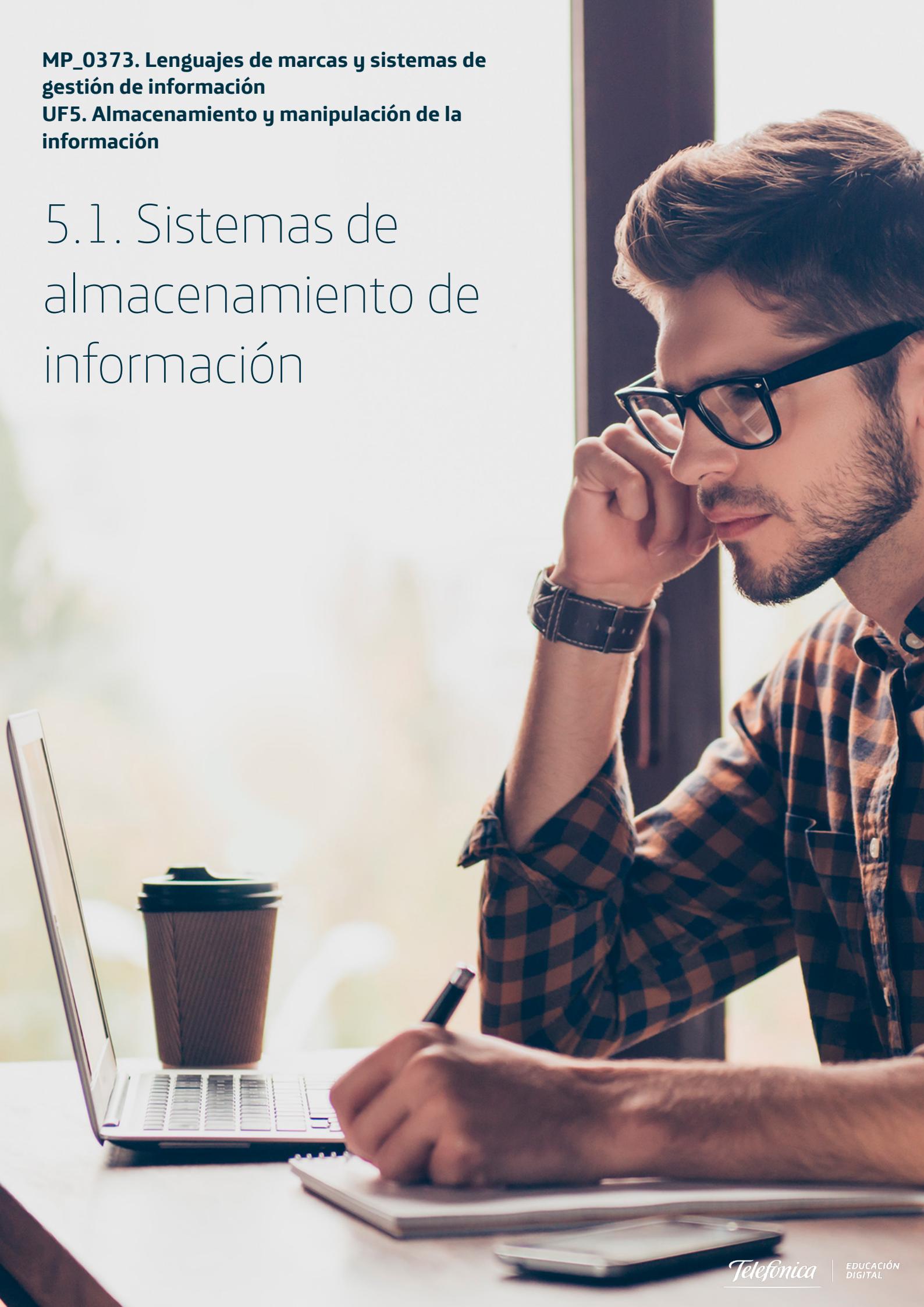
Las **ventajas** del uso de **DOM** son:

- Puede crear elementos e insertarlos en el árbol.
- Puede modificar los valores de los elementos.
- Si se necesita, se pueden realizar múltiples procesados, ya que el árbol está en memoria.
- Una vez analizado y validado un documento se evita tener que volver a analizar el documento.
- Dependiendo del procesador utilizado, se puede acceder al DOM utilizando C++, Java, JavaScript y VBScript.

MP_0373. Lenguajes de marcas y sistemas de gestión de información

UF5. Almacenamiento y manipulación de la información

5.1. Sistemas de almacenamiento de información



Índice

Objetivos	3
Conceptos básicos	4
Bases de datos relacionales.....	4
Bases de datos no relacionales	4
Ventajas y desventajas del uso de XML	5
Aplicación de XML	5
Almacenamiento en XML.....	6
Tipos de bases de datos XML.....	6
Semejanzas entre una BD y XML	6
Bases de datos relacionales con XML	7
XML y bases de datos orientadas a objetos.....	8
Despedida	9
Resumen.....	9

Objetivos

En esta unidad perseguimos los siguientes objetivos:

- Saber cuáles son los **sistemas de almacenamiento y su finalidad**.
- Conocer las **características de los sistemas** que almacenan ficheros XML.
- Conocer la **aplicación de XML** en el almacenamiento de datos.
- Aprender los **diferentes formatos** de almacenamiento de XML.
- Saber cómo se adaptan las **bases de datos relacionales** a trabajar con XML.
- Conocer la relación entre **XML y las bases de datos orientadas a objetos**.

Características de XML

Conceptos básicos

Los sistemas de almacenamiento de datos se han adaptado a la entrada del estándar XML en la persistencia de los datos y su estructuración.

Este tipo de sistemas de almacenamiento trabajan con varios sistemas lógicos y físicos.

- En cuanto a los **sistemas físicos**, son conocidos y su arquitectura no afecta al aspecto, que es de lo que vamos a tratar en esta unidad.
- Sin embargo, sí debemos revisar los **sistemas lógicos**, porque dependiendo de cómo se organice la información y cómo se acceda a ella, comprobaremos que existen distintos tipos.

Bases de datos relacionales

Almacenar los documentos XML en bases de datos relacionales supone una gran ventaja, ya que se usan ampliamente en las aplicaciones existentes. Podemos almacenar datos XML en dichas bases de datos de forma que se pueda acceder a ellos de una forma relacional.

- **Almacenamiento como cadena.** Una forma sencilla es almacenar cada elemento hijo del elemento de mayor nivel como una cadena en una tupla (registro) separada de la base de datos. Esta es una arquitectura en la que la búsqueda es bastante complicada, ya que se debe recorrer todo el XML para encontrar un registro concreto.
- **Representación en árbol.** Los datos XML se pueden modelar como un árbol y almacenar mediante el uso de un par de relaciones. De este modo, a cada elemento y atributo se le proporciona un identificador único.
- **Asignación a relaciones.** Según este enfoque, los elementos XML cuyo esquema es conocido se asignan a relaciones y atributos.

Bases de datos no relacionales

Existen varias alternativas para almacenar datos XML en sistemas de almacenamientos de datos no relacionales:

- **Almacenamiento en archivos planos.** Puesto que XML es principalmente un formato de archivo, un mecanismo de almacenamiento natural es, simplemente, un archivo plano ordenado en el sistema.
- **Almacenamiento en una base de datos XML.** Las bases de datos XML usan XML como su modelo de datos básico. Las bases de datos XML antiguas implementaban el modelo de objetos documento sobre una base de datos orientada a objetos basada en C++.

Ventajas y desventajas del uso de XML

El uso de XML para almacenar datos de forma legible presenta algunas ventajas, y también inconvenientes.

Ventajas

- **XML es un estándar potente y de amplia aceptación para guardar y comunicar información** acerca de objetos. Su forma de estructurar la información en árbol permite añadir una dimensión más a los registros de una base de datos.
- **Permite la codificación de información separada**, tal y como se debe presentar al usuario.
- Una base de datos XML se puede ver como **una colección de documentos XML**, en la que cada documento representa un registro.
- **Cada documento XML 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. De este modo, cada archivo se puede configurar de forma estructurada, por lo que **es independiente**, pero **fácilmente accesible**.
- Tener colecciones de documentos con un esquema independiente **proporciona a la base de datos flexibilidad y facilita el desarrollo de la aplicación**.
- En los últimos años se ha hecho necesaria la existencia de estándares de intercambio de información con el objetivo de que las organizaciones puedan **compartir su información de una manera más cómoda, automática y eficiente**.

Inconvenientes

- Para encontrar un fragmento específico de información en los contenidos de un nodo o atributo XML **hay que procesar completamente el archivo XML**.
- Los documentos XML son indexados de forma compleja para su acceso rápido.
- Aunque conserven relaciones entre ellos, es **complicado realizar consultas Join sobre documentos XML**.
- Los elementos anidados y los elementos que se repiten (correspondientes a atributos con valores de conjunto) **complican el almacenamiento de los datos XML en un formato relacional**.

Aplicación de XML

Los sistemas de almacenamiento XML deben acomodarse dependiendo del uso y ámbito de la aplicación que los va a usar.

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 XML, y almacenar el XML recibido como datos relacionales para transferir, obtener y almacenar los datos requeridos por la aplicación.

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

Centrados en datos

Si el documento XML tiene una estructura definida y contiene datos que pueden ser actualizados y usados de diversos modos, se dice que el documento está centrado en los datos. En este caso, el sistema responde a la organización de **los datos agrupados en documentos que tienen sentido por sí mismos**, como pueden ser documentos de facturación, órdenes de compra y documentos estructurados. También se adapta para generar documentación estructurada, como son los contenidos de periódicos, artículos y publicidad.

Centrados en los documentos

Los documentos tienden a ser más impredecibles en tamaño y contenido que los centrados en los datos, que son altamente estructurados, con tipos de datos de tamaño limitado y reglas menos flexibles para campos opcionales y contenido.

Almacenamiento en XML

Tipos de bases de datos XML

Los datos almacenados en la base de datos XML se pueden consultar mediante **XQuery** serializada y exportar al formato deseado, que puede ser:

- XML enabled.
- XML nativo (NXD).

Semejanzas entre una BD y XML

Entre una base de datos y un fichero XML con su esquema asociado, podemos establecer las siguientes semejanzas:

- 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 API (SAX, DOM).

Pero hay **muchas más cosas que los diferencian**, debido a que XML no es una base de datos:

- La tecnología XML carece, entre otras cosas, de almacenamiento y actualización eficientes.
- No tiene índices.
- Carece de seguridad.
- No dispone de transacciones.
- Carece de integridad de datos.
- No facilita el acceso concurrente.
- Y, por último, carece también de disparadores, 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 que está más que justificado utilizar alguna base de datos de sistemas muy difundidos, como MySql, IBM DB, Microsoft SQL, ORACLE, etc.

Por otro lado, XML **permite integrar sistemas de información hasta ahora separados**:

Almacenamiento directo de ficheros

Es una opción pobre; las opciones que podemos hacer sobre ellos son limitadas y definidas por el sistema. No se pueden realizar operaciones sobre el contenido y deberemos limitarnos al movimiento del documento como unidad.

Almacenamiento sobre una base de datos

En documentos centrados en datos se puede realizar un mapeo entre los distintos elementos definidos en el documento y el modelo de datos del SGBD.

Esta posibilidad está centrada en documentos debido a que una estructura regular y bien controlada es fácilmente transformable en un esquema relacional. Sin embargo, posee el inconveniente de que **sólo se almacenan los datos que nos interesa conservar y partes del documento se pierden** por el camino (por ejemplo, el formato, comentarios, instrucciones de proceso, etc.) y **a la hora de reconstruir el documento a partir de los datos almacenados obtendremos otro distinto**.

Bases de datos relacionales con XML

Aún cuando no sea de forma nativa, existen razones para usar los tipos de bases de datos relacionales y los productos de bases de datos existentes para almacenar XML.

- 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 por ellas, especialmente **por el rendimiento**.

- Las bases de datos relacionales se basan en las **relaciones entre tablas mediante claves primarias y foráneas** como único medio para representar los datos del mundo real.
- Su manejo, a nivel administración y programación, está asociado a **lenguajes estándares como DDL, DLM, DCL**.
- Los objetos o estructuras jerarquizadas XML han requerido del estudio de teorías complejas para acoplarse a bases de datos relacionales.
- Existen numerosos **middleware** (software que permite la comunicación entre dos aplicaciones de software independientes) para poder integrar estos sistemas.
- Un programa de *middleware* podría permitir a una base de datos, o sistema gestión de bases de datos SGDB, **acceder a los datos de otra**, que se encarga automáticamente de la transferencia de información desde ficheros o estructuras complejas XML hacia las bases de datos relacionales.

XML y bases de datos orientadas a objetos

Las bases de datos orientadas a objetos están fabricadas para manejar, almacenar y mantener un modelo de objetos puro. Normalmente no se basan en el modelo relacional, aunque también podrían derivar de él.

Las funcionalidades de las bases de datos orientadas a objetos, su diseño y sus relaciones, están basados en el **paradigma de la programación orientada a objetos** y la usan como diseño funcional.

Como los documentos XML pueden asociarse a objetos estructurados, la relación de esta forma de guardar la información y las bases de datos orientadas a objetos pueden ser una alternativa para el almacenamiento y gestión de documentos XML.

Cuando diseñamos los documentos XML como objetos pasan a ser **gestionados directamente por el SGBDOO** (Sistema Gestor de Base de Datos Orientado a Objetos). Para acceder, manipular, borrar, etc. los objetos se usa el lenguaje de consulta OQL (Object Query Language).

EL SGBDOO implementará los mecanismos de indexación, optimización, procesamiento de consultas, etc., ya que estos no son específicos para el modelo XML ni los objetos en general.

Despedida

Resumen

Has terminado la lección, repasemos los puntos más importantes que hemos tratado.

- En esta unidad hemos conocido los **sistemas de almacenamiento y su finalidad**, aprendiendo las características de los sistemas que almacenan ficheros XML.
- Se han comentado las **ventajas y desventajas que nos ofrece XML** para el almacenamiento de datos, así como los diferentes formatos que podemos usar.
- Hemos visto los diferentes tipos de sistemas de almacenamiento XML, las diferencias entre bases de datos relacionales y no relacionales, y los sistemas basados en datos o en documentos.
- Por último, hemos realizado una introducción a los sistemas de almacenamiento XML utilizando el paradigma orientado a objetos.

MP_0373. Lenguajes de marcas y sistemas de gestión de información

UF5. Almacenamiento y manipulación de la información

5.2. Herramientas de tratamiento y almacenamiento de información



Índice

Objetivos	3
Introducción.....	4
Herramientas	5
Herramientas de creación de documentos XML.....	5
Herramientas de bases de datos	6
Bases de datos XML-Enabled	6
Bases de datos XML Nativas.....	7
Herramientas de BD XML nativas	7
Tamino	7
EXist.....	8
BaseX	8
Sedna	9
Otras herramientas	9
Despedida	11
Resumen.....	11

Objetivos

Con esta unidad perseguimos los siguientes objetivos:

- Conocer las **herramientas** actuales de manejo de XML.
- Conocer algunas herramientas de **creación de documentos XML**.
- Conocer las herramientas de **bases de datos XML**.
- Conocer las herramientas actuales de **bases de datos XML-Enabled**.
- Conocer las herramientas actuales de bases de datos **XML Nativas**.

Introducción

Las herramientas para manipular XML se pueden dividir en dos grupos:

- Herramientas de **creación**.
- Herramientas de **administración de contenido**.

Muchas de las bases de datos del mercado trabajan con documentos XML, que insertan como datos en la base de datos relacional, sin embargo existen otros productos más modernos que se han creado bajo el ámbito de las bases de datos XML nativas, e incluso se han diseñado para poder incorporar estándares de transformación de XML y de consultas XML. En este capítulo revisamos el estado de todas ellas.

Las herramientas que existen en el mercado en la actualidad para trabajar con XML se pueden dividir en dos grandes grupos:

Herramientas de creación de documentos XML

Herramientas que nos permiten construir XML válidos usando DTD, XML Schema, Schematron etc. Solo servirían para incluir datos en un conjunto de documentos XML.

Herramientas de bases de datos

Serían las que organizarían conjuntos de XML que formarían las bases de datos que hemos estudiado con anterioridad. En este apartado se podrían incluir las bases de datos relacionales XML-Enabled y las nativas.

Herramientas

Herramientas de creación de documentos XML

Las herramientas de creación de documentos XML deben tener las siguientes características:

- **Crear documentos** desde cero o en base a DTD, o XML Schemas.
- **Editar documentos XML**, pudiendo insertar, manipular y borrar datos.
- Editar **de forma textual o visual**.
- **Validar documentos** en base a DTD, XML Schema, Schematron o Relax NG.
- Comprobar **espacios de nombres**.
- **Realizar transformaciones en base a XSLT**.
- Procesar datos con el modelo **XPath**.
- **Creación de DTD**.
- **Creación de XML Schema**.
- **Explorador** de esquemas XML.
- **Esquematización de documentos**, así los elementos se pueden expandir y contraer.
- Capacidad de **generar esquemas** de lenguaje de definición de esquemas XML (XSD) a partir del documento de instancia XML.
- Creación de **estilos XSL**.

Entre las más importantes destacan:

- **SoftQuad Xmetal**. Es una aplicación de software utilizada para crear y editar documentos en XML y SGML. Necesita que los documentos XML estén validados.
- **Adobe FrameMaker+SGML**. Proporciona una GUI completa que manipula documentos SGML/XML. Permite abrir documentos no válidos y rectificarlos.
- **IBM Xeena**. Herramienta desarrollada en Java. Proporciona una GUI para editar documentos válidos.
- **Bluestone Visual-XML**. Herramienta de creación XML orientada a la creación de soluciones de bases de datos empresariales que incluyen XML. Posee un asistente para la publicación de bases de datos que genera DTD y plantillas de documentos XML a partir de tablas de bases de datos. Puede descargarse gratuitamente en: <http://www.bluestone.com/xml/visual-XML>.
- **XML Notepad**. Se trata de una herramienta poco potente de Microsoft que sirve para realizar tareas de creación sencillas. Para validar documentos requiere tener instalado IE5.0. Puede descargarse gratuitamente en <http://msdn.microsoft.com/xml/notepad>.
- **XMLSPY**. Editor de XML y entorno de desarrollo integrado. Este software desarrollado por Altiva permite a los desarrolladores crear aplicaciones basadas en XML y servicios web utilizando tecnologías como XML, XML Schema, XSLT, XPath, XQuery entre otras. <http://www.xmlspy.com/>.
- **XMLwriter**. Proporciona funcionalidades de forma sencilla y eficiente para la realización de tareas de formato, validación, transformación y estilo. <http://xmlwriter.net/>.

Herramientas de bases de datos

El software que nos permite crear datos basados en documentos XML e implementa los módulos de manejo y búsqueda de datos se clasifica por dos tipos:

- Software de **base de datos XML-Enabled**, que son las bases de datos clásicas que se han adaptado a trabajar con XML.
- Software que implementa **bases de datos XML nativas**.

Bases de datos XML-Enabled

Las **bases de datos XML-Enabled** son aquellas que incorporan la información de un documento XML en su **correspondiente esquema relacional o de objetos**, por lo que contienen funcionalidad para poder manejar la transferencia de datos entre documentos XML y las estructuras de datos propias.

Son utilizadas por las aplicaciones centradas en datos, excepto cuando la base de datos soporta almacenamiento XML nativo.

La diferencia fundamental entre estas últimas es que las XML-Enabled utilizan estructuras específicas que deben ser mapeadas al documento XML en tiempo de diseño.

Podemos decir también que un **SGBD XML-Enabled solo puede manejar y almacenar los documentos que encajan dentro del modelo definido para ellos**, mientras que un SGBD XML nativo debe manejar todos los tipos de documentos posibles.



IBM DB2 es un sistema de base de datos de nivel de entrada diseñado para el proceso de transacciones y la gestión de cargas de trabajo de consultas complejas. Proporciona funciones a escala de empresa y está optimizada para utilizar hasta 8 núcleos de procesador y 8 GB de memoria.



Oracle es un gestor de base de datos relacional que hace uso de los recursos del sistema informático en todas las arquitecturas de hardware para garantizar su aprovechamiento al máximo en ambientes cargados de información. Es el conjunto de datos que proporciona la capacidad de almacenar y acudir a estos de forma recurrente con un modelo definido como relacional. Además es una suite de productos que ofrece una gran variedad de herramientas. Es el mayor y más usado Sistema Gestor de Base de Datos Relacional (RDBMS) en el mundo. La Corporación Oracle ofrece este RDBMS como un producto incorporado a la línea de producción. Además incluye cuatro generaciones de desarrollo de aplicación, herramientas de reportes y utilitarios.



PostgreSQL es un SGBD relacional orientado a objetos y libre, publicado bajo la licencia BSD.

Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyada por organizaciones comerciales. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

Bases de datos XML Nativas

Una base de datos XML Nativa es aquella que:

- **Define un modelo de datos XML.** Define qué elementos son lógicamente significativos. Todos tendrán en cuenta los elementos, atributos, texto y orden en el documento, aunque es posible que de un sistema a otro los elementos reflejados en el modelo varíen.
- Utiliza el documento como unidad mínima de almacenamiento.
- **Puede usar cualquier estrategia de almacenamiento.** El almacenamiento físico de los documentos puede realizarse sobre un SGBD relacional tradicional, sobre ficheros con estructura propia o cualquier otro método.

La principal diferencia es que **una base de datos XML Nativa provee de su propio modelo de datos**, mientras que un sistema XML-Enabled tiene su propio modelo de datos y añade una capa de software que permite de alguna manera almacenar documentos XML y recuperar los datos generando nuevos documentos XML.

Herramientas de BD XML nativas

Algunas bases de datos XML Nativas son:

Tamino

Tamino es el SGBD nativo de la empresa SoftwareAG. Es un producto comercial de alto rendimiento y disponibilidad, además de ser uno de los primeros SGBD XML nativos disponibles.

Native XML Data Store + XML Engine es el componente central de la arquitectura, contiene el parser XML, el almacén nativo de datos y el intérprete de consultas.

- **Arquitectura**
 - **Data Map.** Almacén de metadatos, contiene información sobre cómo validar esquemas, almacenar e indexar información, mapeo de estructuras, etc.
 - **X-Node.** Es el componente para dar acceso a bases de datos externas, mapeando los datos a estructuras XML. El acceso a esta información es transparente para el usuario y se accede a ella cada vez que se necesita, es decir, que no es replicada.

- **X-Tension.** Permite la definición de funciones de usuario para ampliar las prestadas por Tamino.
- **Tamino Manager.** Herramienta gráfica de gestión.
- **Almacenamiento de documentos.** Los documentos se almacenan en una base de datos propia y no se transforman en otro modelo. Existe un espacio separado para documentos y para índices.

Un *doctype* es el elemento raíz de un DTD o XML Schema, es decir, el elemento que define el comienzo y el final del documento.

La base de datos está estructurada en colecciones. Una colección es un conjunto de documentos, de modo que es una estructura de árbol donde cada documento pertenece a una única colección.

Ex**I**st

Ex**I**st-db es un sistema de gestión de bases de datos libre y de código abierto que almacena datos XML de acuerdo a un modelo de datos XML.

Algunas de sus características son: soporte para distintos lenguajes de consultas XML como XQuery, XPath o XSLT, indexación de documentos y soporte para la actualización de los datos y para multitud de protocolos como SOAP, XML-RPC, WebDav y REST. En la actualidad cumple el estándar XQuery en un 99.4%.

Descarga y documentación

Para descargar la aplicación y examinar la documentación puedes visitar este enlace.

<http://exist-db.org/exist/apps/homepage/index.html>

Base**X**

Base**X** es una base de datos XML escalable, con un alto rendimiento y muy ligera. Con procesador XPath/XQuery y soporte de la última actualización del W3C, es una opción idónea para nuestro trabajo.

Entre sus características se encuentran:

- Alto rendimiento de almacenamiento con texto, atributos, texto completo y rutas.
- Apoyo de las recomendaciones del W3C XPath/XQuery.
- Una de las más altas tasas de cumplimiento para todas las especificaciones admitidas.
- Arquitectura cliente/servidor con soporte de transacciones seguras ACID, gestión de usuarios y autenticación.
- Visualizaciones altamente interactivas, con soporte para documentos XML de gran extensión.
- Editor XQuery en tiempo real disponible, con resultado de sintaxis y *feedback* de errores.
- Amplia gama de interfaces: REST/RESTXQ, WebDAV, XQJ, XML:DB.

Descarga

Puedes descargar el software en este enlace. <http://baseX.org/download/>

Sedna

Sedna es un sistema de gestión de base de datos XML nativa de código abierto, implementado a partir de cero con todas las funciones de administración de una base de datos para almacenar grandes cantidades de datos XML.

La aportación especial de la base de datos Sedna es su estrategia de almacenamiento mejorado basado en esquemas (*schema-driven storage strategy*), que permite gran eficiencia a la hora de procesar las consultas y almacenamientos en la base de datos. Tiene un sistema de manejo de memoria muy novedoso.

Dentro del apartado de almacenamiento debemos mencionar también un elemento novedoso de Sedna: las relaciones de inclusión entre elementos; se utilizan punteros directos para representar relaciones de un nodo XML, como padre-hijo con sus respectivos hermanos. Este método, a diferencia de enfoques relacionales que requieren realizar “*joins*” para atravesar un documento XML, permite realizar esto a través de un puntero directo.

Descarga y documentación

Puedes descargar el software en este enlace. <https://www.sedna.org/>

Otras herramientas

- **Qexo:** escrito en Java y con licencia GPL, se distribuye integrado dentro del paquete Kawa; un IDE (Entorno de Desarrollo Integrado) que permite crear aplicaciones en lenguaje Java.
- **Saxon:** escrito en Java y distribuido en dos paquetes:
 - **Saxon-B** es open-source bajo licencia GPL y contiene una implementación básica de XSLT 2.0 y XQuery.
 - **Saxon-SA**, contiene un procesador completo XSLT y XQuery, pero tiene licencia propietaria, aunque está disponible una licencia de evaluación de 30 días.
- **Qizx/open:** es una implementación Java de todas las características del lenguaje, excepto la importación y validación de XML-Schemas. Es uno de los motores con licencia GPL más completo que existe.
- **Xquark Bridge:** es una herramienta que permite importar y exportar datos a bases de datos relacionales utilizando XML, ofreciendo, por tanto, la posibilidad de manejar estructuras XML y realizar la transformación a objetos de la base de datos y viceversa. Además, XQuark respeta las restricciones de integridad y transforma las relaciones implícitas en los documentos XML en relaciones explícitas en la base de datos. También soporta la consulta y manipulación de los datos en formato XML utilizando el lenguaje XQuery. XQuark-Bridge soporta MySQL, Oracle, SQLServer y Sybase y tiene una licencia LGPL.

- **BumbleBee:** es un entorno de prueba automático, creado para evaluar motores de XQuery y validar consultas XQuery. BumbleBee permite, entre otras cosas, comprobar si una versión más moderna de nuestro motor permite seguir ejecutando nuestras consultas. BumbleBee se distribuye con un conjunto de pruebas ya preparadas y además ofrece un entorno sencillo para redactar y ejecutar nuestras propias pruebas.

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

En esta unidad hemos conocido algunas de las herramientas con las que podemos trabajar con los datos XML, tanto para su creación como para la gestión de bases de datos.

- Las herramientas de creación nos permiten construir XML válidos usando DTD, XML Schema, Schematron etc. Estas solo servirían para incluir datos en un conjunto de documentos XML.
- Las herramientas de bases de datos son las que organizarían conjuntos de XML que formarían las bases de datos que hemos estudiado con anterioridad. Éstas pueden ser bases de datos relacionales XML-Enabled o nativas.

5.3. Manipulación de información en documentos XML



Índice

Objetivos	3
Conceptos	4
Búsqueda de la información (XQUERY)	5
Técnicas de búsqueda en XML.....	5
XQuery.....	6
XQuery, características.....	7
XQuery, funciones de entrada	7
XQuery, consultas	8
Ejemplo consultas XQuery con BaseX	9
XQuery, reglas	12
Diferencias entre la cláusula <i>for</i> y la cláusula <i>let</i>	13
Tipos de nodos en XQuery	14
Cuantificadores existenciales	15
Operadores y funciones principales	15
Unión	16
Except	16
distinct-values()	17
empty().....	17
Manipulación de la información.....	19
XUpdate	19
Insertar un elemento	20
Modificar un elemento	21
Borrar un elemento	22
Renombrar elementos	23
XQuery Update Facility.....	23
Insert	24
Delete.....	25
Replace	25
Rename	26
Despedida	27
Lecturas recomendadas.....	27
Resumen.....	27

Objetivos

Con esta unidad perseguimos los siguientes objetivos:

- Conocer cómo podemos **actualizar los datos en fichero XML**.
- Conocer las **herramientas** de gestión de Bases de Datos XML.
- Aprender a realizar consultas con **XQuery**.
- Conocer **XUpdate** y su funcionamiento.
- Conocer **XQuery Update Facility** y su funcionamiento.

Introducción

Conceptos

Tener una base de datos basada en registros XML permite tener gran cantidad de datos **accesibles en cualquier momento**, pero los datos son algo vivo y evolucionan con el tiempo, lo cual nos lleva a la **necesidad de poder modificarlos**.

Como hemos definido anteriormente la modificación de documentos XML se puede realizar de dos formas:

- Usando las API de manipulación SAX y DOM. La primera (SAX) permite recorrer y por lo tanto consultar los datos que se almacenan en el XML. La segunda es (DOM) que permite no solo leer el documento sino insertar información en él. Como DOM trabaja a nivel de árbol en memoria, debe existir un Parser que transforme el árbol en memoria de nuevo a un fichero XML.
- Usando lenguajes de manipulación. Los lenguajes que vamos a revisar en este documento son los que tienen entre sus características la de poder modificar documentos XML. Estos lenguajes son: **XUpdate y XQuery Update Facility**.

Una posibilidad para la recuperación de un documento y su modificación es mediante un programa específico (a través de algún API como SAX ó DOM) y sustituir al documento original; esta solución es muy costosa y nada flexible ya que tendríamos que generar un programa para cada consulta. Esto hace evidente la necesidad de lenguajes que permitan modificar el contenido del mismo de manera declarativa.

Búsqueda de la información (XQUERY)

Técnicas de búsqueda en XML

Cuando trabajamos con bases de datos, la búsqueda es una de las funciones fundamentales. Por lo tanto los sistemas de bases de datos que almacenan documentos XML deben de incluir esa misma funcionalidad. La forma de abordar esa función en bases de datos XML dependerá del tipo de base de datos que sea.

Las bases de datos en XML **son herramientas para almacenar y recuperar la información** cuya base son documentos XML.

- Las bases de datos deben mostrar la información que almacena sobre un tema y en qué documento se encuentra dicha información.
- Estas herramientas proporcionan la referencia, la versión electrónica de la información o bien el propio conjunto de datos.

La problemática existente en estas bases de datos es cómo buscar en ellas. No porque sea especialmente difícil, sino porque no hay un sistema de interrogación común para todas ellas.

Para poder buscar de una forma metódica en una base de datos basada en XML necesitamos poder contar con una serie de herramientas que el sistema gestor implemente.

XQuery

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. Este lenguaje se revisa en las siguientes lecciones.

XPath

XPath es un lenguaje estándar y muy importante en XSLT.

- XPath se puede utilizar para navegar a través de elementos y atributos en un documento XML.
- Tiene una sintaxis para definir partes de un documento XML.
- Utiliza expresiones de ruta para navegar en documentos XML.
- Contiene una biblioteca de funciones estándar.
- XPath es una recomendación del W3C.

XQuery

La realización de consultas en bases de datos XML permite acceder a la información a través de un lenguaje común que tiene semejanzas con otros lenguajes de consultas como SQL. A través de dicho lenguaje los elementos y datos que conforman cualquier documento XML bien formado son accesibles.

Actualmente, XML se ha convertido en una herramienta de uso cotidiano en los entornos de tratamiento de información y en los entornos de programación.

Sin embargo, conforme más se utiliza en proyectos complejos y se trabaja con mayor cantidad de datos, se comprueba que las herramientas usadas hasta ahora que pueden manipular los documentos XML con su conjunto de datos, **los parsers SAX y DOM**, no son buenas soluciones para manejar colecciones de datos en XML grandes y complejas. **Estas herramientas permiten a muy bajo nivel manejar los datos pero generan mucho código y no son muy manejables a no ser que sean manejados por un software a medida.**

Otra solución existente es el **estándar XSLT**, que permite definir transformaciones sobre colecciones de datos en XML en otros formatos, como HTML o PDF y las herramientas que le dan soporte. Sin embargo XSLT sólo es aplicable en los casos en los que se quiera obtener una representación distinta de los datos, no cuando se desea localizar una información concreta dentro de dicho conjunto de datos.

Por todo ello, es necesario pensar en **un lenguaje que defina de forma sencilla consultas** o recorridos complejos sobre colecciones de datos en XML, los cuales devuelvan todos los nodos que cumplan ciertas condiciones.

Este **lenguaje** debe ser, además, **declarativo**, por lo tanto independientemente de la forma en que se realice el recorrido o de dónde se encuentren los datos. **Este lenguaje ya existe y se llama XQuery.**

XQuery es a XML lo que es SQL a las bases de datos relacionales.

XQuery es un lenguaje para encontrar y extraer elementos y atributos de documentos XML.

XQuery 1.0 y XPath 2.0 comparten el mismo modelo de datos y admiten las mismas funciones y operadores.

XQuery, características

Un lenguaje procedimental clásico ejecuta una lista de comandos. XQuery es un **lenguaje funcional**, donde **cada consulta es una expresión** que es evaluada y devuelve un resultado, al igual que en SQL.

Diversas **expresiones** pueden combinarse de una manera muy flexible para crear nuevas expresiones más complejas y de mayor potencia semántica.

Características

- **XQuery es un lenguaje declarativo.** Al igual que SQL hay que indicar qué se quiere, no cómo se puede obtener.
- **XQuery es independiente del protocolo de acceso a la colección de datos.** Una consulta en XQuery debe funcionar igual al consultar un archivo local, que al consultar un servidor de bases de datos, que al consultar un archivo XML en un servidor web.
- Las consultas y los resultados **respetan el modelo de datos XML**.
- Las consultas y los resultados ofrecen **soporte para los namespace**.
- XQuery es capaz de **soportar XML-Schemas y DTDs**.
- **XQuery trabaja con independencia de la estructura del documento**, esto es, sin necesidad de conocerla.
- XQuery soporta tanto **tipos simples**; como enteros y cadenas, y tipos complejos; como un nodo compuesto por varios nodos hijos.
- Las consultas soportan **cuantificadores** universales (para todo) y existenciales (existe).
- Las consultas soportan operaciones sobre **jerarquías de nodos y secuencias de nodos**.
- En una consulta se puede combinar información de **múltiples fuentes**.
- Las consultas son capaces de **manipular** los datos independientemente del origen de estos.
- Mediante XQuery es posible definir consultas que **transformen las estructuras** de información originales y debe ser posible crear nuevas estructuras de datos.
- El lenguaje de consulta **es independiente de la sintaxis**, esto es, debe ser posible que existan varias sintaxis distintas para expresar una misma consulta en XQuery.

XQuery, funciones de entrada

XQuery utiliza las **funciones de entrada** en las cláusulas *for* o *let* o en expresiones XPath para identificar el origen de los datos.

Actualmente XPath y XQuery definen dos funciones de entrada distintas, ***doc (URI)* y *collection (URI)***.

La función *doc (URI)*

Devuelve el nodo documento, o nodo raíz, del documento referenciado por un identificador universal de recursos (URI). Esta es la función más habitual para acceder a la información almacenada en archivos.

La función *collection (URI)*

Devuelve una secuencia de nodos referenciados por una URI, sin necesidad de que exista un nodo documento o nodo raíz.

Esta es la función más habitual para acceder a la información almacenada en una base de datos que tenga capacidad para crear estructuras de datos XML.

XQuery, consultas

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.

En XQuery las expresiones y los valores que devuelven son dependientes del contexto. Por ejemplo los nodos que aparecerán en el resultado dependen de los namespaces, de la posición donde aparezca la etiqueta raíz del nodo (dentro de otra, por ejemplo), etc.

En XQuery las consultas pueden estar compuestas por cláusulas de hasta cinco tipos distintos. Las consultas siguen la norma *FLWOR* (leído como *flower*), siendo *FLWOR* las siglas de *For, Let, Where, Order y Return*.

A continuación, se describe la función de cada bloque.

for:

Vincula una o más variables a expresiones escritas en XPath, creando un flujo de tuplas en el que cada tupla está vinculada a una de las variables.

let:

Vincula una variable al resultado completo de una expresión añadiendo esos vínculos a las tuplas generadas por una cláusula *for* o, si no existe ninguna cláusula *for*, creando una única tupla que contenga esos vínculos.

where:

Filtira las tuplas eliminando todos los valores que no cumplan las condiciones dadas.

order by:

Ordena las tuplas según el criterio dado.

return:

Construye el resultado de la consulta para una tupla dada, después de haber sido filtrada por la cláusula *where* y ordenada por la cláusula *order by*.

Ejemplo consultas XQuery con BaseX

Realicemos un pequeño ejemplo de uso utilizando XQuery. Para realizar las consultas instalaremos el editor BaseX.

Base X

BaseX es muy liviano, fácil de usar y se ejecuta de forma automática.

<http://basex.org/download/>

Según el siguiente documento XML:

```
<bib>
    <libro año="1994">
        <titulo>Redes TCP/IP</titulo>
        <autor>
            <apellido>Pérez</apellido>
            <nombre>Juan Carlos</nombre>
        </autor>
        <editorial>Addison-Wesley</editorial>
        <precio> 49.80</precio>
    </libro>
    <libro año="1992">
        <titulo>Programación Java</titulo>
        <autor>
            <apellido>Pérez</apellido>
            <nombre>Juan Carlos</nombre>
        </autor>
        <editorial>Addison-Wesley</editorial>
        <precio>49.80</precio>
    </libro>
    <libro año="2000">
        <titulo>Manejo de datos</titulo>
        <autor>
            <apellido>Ramírez</apellido>
            <nombre>Julio</nombre>
        </autor>
        <autor>
            <apellido>Bueno</apellido>
            <nombre>Federico</nombre>
        </autor>
        <autor>
            <apellido>Gálvez</apellido>
            <nombre>Jesús</nombre>
        </autor>
        <editorial>Anaya Mutimedia</editorial>
        <precio>39.95</precio>
    </libro>
    <libro año="1999">
        <titulo> Programación Python</titulo>
        <autor>
            <apellido>Palomo</apellido>
            <nombre>Roberto</nombre>
        </autor>
        <autor>
            <apellido>Otero</apellido>
            <nombre>Julio</nombre>
        </autor>
        <autor>
            <apellido>Alvarez</apellido>
            <nombre>Juan</nombre>
        </autor>
        <editor>
```

```

        <apellido>Marchenado</apellido>
        <nombre>Félix</nombre>
        <afiliacion>CITI</afiliacion>
    </editor>
    <editorial>Editorial S.M.</editorial>
    <precio>129.95</precio>
</libro>
</bib>
```

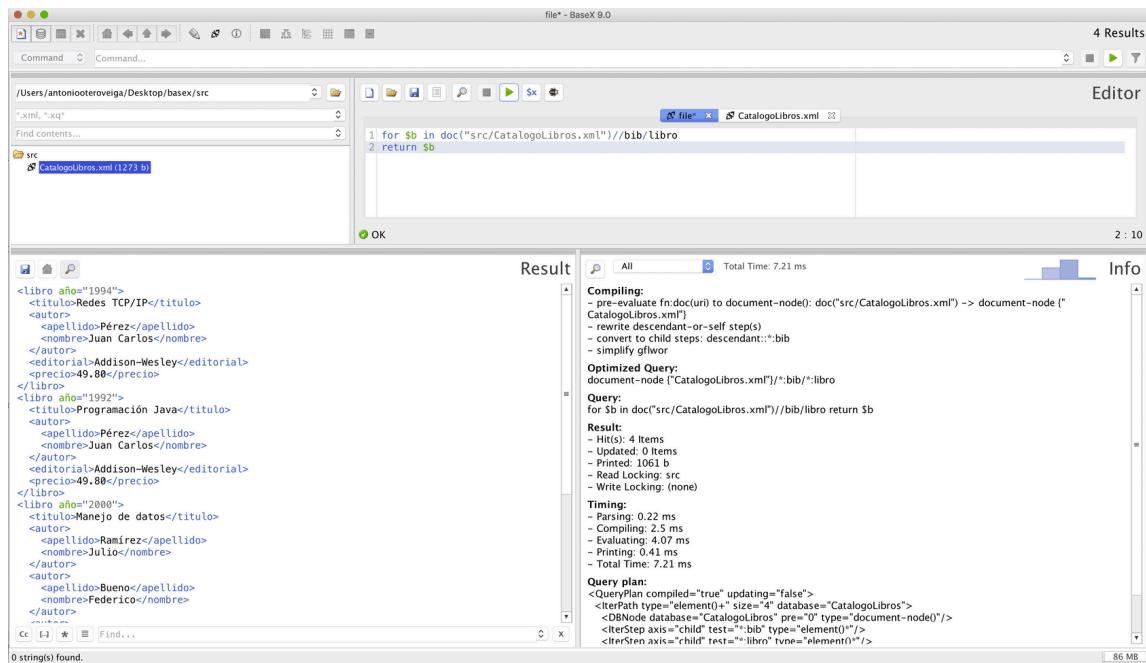
Contenido del archivo de XML de ejemplo

Si quieres probar este ejemplo, solo tienes que instalar BaseX en tu ordenador y guardar en la carpeta *src* del programa, el contenido del XML en un archivo llamado "CatalogoLibros.xml".

En el siguiente ejemplo de cláusula **for**, la variable **\$b** tomará como valor cada uno de los nodos *libros* que contenga en el archivo "CatalogoLibros.xml" (mostrado en el ejemplo anterior). Cada uno de esos nodos *libros*, será una tupla vinculada a la variable **\$b**.

```
for $b in doc("src/CatalogoLibros.xml")//bib/libro
return $b
```

Cláusula *for*



A continuación se muestra un ejemplo de una consulta donde aparecen las 5 cláusulas que se usan en las consultas. La siguiente consulta devuelve los títulos de los libros que tengan más de dos autores ordenados por su título.

```
for $b in doc("src/CatalogoLibros.xml")//libro
let $c := $b//autor
where count($c) > 2
order by $b/titulo
return $b/ titulo
```

The screenshot shows the BaseX 9.0 interface. In the top right, it says "2 Results". The main area is the "Editor" pane where the following XQuery code is written:

```

1 for $b in doc("src/CatalogoLibros.xml")//libro
2 let $c := $b//autor
3 where count($c) > 2
4 order by $b/titulo
5 return $b/titulo
    
```

Below the editor is the "Result" pane, which displays the output of the query:

```

<titulo>Manejo de datos</titulo>
<titulo>Programación Python</titulo>
    
```

The "Info" pane at the bottom right provides compilation details:

- Compiling:**
 - pre-evaluate fn:doc(uri) to document-node(): doc("src/CatalogoLibros.xml") -> document-node ("CatalogoLibros.xml")
 - rewrite descendant-or-self step(s)
 - convert to child steps: descendant::*:libro
 - rewrite descendant-or-self step(s)
 - rewrite > operator to range comparison: (count(\$c_1) > 2) -> (2.0000000000000004 <= count(\$c_1))
 - inline \$c_1
 - rewrite where clause(s)
- Optimized Query:**

$$\text{for } \$b_0 \text{ in document-node ("CatalogoLibros.xml")/*:bib/*:libro}[(2.0000000000000004 \leq \text{count}(\text{descendant}::*:autor))) \text{ order by } \$b_0/*:titulo \text{ empty least return } \$b_0/*:titulo]$$
- Query:**

$$\text{for } \$b \text{ in doc("src/CatalogoLibros.xml")//libro} \text{ let } \$c := \$b//autor \text{ where count}(\$c) > 2 \text{ order by } \$b/titulo \text{ return } \$b/titulo$$

Una expresión *FLWOR* vincula variables a valores con cláusulas *for* y *let* y maneja esos vínculos para crear nuevas estructuras de datos XML. A continuación se muestra otro ejemplo de consulta XQuery.

La siguiente consulta devuelve los títulos de los libros del año 1992. Como “año” es un atributo y no una etiqueta se le antecede con un carácter “@”.

```
for $b in doc("src/CatalogoLibros.xml") //libro
where $b/@año = "1992"
return $b/titulo
```

The screenshot shows the BaseX 9.0 interface. In the top right, it says "1 Results". The main area is the "Editor" pane where the following XQuery code is written:

```

1 for $b in doc("src/CatalogoLibros.xml")//libro
2 where $b/@año = "1992"
3 return $b/titulo
    
```

Below the editor is the "Result" pane, which displays the output of the query:

```

<titulo>Programación Java</titulo>
    
```

The "Info" pane at the bottom right provides compilation details:

- Compiling:**
 - pre-evaluate fn:doc(uri) to document-node(): doc("src/CatalogoLibros.xml") -> document-node ("CatalogoLibros.xml")
 - rewrite descendant-or-self step(s)
 - convert to child steps: descendant::*:libro
 - apply attribute index for "1992"
 - rewrite where clauses(s)
- Optimized Query:**

$$\text{for } \$b_0 \text{ in db:attribute("CatalogoLibros", "1992")/self::*:año/parent::*:libro} \text{ return } \$b_0/*:titulo$$
- Query:**

$$\text{for } \$b \text{ in doc("src/CatalogoLibros.xml")//libro} \text{ where } \$b/@año = "1992" \text{ return } \$b/titulo$$
- Result:**
 - Hits(1 Item
 - Updated: 0 Items
 - Printed: 35 b
 - Read Locking: src
 - Write Locking: none

XQuery, reglas

A continuación, vamos a revisar las reglas que son de obligado cumplimiento para conformar cualquier consulta escrita en XQuery.

- **for y let sirven para crear las tuplas** con las que trabajará el resto de las cláusulas de la consulta y pueden usarse tantas veces como se desee en una consulta, incluso dentro de otras cláusulas. Sin embargo **solo puede declararse una única cláusula where**, una única cláusula *order by* y una única cláusula *return*.
- Las cláusulas **FLWOR** no son obligatorias en una consulta XQuery. Por ejemplo, una expresión XPath, como la que se muestra a continuación, es una consulta válida y no contiene ninguna de las cláusulas *FLWOR*.

```
doc("src/CatalogoLibros.xml")/bib/libro/titulo[/bib/libro/autor/apellido='Pérez']
```

Esta expresión **XPath**, que también es una consulta XQuery válida, devuelve los títulos de los libros que tengan algún autor de apellido "Pérez". Es posible especificar varios criterios de ordenación en la cláusula *order by*, separándolos por comas.

The screenshot shows the BaseX 9.0 interface with the following details:

- Left Panel (File Explorer):** Shows the file system path: /Users/antoniooteroivega/Desktop/basex/src. It contains a folder named 'src' which contains 'CatalogoLibros.xml' (1505 b).
- Middle Panel (Editor):** Displays the XQuery code: `1 doc("src/CatalogoLibros.xml")/bib/libro/titulo[/bib/libro/autor/apellido='Pérez']`. The result count is shown as '4 Results'.
- Bottom Panel (Result):** Shows the query results as XML fragments:


```
<titulo>Redes TCP/IP</titulo>
<titulo>Programación Java</titulo>
<titulo>Manejo de datos</titulo>
<titulo>Programación Python</ultimo>
```

 Below the results, the 'Info' tab provides performance metrics:
 - Compiling:** - pre-evaluate fn:doc(uri) to document-node(): doc("src/CatalogoLibros.xml") -> document-node {CatalogoLibros.xml"}
 - Optimized Query:** document-node ("CatalogoLibros.xml")/*:bib/*:libro/*:titulo{root()/*:bib/*:libro/*:autor/*:apellido = "Pérez"}
 - Query:** doc("src/CatalogoLibros.xml")/bib/libro/titulo[autor/apellido='Pérez']
 - Result:** - Hit(s): 4 Items
 - Updated: 0 Items
 - Printed: 136 b
 - Read Locking: src
 - Write Locking: (none)
 - Timing:**
 - Parsing: 0.21 ms
 - Compiling: 2.95 ms

Ejemplo de aplicación de `doc("src/CatalogoLibros.xml")/bib/libro/titulo[/bib/libro/autor/apellido='Pérez']` en BaseX.

Diferencias entre la cláusula *for* y la cláusula *let*

Revisemos la diferencia entre una cláusula *for* y una cláusula *let*. Ejecutamos una consulta que muestre los títulos de todos los libros almacenados en el archivo "CatalogoLibros.xml", primero con una cláusula *for* y, a continuación, con una cláusula *let* y vamos a detallar qué diferencia hay en la información obtenida.

Cláusula *for*

```
for $d in doc("CatalogoLibros.xml")/bib/libro/titulo return { $d }
```

Cláusula *For*

```
<titulos>
    <titulo>Redes TCP/IP</titulo>
</titulos>
<titulos>
    <titulo>Programación Java</titulo>
</titulos>
<titulos>
    <titulo>Manejo de datos</titulo>
</titulos>
<titulos>
    <titulo> Programación Python</titulo>
</titulos>
```

Resultado cláusula *For*

Cláusula *let*

```
let $d := doc("CatalogoLibros.xml")/bib/libro/titulo
return { $d }
```

Cláusula *let*

```
<titulos>
    <titulo>Redes TCP/IP</titulo>
    <titulo>Programación Java</titulo>
    <titulo>Manejo de datos</titulo>
    <titulo> Programación Python</titulo>
</titulos>
```

Resultado cláusula *let*

Examinando el resultado de los dos ejemplos anteriores podemos comprobar que, **aunque afecta a los mismos datos, la composición de los nodos es diferente.**

Tipos de nodos en XQuery

En XQuery, hay siete tipos de nodos: elemento, atributo, texto, espacio de nombres, nodo de instrucción de proceso, comentario y documento (raíz).

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

Nodos

Al igual que en XML tenemos el nodo documento (raíz), nodos de elemento y nodos de atributo.

Padres

Cada elemento y atributo tiene un parente. El elemento *libro* es el parente de título, autor, año y precio.

Hermanos

Son los nodos que tienen el mismo parente. El título, autor, año y precio son todos hermanos.

Descendientes

Los descendientes de un nodo son los que contiene. Los descendientes de *librería* son el libro, el título, el autor, el año y el precio.

Valores atómicos

Una serie de datos indivisible, Ej.: J K. Rowling.

Hijos

Los nodos pueden tener cero, uno o más hijos. El título, el autor, el año y precio son todos hijos del elemento *libro*.

Antepasados

Los antepasados son el parente de un nodo, parente de los padres, etc. Los antepasados de *título* son el elemento *libro* y el elemento *librería*.

Cuantificadores existenciales

XQuery trabaja con dos tipos de cuantificadores existenciales llamados **some** y **every**.

Some nos permite definir consultas que devuelvan **algún elemento** que satisfaga la condición y **every** consultas que devuelvan los elementos en los que **todos sus nodos** satisfagan la condición.

Por ejemplo, la siguiente consulta devuelve los títulos de los libros en los que **al menos uno** de sus autores es Juan Carlos Pérez.

```
for $b in doc("CatalogoLibros.xml")//libro
where some $a in $b/autor
satisfies ($a/last="Pérez" and $a/first="Juan Carlos")
return $b/titulo
```

Consulta usando **some**

El resultado sería:

```
<title>Redes TCP/IP</title>
<title>Programación Java</title>
```

Resultado de la consulta **some**

Operadores y funciones principales

El conjunto de funciones y operadores soportado por XQuery 1.0 es el mismo conjunto de funciones y operadores utilizado en XPath 2.0 y XSLT 2.0.

XQuery soporta operadores y 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**.

Los operadores y funciones más importantes son:

- **Matemáticos:** +, -, *, div(*), idiv(*), mod.

- **Comparación:** =, !=, <, >, <=, >=, not().
- **Secuencia:** union (), intersect, except.
- **Redondeo:** floor(), ceiling(), round().
- **Funciones de agrupación:** count(), min(), max(), avg(), sum().
- **Funciones de cadena:** concat(), string-length(), startswith(), ends-with(), substring(), upper-case(), lower-case(), string().
- **Uso general:** distinct-values(), empty(), exists().

Unión

El operador **union (|)** recibe dos secuencias de nodos y devuelve una secuencia con todos los nodos existentes en las dos secuencias originales.

A continuación se muestra una consulta que usa el operador unión para obtener una lista ordenada de apellidos de todos los autores y editores:

```
for $1 in distinct-values(doc("CatalogoLibros.xml")  
//(autor | editor)/apellido)  
order by $1  
return <apellidos>{ $1 }</apellidos>
```

Cláusula XQuery

```
<apellidos>Ramírez</apellidos>  
<apellidos>Bueno</apellidos>  
<apellidos>Marchenado</apellidos>  
<apellidos>Pérez</apellidos>  
<apellidos>Gálvez</apellidos>
```

Resultado

Except

El operador de **sustracción (except)** recibe dos secuencias de nodos como operandos y devuelve una secuencia conteniendo todos los nodos del primer operando que no aparezcan en el segundo operando.

A continuación se muestra una consulta que usa el operador sustracción para obtener un nodo libro con todos sus nodos hijos salvo el nodo <precio>.

```
for $b in doc("CatalogoLibros.xml")//libro
where $b/titulo = "Redes TCP/IP"
return
<libro>
{ $b/* }
{ $b/* except $b/precio }
</libro>
```

Cláusula XQuery

```
<libro year = "1994">
    <title>Redes TCP/IP</title>
    <author>
        <apellidos>Pérez</apellidos>
        <first>Juan Carlos</first>
    </author>
    <publisher>Addison-Wesley</publisher>
</libro>
```

Resultado

distinct-values()

La función **distinct-values()** extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados.

Por ejemplo, la siguiente consulta devuelve todos los apellidos distintos de los autores.

```
for $l in distinct-values(doc("CatalogoLibros.xml")//autor/apellidos)
return <apellidos>{ $l }</apellidos>
```

Cláusula XQuery

```
<apellidos>Pérez</apellidos>
<apellidos>Ramírez</apellidos>
<apellidos>Bueno</apellidos>
<apellidos>Gálvez</apellidos>
```

Resultado

empty()

La función **empty()** devuelve *true* si la expresión entre paréntesis está vacía.

Por ejemplo, la siguiente consulta devuelve todos los nodos *libro* que tengan al menos un nodo *autor*.

```
for $b in doc("CatalogoLibros.xml")//libro
where not(empty($b/autor)) s
return $b
```

Cláusula XQuery

```
<libro año="1994">
    <titulo>Redes TCP/IP</titulo>
    <autor>
        <apellido>Pérez</apellido>
        <nombre>Juan Carlos</nombre>
    </autor>
    <editorial>Addison-Wesley</editorial>
    <precio> 49.80</precio>
</libro>
<libro año="1992">
    <titulo>Programación Java</titulo>
    <autor>
        <apellido>Pérez</apellido>
        <nombre>Juan Carlos</nombre>
    </autor>
    <editorial>Addison-Wesley</editorial>
    <precio>49.80</precio>
</libro>
```

Resultado

Manipulación de la información

XUpdate

XUpdate utiliza la sintaxis XML para realizar las modificaciones en un documento. Es un lenguaje para permitir la actualización de documentos XML.

Los requisitos planteados con este lenguaje son:

- Es **declarativo**, define cómo consultar y actualizar el contenido XML.
- Es simple y directo.
- Es compatible con XML Namespaces, XPath y XPointer.
- Permite extensiones futuras.
- Posee sintaxis XML.
- Es independiente de cualquier *parser* o modelo.
- Puede actualizar una parte o el documento completo.
- Puede actuar específicamente con el API XML:DB.

La estructura básica es

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
xmlns:xupdate="http://www.xmldb.org/xupdate">
...
...
...
</xupdate:modifications>
```

En la sentencia **xupdate:modifications**, se pueden incluir:

xupdate:insert-before:

Inserta un nodo hermano precediendo al nodo de contexto.

xupdate:append:

Inserta un nodo como hijo del nodo de contexto.

xupdate:insert-after:

Inserta un nodo hermano a continuación del nodo de contexto.

xupdate:update:

Actualiza el contenido de los nodos seleccionados.

xupdate:remove:

Elimina los nodos seleccionados.

xupdate:rename:

Permite cambiar el nombre de un elemento o atributo.

Para acceder a los elementos y atributos se usa XPath.

Documentación

La misión de XUpdate es proporcionar un medio de actualización abiertas y flexibles para modificar los datos en documentos XML. <http://xmldb-org.sourceforge.net/xupdate/index.html>

Insertar un elemento

Veamos cómo insertar elemento mediante unos ejemplos:

Tenemos el siguiente XML:

```
<alumnos>
  <alumno>
    <nombre>Juan Palomo</nombre>
    <estudios>Grado superior de DAM</estudios>
  </alumno>
  <alumno>
    <nombre>María Palomita</nombre>
    <estudios>Grado superior de DAW</estudios>
  </alumno>
  <alumno>
    <nombre>Roberto Pomba</nombre>
    <estudios>Grado superior de DAM</estudios>
  </alumno>
</alumnos>
```

Si queremos insertar un nuevo elemento **nota** en el alumno María Palomita, la consulta XUpdate sería:

```
<xupdate:modifications version='1.0'
  xmlns:xupdate="http://www.xmldb.org/xupdate">
  <xupdate:insert-after select="/alumnos/alumno/nombre[.='María Palomita']">
    <xupdate:element name='nota'>Sobresaliente</xupdate:element>
  </xupdate:insert-after>
</xupdate:modifications>
```

Ejemplo de **xupdate:insert-after**

Esta consulta insertará el elemento **<nota>** después (**xupdate:insert-after**) del elemento **<nombre>** que tenga como valor 'María palomita'.

Realicemos una acción similar, pero insertando el elemento **antes (xupdate:insert-before)** del seleccionado con XPath. Para ello vamos insertar un elemento **<matrícula>**, antes del nodo **<nombre>** del elemento que estén realizando los estudios de DAM.

```
<xupdate:modifications version='1.0'  
    xmlns:xupdate="http://www.xmldb.org/xupdate">  
    <xupdate:insert-before select="/alumnos/alumno[estudios='Grado superior  
DAM'] /nombre">  
        <xupdate:element name='matricula'>DS0001</xupdate:element>  
    </xupdate:insert-before>  
</xupdate:modifications>
```

Ejemplo de **xupdate:insert-before**

El resultado de estas modificaciones sería:

```
<alumnos>  
    <alumno>  
        <matricula>DS0001</matricula>  
        <!-- Inserción -->  
        <nombre>Juan Palomo</nombre>  
        <estudios>Grado superior de DAM</estudios>  
    </alumno>  
    <alumno>  
        <nombre>María Palomita</nombre>  
        <nota>Sobresaliente</nota>  
        <!-- Inserción -->  
        <estudios>Grado superior de DAW</estudios>  
    </alumno>  
    <alumno>  
        <nombre>Roberto Pomba</nombre>  
        <estudios>Grado superior de DAM</estudios>  
    </alumno>  
</alumnos>
```

Resultado del documento después de aplicar los dos ejemplos planteados

Otra forma de realizar la inserción podía ser utilizando, **update:append** sobre el elemento alumno que insertará el elemento al final como último hermano. Igualmente podemos añadir atributos con **update:attribute**.

Modificar un elemento

Partiendo del ejemplo anterior, veamos cómo modificar un dato del XML con **xupdate:update**. Con la siguiente sentencia vamos a cambiar el nombre de los estudios del alumno "Juan Palomo" por "Graduado Superior en Desarrollo de Aplicaciones Multiplataforma."

```
<xupdate:modifications version='1.0' xmlns:xupdate="http://www.xmldb.org/xupdate">  
    <xupdate:update select="/alumnos/alumno[nombre='Juan Palomo']/estudios">  
        Graduado Superior en Desarrollo de Aplicaciones Multiplataforma  
    </xupdate:update>  
</xupdate:modifications>
```

Ejemplo de **xupdate:update**

El resultado sería:

```
<alumnos>
    <alumno>
        <matricula>DS0001</matricula>
        <!-- Inserción -->
        <nombre>Juan Palomo</nombre>
        <estudios>Graduado Superior en Desarrollo de Aplicaciones
Multiplataforma</estudios>
        <!-- Modificación -->
    </alumno>
    <alumno>
        <nombre>María Palomita</nombre>
        <nota>Sobresaliente</nota>
        <!-- Inserción -->
        <estudios>Grado superior de DAW</estudios>
    </alumno>
    <alumno>
        <nombre>Roberto Pomba</nombre>
        <estudios>Grado superior de DAM</estudios>
    </alumno>
</alumnos>
```

Resultado del documento después de modificar el elemento.

Borrar un elemento

Para eliminar elementos podemos usar **xupdate:remove**. Por ejemplo si queremos borrar el primer elemento del XML:

```
<xupdate:modifications version='1.0' xmlns:xupdate="http://www.xmldb.org/xupdate">
    <xupdate:remove select='/alumnos/alumno[1]' />
</xupdate:modifications>
```

Ejemplos de **xupdate:remove** para borrar un elemento

El resultado sería:

```
<alumnos>
    <alumno>
        <nombre>María Palomita</nombre>
        <nota>Sobresaliente</nota>
        <!-- Inserción -->
        <estudios>Grado superior de DAW</estudios>
    </alumno>
    <alumno>
        <nombre>Roberto Pomba</nombre>
        <estudios>Grado superior de DAM</estudios>
    </alumno>
</alumnos>
```

Resultado después de eliminar el primer elemento.

Renombrar elementos

En algunos casos, no solo queremos modificar los datos o valores guardados en los elementos (nodos) de nuestros documentos. Sino que queremos cambiar el nombre (la clave) de dicho elemento, para ello podemos usar **xupdate:rename**.

Por ejemplo si queremos cambiar el nombre del nodo **<alumno>** por **<estudiante>**, pondremos

```
<xupdate:modifications version='1.0' xmlns:xupdate="http://www.xmldb.org/xupdate">
    <xupdate:rename select="/alumnos/alumno/"> Escritor </xupdate:rename>
</xupdate:modifications>
```

Ejemplo de **xupdate:rename**

Por lo que el resultado final sería:

```
<alumnos>
    <estudiante>
        <!-- Renombrado del nodo -->
        <nombre>María Palomita</nombre>
        <nota>Sobresaliente</nota>
        <estudios>Grado superior de DAW</estudios>
    </estudiante>
    <!-- Renombrado del nodo -->
    <estudiante>
        <!-- Renombrado del nodo -->
        <nombre>Roberto Pomba</nombre>
        <estudios>Grado superior de DAM</estudios>
    </estudiante>
    <!-- Renombrado del nodo -->
</alumnos>
```

Resultado de **xupdate:rename** sobre el nodo alumno

XQuery Update Facility

La principal carencia del lenguaje **XQuery** es la falta de definición de sentencias para la actualización de datos. El diseño de XQuery se ha realizado teniendo en cuenta esta necesidad, y **XQuery Update Facility 1.0** es una extensión del estándar para cubrirla.

XQuery Update Facility 1.0 divide las expresiones en tres tipos:

non-updating expression

Expresión XQuery normal.

Basic updating expression

Sentencia *insert*, *delete*, *replace*, *rename* o llamada a función de actualización.

Updating expression

Es una *Basic updating expression* o cualquier otra expresión que contiene una sentencia de actualización (por ejemplo una sentencia *FLWOR* con una sentencia *update* en el *return*).

Documentación

XQuery Update Facility 1.0. <https://www.w3.org/TR/xquery-update-10/>

Insert

Esta sentencia **inserta** copias de 0 o más nodos especificados en *origen* en las posiciones indicadas en *destino*.

Tanto *origen* como *destino* no pueden ser expresiones de actualización.

```
insert---node----"origen"----after-----"destino"
      | -nodes-|           | --before-----|
                  | -----into--|
                  | --as---first---|
                  | -last--|
```

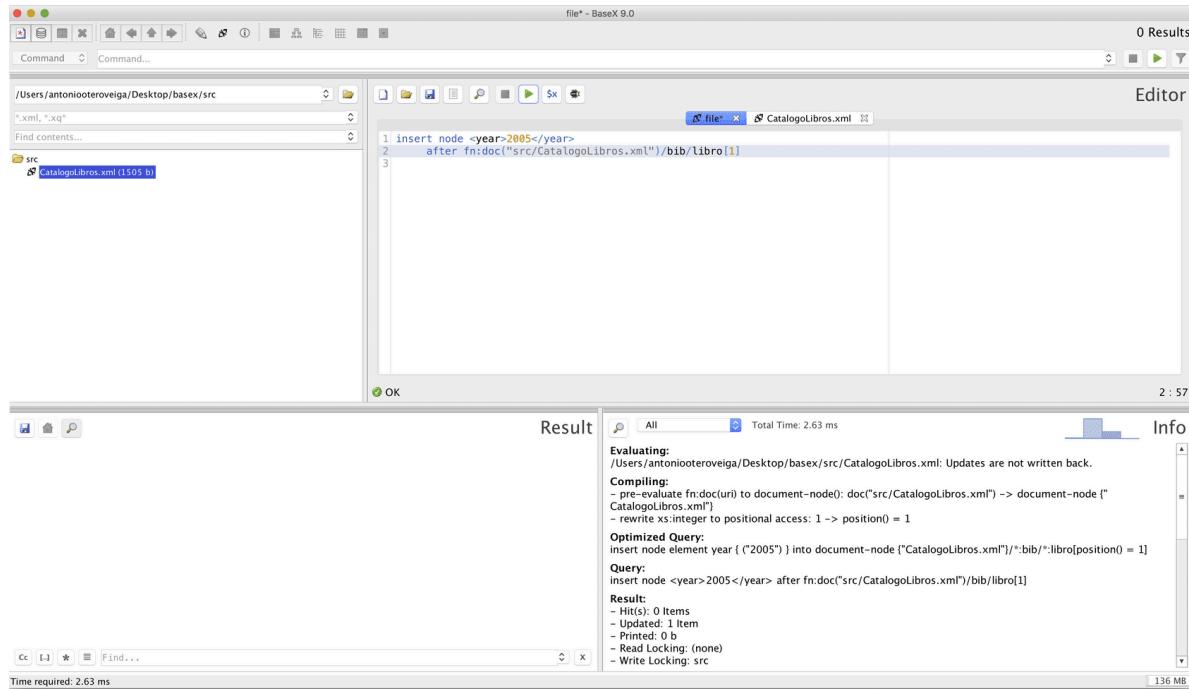
Sintaxis

Origen especifica los datos que vamos a insertar, bien nodos de una expresión bien un literal, *destino* especifica dónde queremos insertar los nodos de origen.

- **Node/nodes** se puede usar indistintamente.
- Con **before**, el nodo se inserta como *preceding-sibling* del nodo destino, si se insertan varios su orden es mantenido.
- Con **after**, el nodo se inserta como *following-sibling* del nodo destino, si se insertan varios su orden es mantenido.
- Con **into** el nodo es insertado como hijo del nodo destino, si se insertan varios su orden es mantenido.
- La posición de inserción será dependiente de la implementación.
- Si se añade **as first**, el nodo(s) serán los primeros hijos del nodo destino.
- Si se añade **as last**, el nodo(s) serán los últimos hijos del nodo destino.

```
insert node <year>2005</year>
       after fn:doc("bib.xml")/books/book[1]/publisher
insert node $new-police-report
       as last into fn:doc("insurance.xml")/policies
                  /policy[id = $pid]
                  /driver[license = $license]
                  /accident[date = $accdate]
                  /police-reports
```

Ejemplos insert



Ejemplo de aplicación con BaseX

Delete

Con esta sentencia borramos 0 o más nodos.

```
delete---node----"destino"
| -nodes - |
```

Sintaxis

```
delete node fn:doc("bib.xml")/books/book[1]/author[last()]
delete nodes /email/message
[fn:currentDate() - date > xs:dayTimeDuration("P365D")]
```

Ejemplos *delete*

Destino debe de ser una expresión de no-actualización y que tenga como resultado una secuencia de nodos.

Replace

Con esta sentencia reemplazamos una secuencia de 0 o más nodos. En este caso debemos tener en cuenta que podemos reemplazar bien el contenido del nodo bien el nodo mismo.

```
replace---node---"destino"--with---"expr.valor"
| -value of - |
```

Sintaxis

```
replace node fn:doc("bib.xml")/books/book[1]/publisher  
with fn:doc("bib.xml")/books/book[2]/publisher  
replace value of node fn:doc("bib.xml")/books/book[1]/price  
with fn:doc("bib.xml")/books/book[1]/price * 1.
```

Ejemplo *replace*

Tanto *destino* como *exp.valor* no pueden ser expresiones de actualización. Si no especificamos **value of** vamos a reemplazar un nodo, por lo que el nuevo nodo va a ocupar su posición en la jerarquía; esto implica que un nodo elemento puede ser reemplazado por otro nodo elemento, o un nodo atributo por un nodo atributo etc. Pero no podemos cambiar el tipo de nodo.

Rename

Se utiliza esta sentencia para cambiar la propiedad *name* con un nuevo valor.

```
rename node "destino" as "nombre"
```

Sintaxis

```
rename node fn:doc("bib.xml")/books/book[1]/author[1] as "principal-author"
```

Ejemplo *rename*

Despedida

Lecturas recomendadas

A lo largo de esta unidad has podido ver algunos ejemplos de cláusulas XQuery y el uso de la herramienta BaseX.

Si quieres ver más ejemplos puedes visitar los siguientes enlaces.

Introducción a XQuery con ejemplos

Uso de BaseX y ejemplos de aplicación paso a paso.

<https://www.adictosaltrabajo.com/tutoriales/introduccion-x-query/>

Ejercicios prácticos de XQuery

Resolución de casos de búsqueda en XQuery.

<https://www.tutorialspoint.com/xquery/index.htm>

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

En esta lección hemos visto los conceptos sobre los que se basa el almacenamiento de datos con el uso de XML. Desde sus herramientas a los lenguajes que utilizamos para su manipulación o consulta.

Con XQuery hemos realizado consultas en bases de datos XML, pudiendo filtrar los resultados como hacemos en bases de datos relacionales con SQL. Además gracias a Xupdate, también podemos modificar nuestros documentos, sin tener que utilizar APIs de lenguajes externos.

MP_0373. Lenguajes de marcas y sistemas de gestión de información

UF5. Almacenamiento y manipulación de la información

5.4. Almacenamiento XML nativo



Índice

Objetivos	3
XML Nativo	4
Conceptos	4
Características	4
Aplicaciones	5
Manejo de colecciones	6
Base de datos clásica frente a base de datos nativa	6
Áreas de aplicación	7
Despedida	8
Resumen.....	8

Objetivos

Con esta unidad perseguimos los siguientes objetivos:

- Conocer los conceptos fundamentales de las bases de datos XML nativas y sus características.
- Conocer cómo es el manejo de colecciones en bases de datos XML nativas.
- Conocer las diferencias con otros sistemas.
- Conocer las áreas de aplicación.

XML Nativo

Conceptos

XML está provocando la aparición de una nueva generación de bases de datos.

Las bases de datos XML nativas (NXD) son muy distintas a las relacionales, ya que tienen capacidades para soportar el almacenamiento de documentos XML, aunque los datos almacenados se derivan al formato relacional usando tablas, registros y columnas, o en otras ocasiones almacenando todo el documento en formato *Binary Large Object* (BLOB).

La cualidad principal de las bases de datos no nativas XML que trabajan con documentos XML es la capacidad de obtener resultados de las consultas en formato XML; es por ello que dichas bases de datos pertenecen a la categoría de "XML-Enabled database".

Un **SGBD XML-Enabled** es un SGBD tradicional con capacidad para el tratamiento y obtención de documentos XML.

Hay que tener en cuenta que los grandes fabricantes de SGBD están aumentando las capacidades para la inserción y el tratamiento de documentos XML dentro de sus productos, incluyendo características de SGBD XML nativas, por lo que en un futuro próximo es posible que la frontera entre estos dos tipos de gestores quede diluida.

Las bases de datos más relacionadas con XML son las "Native XML Database" NXD o bases de datos XML nativas.

Características

¿En qué consisten las bases de datos nativas en XML?

Modelo lógico

Las bases de datos nativas definen un modelo lógico para el documento XML, almacenan y recuperan documentos de la misma manera que los XML. Por lo tanto, el **modelo debe incluir atributos como PCDATA y documentos en orden**.

Ejemplos de estos modelos son: XPath, XML Infoset y modelos que implican DOM y SAX 1.0.

Unidad de información

Tienen un documento de **XML como unidad fundamental del almacenamiento**, como si en una base de datos relacional se tiene una fila en una tabla como unidad fundamental del almacenamiento lógico.

Estructura física

No se requiere tener ningún modelo de almacenamiento físico en particular. Por ejemplo, el documento puede ser creado en una base de datos relacional, jerárquica, u orientada a objetos, o utilizar un formato de almacenamiento propietario tal como archivos indexados y comprimidos.

Aplicaciones

Existen diversos productos que brindan diferentes funcionalidades para las bases de datos nativas en XML, pero generalmente tienen las siguientes características.

Procesamiento de datos

El procesamiento de datos en este tipo de bases de datos es costoso y muy poco amigable, esto es debido al formato jerárquico en el que se almacena la información.

En la mayoría de SGBD XML nativas la recuperación del documento es completa, su tratamiento se realiza a través del API de referencias SAX o DOM y posteriormente se vuelve a almacenar el documento en el repositorio. Esto ocurre porque aún no hay un lenguaje estándar que permita de forma sencilla y procedimental la actualización, inserción o eliminación de elementos de un documento XML.

Xupdate es un lenguaje que permite realizar actualizaciones en un documento XML, pero aún no es un estándar y muchos gestores de este tipo de bases de datos no lo soportan.

Almacenamiento

Una base de datos nativa en XML almacena la información en documentos XML. Este tipo de bases de datos tienen repositorios con un formato "tipo XML", como puede ser DOM o InfoSet. En este mismo "repositorio" (paquete de archivos) se almacenan los índices que se generan por cada documento XML almacenado.

Búsquedas

Este tipo de bases de datos no utiliza SQL como lenguaje de consulta. En lugar de ello utilizan **Xpath**. Algunas bases de datos permiten seleccionar los elementos que deberán tener índices, mientras que otras bases de datos indexan todo el contenido del documento.

El problema que tienen las búsquedas en este tipo de bases de datos es que no permiten realizar búsquedas muy complicadas, como por ejemplo ordenamiento y *cross join*, debido a que Xpath no fue creado realmente para búsquedas en bases de datos, sino simplemente para búsquedas en un solo documento.

Muchas bases de datos permiten realizar búsquedas utilizando la tecnología *Full-Text Search*, de esta manera se puede agilizar la búsqueda de datos en los documentos XML.

Manejo de colecciones

Se espera que en un futuro los **esquemas propuestos por el W3C** se impongan como estándar para las NXD.

Las **NXD** manejan colecciones de documentos, permitiendo que se consulten y se manejen como un conjunto. Este método es muy similar al concepto relacional de una tabla.

Las **NXD se diferencian del concepto de las tablas relacionales en que no todas las bases de datos nativas de XML requieren un esquema para estar asociado a una colección**. Esto significa que se puede almacenar cualquier documento de XML en la colección, sin importar el esquema.

Al hacer esto se pueden construir consultas a través de todos los documentos en la colección. Las NXD que utilizan estas funciones se llaman esquema-independientes. Tener colecciones esquema-independientes del documento da a la base de datos mucha flexibilidad y hace que el desarrollo de las aplicaciones sea más fácil.

Si se necesita una estructura de esquemas fuerte, entonces hay que asegurarse de que se utiliza una NXD que soporte esquemas o se deberá encontrar otra manera de almacenar los datos del XML.

Base de datos clásica frente a base de datos nativa

Veamos las diferencias entre una base de datos clásica y una base de datos nativa.

1. Una base de datos XML nativa **provee de su propio modelo de datos**.
2. Un SGBD XML-Enabled solo puede manejar y almacenar los documentos que encajan dentro del modelo definido para ellos, mientras que un **SGMD XML nativo debe manejar todos los tipos de documentos posibles**.
3. **Todas las bases de datos relacionales están centradas en los datos**, pues lo que ellas almacenan en sus campos son datos simples, más conocidos como datos atómicos.
4. Una base de datos nativa en XML **no posee campos, ni almacena datos atómicos**, lo que almacena son documentos XML, por lo tanto a este tipo de bases de datos se les denomina bases de datos centradas en documentos.

Áreas de aplicación

En general, cualquier NXD es una excelente herramienta para almacenar documentos orientados a datos. Básicamente, si la información es representada por XML una NXD probablemente es una buena solución.

Una NXD puede almacenar cualquier tipo de información XML, pero probablemente no sería la mejor herramienta para algo como un sistema de contabilidad, donde los datos están muy bien definidos y son rígidos.

Algunas áreas potenciales de aplicación podrían ser:

- Portales de información corporativa.
- Información de catálogos.
- BD en partes de manufactura.
- Información médica.
- BD personalizadas.

Despedida

Resumen

En esta lección hemos visto una introducción a la importancia que tendrán en un futuro las bases de datos XML nativas.

XML está provocando la aparición de una nueva generación de bases de datos, que aunque en la actualidad se encuentran en una fase de investigación y desarrollo, en breve tiempo se convertirán en una buena alternativa a las bases de datos tradicionales. Las bases de datos más relacionadas con XML son las "Native XML Database", NXD o bases de datos XML nativas.

6.1. Descripción y características de la sindicación de contenidos



Índice

Objetivos	3
Sindicación de contenidos.....	4
Introducción	4
Ventajas de la sindicación de contenidos	4
Redifusión en la web	5
Estándares y formatos de redifusión.....	6
Despedida	7
Resumen.....	7

Objetivos

En este punto vamos a definir el concepto de sindicación de contenidos con XML.

- Conocer en qué consiste la sindicación de contenidos web.
- Conocer las características de la sindicación de contenidos.
- Entender el concepto de **redifusión**.
- Conocer las ventajas de la redifusión de contenidos.

Sindicación de contenidos

Introducción

La sindicación de contenidos, desde un punto de vista web, permite que un sitio utilice servicios u ofrezca contenidos de otra web distinta.

A estos servicios se les llama "feed" o "canales de contenido". Esta **redifusión** se realiza normalmente bajo una licencia.

Actualmente **la redifusión web se centra en ofrecer contenido desde una fuente web**, cuyo origen está en una página web, con el fin de proporcionar a los usuarios la actualización de este.

Estas fuentes **suelen estar codificadas en XML** o en otro lenguaje que pueda ser transportado con el protocolo HTTP.

Podemos hablar de la redifusión como un **flujo de información** que tiene su origen en unos ficheros localizados en un servidor, al que otras aplicaciones pueden acceder para obtener datos.

Para leer una fuente o canal, hay que suscribirse a ella utilizando un agregador. Este agregador es un software que permite suscribirse a fuentes web mostrando al suscriptor las modificaciones que han tenido lugar en los contenidos publicados por el canal de contenidos.

Ventajas de la sindicación de contenidos

Las ventajas de la sindicación de contenidos son:

1. Favorece el **aumento del tráfico** de un sitio web.
2. Favorece el **posicionamiento** del sitio web.
3. Permite establecer **relaciones** entre distintos sitios web.
4. Permite a los usuarios establecer una serie de características a los servicios web, ofreciendo, con aplicaciones o tecnologías adicionales, sistemas de alerta o notificaciones.

Redifusión en la web

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

No se pueden entender las herramientas sociales y **el acceso a la información** a través de los blogs y editores sin la sindicación de contenidos.

El hecho de que los autores (ya sean editoriales, periódicos o personas expertas) puedan sindicar contenidos etiquetados en sus herramientas colaborativas, sirve al resto de los usuarios para estar al día en todas las fotografías, textos, vídeos, elementos, etc. que suben a la red los millones de usuarios, sobre temas infinitos.

La sindicación de contenidos es una forma de ofrecer contenidos propios para que sean mostrados en otras páginas web de forma integrada, aumentando el valor de la página que muestra el contenido y la de la fuente de origen, ya que normalmente la redifusión web siempre enlaza con los contenidos originales.

Es importante recordar que la redifusión de contenidos web puede aplicarse a todo tipo de contenidos, es decir, texto, audio, vídeos e imágenes. En cuanto a los suscriptores, la redifusión de contenidos permite la actualización profesional.

Ejemplos de emisores de contenido:

- **Periodista Digital**: el periódico de periódicos. <http://feeds.feedburner.com/PDPortada>
- **El País**, titulares de El País. <https://servicios.elpais.com/rss/>
- Diario **El Mundo-RSS**. El diario El Mundo publica los titulares de sus noticias en formato RSS (del inglés Really Simple Syndication, sindicación realmente sencilla).
<http://rss.elmundo.es/rss/>
- Universia **Noticias universitarias** con muchos canales especializados: estudiantes, investigación, etc. http://rss.universia.net/ES/Noticias_ES/es/
- **Libertad Digital** con muchos canales especializados de noticias.
<https://www.libertaddigital.com/rss/>
- **Iblnews**, agencia de información en español. <https://iblnews.com/>
- **Octeto**. Excelentes noticias y una de las primeras apuestas RSS en español (Universidad Jaume I de Castelló). <http://cent.uji.es/octeto/rss.xml>
- **Barrapunto**. El RSS de la comunidad hispana de software libre.
<http://barrapunto.com/index.rss>

Estándares y formatos de redifusión

Como decíamos en el anterior punto, publicar en la web puede ser visto como un flujo de información. Para suministrar información un canal debe incluir en su cabecera un enlace al canal de contenidos.

Para esto, dependiendo de si el canal está realizado con un estándar RSS o con uno **Atom**, se utiliza, respectivamente, este tipo de estructura:

```
<link rel="alternate" type="application/rss+xml"
title="Noticia en RSS" href="http://www.rss.com/fichero.rss" />

<link rel="alternate" type="application/atom+xml"
title="Noticia en ATOM" href="http://www.atom.com/fichero.atom" />
```

Actualmente existen dos principales formatos de redifusión o *feed*:

- RSS
- Atom

Ambos formatos están escritos en **XML** y tienen la misma función. Estos formatos surgen de la necesidad de poder combinar información de varias fuentes, tanto de manera manual como automática.

Con un lector **RSS/Atom** puedes realizar esta tarea, puedes "suscribirte" a diferentes blogs o páginas que deseas seguir y, desde un lugar centralizado, consultar las noticias y actualizaciones de estos sin necesidad de tener que visualizar cada una de las páginas por separado.

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

- El concepto *feed* es un medio de redifusión web, que por lo general es utilizado especialmente en los blogs.
- Los dos formatos más utilizados son el RSS y el Atom. Se encuentran escritos en formato XML.
- XML ofrece la posibilidad de que el contenido difundido sea procesado por aplicaciones de manera fácil.
- Existen muchas aplicaciones que nos permiten leer y consumir *feeds*: lectores RSS, agregadores, etc.
- Para consumir un *feed* nos debemos suscribir a él. Normalmente es indicarle a la aplicación que vamos a utilizar como lector la URL del *feed* en formato RSS o Atom.
- Las aplicaciones que se pueden utilizar para leer *feeds* pueden ser web, estar instaladas en nuestro ordenador, en nuestro correo, en nuestro móvil, etc.

6.2. Tecnologías de creación de canales de contenidos



Índice

Objetivos	3
Introducción.....	4
Estructura de los canales de contenidos	6
La estructura	6
La Estructura II.....	7
Formatos	8
RSS.....	9
RSS	9
Manipulación de documentos RSS	10
Versiones RSS	11
Atom.....	12
Atom	12
Diferencias entre RSS y Atom.....	12
Despedida	14
Resumen.....	14

Objetivos

En esta lección vamos a definir el concepto de sindicación de contenidos con XML.

- Conocer la estructura de los canales de contenidos.
- Conocer los elementos básicos de los canales de contenidos.

Introducción

Construir un canal de contenido no es más que crear un fichero que, dependiendo del formato que elijamos, tendrá una extensión RSS o Atom y, por supuesto, estará basado en XML.

Este fichero debe publicarse en uno de los directorios del sitio web desde el que se ofrecerá.

Este fichero consta de los siguientes elementos básicos:

- **Declaración** del documento XML.
- **Definición** de la codificación empleada en el documento, la más utilizada UTF-8 (Unicode Transformation Format de 8 bits).
- Un canal en el que se determina el sitio web asociado a la fuente a la que hace referencia el fichero. Estará formado por:
 - **Secciones**. Hacen referencia a la web que contiene uno de los servicios que se van a ofrecer. Pueden ser múltiples secciones, lo que hace que un canal de contenido pueda tener un tamaño considerable si contiene un gran número de enlaces independientes.

```
<rss xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:taxo="http://purl.org/rss/1.0/modules/taxonomy/" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:itunes="http://www.itunes.com/dtds/podcast-1.0.dtd" version="2.0">
<channel>
    <title>España</title>
    <link>http://www.abc.es</link>
    <description>España</description>
    <language>es-es</language>
    <pubDate>Mon, 30 Abr 2018 12:20:28 GMT</pubDate>
    <lastBuildDate>Mon, 30 Abr 2018 12:20:28 GMT</lastBuildDate>
    <ttl>2</ttl>
    <item>
        <title>
            El mensaje viral de una deportista de élite una niña de cinco años criticada por
            jugar al fútbol siendo mujer
        </title>
        <link>
            http://www.abc.es/espagna/galicia/abci-mensaje-viral-deportista-elite-nina-cinco-anos-
            criticada-jugar-futbol-siendo-mujer-201804301217_noticia.html
        </link>
        <description>
            «Mamá, hoy un niño del cole me dijo que no podía jugar al fútbol porque las niñas
no juegan al fútbol». De este modo explicaba Lola, una niña de cinco años, a su madre las críticas
que había recibido de un compañero de colegio por querer practicar este deporte siendo mujer. Fue el
propio club del que es miembro la joven, la Sociedad Deportiva Esclatitud (en Padrón, La Coruña), el
que hizo público este mensaje advirtiendo de que, aunque «son cosas de niños», detrás de estas
palabras «están las actitudes machistas de nuestra sociedad». «Desde la SD Esclatitud estamos muy
orgullosos de Lola y del resto de niñas que tenemos, si tenéis niñas que quieran jugar estamos
abiertos a todas ellas, porque sin las mujeres nuestro mundo se para», afirmaron a través de sus
redes sociales, desatando una ola de mensajes de apoyo a la que también se ha sumado la futbolista
gallega Verónica Boquete, capitana de la selección española femenina y embajadora de la Uefa para el
desarrollo del deporte femenino. «Aquí Vero Boquete. Mujer y futbolista profesional. El deporte es
de todos y para todos, y al fútbol juega quien quiere. Aunque cueste hacérselo entender a algunos y
algunas por la educación machista que frena nuestro avance cara la igualdad, lo conseguiremos»,
```

señaló la mediapunta del Paris Saint Germain en el muro de Facebook del club coruñés, al tiempo que aseguró que «en los últimos años abrimos muchas puertas». «Ahora, Lola, puedes soñar y hacer lo que quieras. Disfruta del fútbol que para eso está. Un abrazo», concluyó la deportista natural de Santiago de Compostela. El club coruñés agradeció a la futbolista profesional su mensaje de ánimo y se comprometieron a hacérselo llegar a la menor. «Le explicaremos quién eres y a donde has llegado como futbolista y como persona. Gracias por invitar a Lola a soñar», manifestaron.

```
</description>
<category domain="España">España</category>
<category domain="Galicia">Galicia</category>
<pubDate>Mon, 30 Abr 2018 12:17:07 GMT</pubDate>
<guid>
http://www.abc.es/espana/galicia/abci-mensaje-viral-deportista-elite-nina-cinco-anos-
criticada-jugar-futbol-siendo-mujer-201804301217_noticia.html
</guid>
<dc:creator>(abc)</dc:creator>
</item>
<item>
<title>
«Las gachas y el zarajo pueden dar mucho juego en El Celler de Can Roca»
</title>
<link>
http://www.abc.es/espana/castilla-la-mancha/abci-gachas-y-zarajo-pueden-mucho-juego-
celler-roca-201804301211_noticia.html
</link>
<description>
Con 25 años recién cumplidos, José Martínez Ortiz (Villamayor de Santiago, Cuenca), estudiante de cocina en la Escuela Superior de Hostelería y Turismo de Madrid, en breve cumplirá uno de sus sueños: trabajar en uno de los grandes templos de la cocina a nivel mundial, El Celler de Can Roca, en Gerona. Y lo hará gracias a una beca del BBVA, que le permitirá formarse durante cuatro meses junto a los hermanos Roca, una experiencia que espera le sirva para dar un gran impulso a su carrera en el mundo de la gastronomía. ¿Qué supone para usted la beca del BBVA que acaba de conseguir para formarse en El Celler de Can Roca? Es un sueño. Aún no me creo que vaya a trabajar con gente tan grande y de tanto prestigio como los hermanos Roca. Pero también supone una oportunidad única que me cambiará la vida, al menos profesionalmente, ya que durante cuatro meses allí, imaginate lo que puedo aprender.
</description>
<category domain="España">España</category>
<category domain="Castilla La Mancha">Castilla La Mancha</category>
<pubDate>Mon, 30 Abr 2018 12:11:43 GMT</pubDate>
<guid>
http://www.abc.es/espana/castilla-la-mancha/abci-gachas-y-zarajo-pueden-mucho-juego-
celler-roca-201804301211_noticia.html
</guid>
<dc:creator>(abc)</dc:creator>
</item>
</channel>
```

Ejemplo de archivo RSS.

Estructura de los canales de contenidos

La estructura

Para entender la estructura vamos a construir un canal de contenidos con el estándar Atom.

Para ello vamos a seguir los siguientes pasos:

- Crear un fichero con extensión Atom; podemos utilizar un editor de texto plano para ello.
- En la primera línea haremos la declaración XML y estableceremos la codificación que se va a utilizar, en este caso UTF-8.
- Después de esto definiremos el canal, el estándar de Atom y el lenguaje utilizado en el fichero. Utilizaremos para ello un elemento "feed".

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.canalatom.org/Atom">

  <title>Ejemplo Feed</title>
  <link href="http://ejemploweb.com/" />
  <updated>2016-12-13T18:30:02Z</updated>
  <author>
    <name>JC</name>
  </author>
  <id>urn:uuid:644a76c80-d399-12d9-b93C-0004439e0af6</id>

  <entry>
    <title>Esta es la primera entrada</title>
    <link href="http://ejemploweb.com/2013/64/12/atom03" />
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2016-12-13T18:30:02Z</updated>
    <summary>Este es el resumen de la entrada <summary>
  </entry>

</feed>
```

Ejemplo.

Para definir el canal, el elemento "feed" debe incluir estos elementos:

Title

Título de la web a la que hace referencia.

Id

Identificador de canal único.

Link

Enlaces a la fuente de la sección ("rel"="alternate").

Update

Fecha y hora de actualización en formato CCYY-MM-DDTHH:MM:SSZ siendo T el separador de la fecha y Z el sistema de tiempo Greenwich.

Author

Autor del enlace.

Summary

Resumen del contenido del enlace.

Entry

Sección del canal.

Dentro de **entry** tendremos los elementos que definen cada una de las entradas, las cuales volverán a estar compuestas por los elementos: **<title>**, **<id>**, **<link>**, **<update>**, **<autor>** y **<summary>**.

La Estructura II

```
<?xml version="1.0" encoding="utf-8"?> 1
<feed xmlns="http://www.canalatom.org/Atom"> 2
    <title>Ejemplo Feed</title> 3
    <link href="http://ejemploweb.com/"> 4
    <updated>2016-12-13T18:30:02Z</updated> 5
    <author>
        <name>JC</name> 6
    </author>
    <id>urn:uuid:644a76c80-d399-12d9-b93C-0004439e0af6</id> 7
    <entry>
        <title>Esta es la primera entrada</title>
        <link href="http://ejemploweb.com/2013/64/12/atom03"/>
        <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id> 8
        <updated>2016-12-13T18:30:02Z</updated>
        <summary>Este es el resumen de la entrada <summary>
    </entry>
</feed>
```

1. Declaración

Declaración del documento XML.

2. Canal

Determina el sitio web asociado a la fuente web a la que hace referencia el fichero.

3. Title

Título de la web.

4. Link

Enlaces que definirán al canal:

- Al propio fichero *.atom*.
- A otro fichero de la web que oferta ese canal.

5. Update

Fecha y hora de actualización en formato CCYY-MM-DDTHH:MM:SSZ, siendo T el separador de la fecha y Z el sistema de tiempo Greenwich.

6. Author

Autor o autores de la entrada.

7. Id

Identificador de canal (URL).

8. Entry

Sección del canal. A su vez, estará compuesto por los elementos que definirán cada entrada.

Formatos

La **sindicación de contenidos** permite que el usuario de un sitio web sea capaz de **acceder a una información** o servicio que se encuentra en un sitio web diferente.

Anteriormente, hemos visto que para ofrecer estos servicios de sindicación de contenidos hay que utilizar alguno de los estándares de sindicación basados en XML, como son **RSS** y **Atom**.

- **RSS (Really Simple Syndication o Rich Site Summary)** es el formato más extendido en Internet.
- **Atom (Atom Syndication Format)** fue desarrollado para evitar las limitaciones y los defectos de RSS.

Aunque Atom es más robusto que RSS, este último sigue siendo el estándar más utilizado. De tal modo que el término RSS generalmente se ha convertido en el común para referirse a todos los feeds, ya sean en el propio formato RSS o en Atom.

RSS

RSS

RSS es parte de la familia de los formatos XML dedicados a compartir información que se actualiza periódicamente entre sitios web.

Utilizado también en la conexión con sistemas de mensajería instantánea, ha sido desarrollado por varias organizaciones diferentes que han dado lugar a varios formatos.

Un usuario puede suscribirse a un *feed* con RSS y así disponer de las últimas actualizaciones realizadas en la página.

Utilizando *feeds* y agregadores podemos gestionar la visualización de esas novedades.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">
  <channel>
    <title>RSS Guía Fácil</title>
    <link>http://es.geocities.com/rss_guia_facil</link>
    <description>Guía fácil y sencilla sobre el formato RSS</description>

    <item>
      <title>Qué es el RSS</title>
      <link>http://es.geocities.com/rss_guia_facil/que_es_rss.html</link>
      <description>Visita la Guía Fácil del RSS para saber qué es el
      formato RSS</description>
    </item>
    <item>
      <title>Para qué sirve el RSS</title>
      <link>http://es.geocities.com/rss_guia_facil/para_que_sirve_rss.html
      </link>
      <description>Visita la Guía Fácil del RSS y conoce para qué sirve el
      formato RSS</description>
    </item>
    <item>
      <title>Cómo se usa el RSS</title>
      <link>http://es.geocities.com/rss_guia_facil/como_se_usa_rss.html
      </link>
      <description>Visita la Guía Fácil del RSS para saber cómo usar el
      formato RSS</description>
    </item>
  </channel>
</rss>
```

Ejemplo RSS.

XML RSS

Sencillo tutorial sobre la creación de documentos RSS.

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

Manipulación de documentos RSS

Vamos a ver cómo crear un canal de contenidos en el formato RSS. Lo primero que debemos hacer es crear un fichero con extensión .rss utilizando un editor especializado o con un editor de texto plano.

1. La primera línea del fichero siempre debe ser la declaración XML y la definición de la codificación que será utilizada en este.

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

2. Una vez descrita la primera línea, obligatoria, debemos determinar la versión de RSS que vamos a utilizar. En este caso vamos a usar la 2.0, por lo que la línea será:

```
<rss version="2.0">
```

3. Una vez hecho esto debemos definir el canal. Para ello utilizaremos el elemento ***channel***.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
  ....
</channel>
</rss>
```

Para la **definición del canal RSS** existen una serie de elementos imprescindibles que debemos utilizar:

- ***<title>*** - Título para la web.
- ***<link>*** - Dirección de la página web asociada al fichero RSS.
- ***<description>*** - Comentario para describir el sitio web.
- ***<language>*** - Lenguaje que se utiliza en el sitio web.
- ***<item>*** - Identifica una sección del canal.

4. Una vez hayamos completado estos elementos, debemos saber que por cada uno de los ***<item>*** existen otros elementos **imprescindibles**:

```
<title>Canal 1 Ejemplo</title>
<link>http://www.canalejemplo.com</link>
<description>Esta es la descripción del contenido del canal</description>
```

Elementos imprescindibles de cada bloque ***<item>***.

5. Una vez vistos los elementos que conforman el documento RSS vamos a ver un ejemplo:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
    <title>Canal 1 Ejemplo</title>
    <link>http://www.canalejemplo.com</link>
    <description>Esta es la descripción del contenido del canal</description>
    <item>
        <title>Primer item ejemplo</title>
        <link>http://www.canalejemplo.com/primer_item</link>
        <description>Esta es la descripción del primer item</description>
    </item>
    <item>
        <title>Segundo item ejemplo</title>
        <link>http://www.canalejemplo.com/segundo_item</link>
        <description>Esta es la descripción del segundo item</description>
    </item>
</channel>
</rss>
```

Versiones RSS

RSS 0.90

Creado por la empresa Netscape en 1999 forma un estándar que se basa en la especificación RDF de metadatos.

RDF es un modelo estándar para el intercambio de datos en la web que facilita la fusión de datos, incluso si los esquemas subyacentes tienen distinta estructura.

RSS 0.91

Esta es la versión simplificada de RSS 0.90 lanzada posteriormente por la misma empresa Netscape.

Utilizada para desarrollar blogs por la empresa UserLand Software, su desarrollo fue detenido por falta de éxito.

RSS 1.0

Basado en el estándar RSS 0.90 aporta una versión más estable y permite definir una cantidad mayor de datos que el resto de versiones de RSS.

RSS 2.0

UserLand Software continuó el desarrollo abandonado de RSS 0.91 creando las versiones 0.92, 0.93 y 0.94. Debido a que estaban incompletos y no cumplían todas las normas XML, se creó el estándar RSS 2.0 como solución.

Atom

Atom

Atom es un estándar del grupo Atom Publishing Format and Protocol que se desarrolló como 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 incompatibilidad.

Atom no buscaba sustituir a los estándares existentes, sino que creó uno nuevo para convivir con ellos. Atom se caracteriza por ser muy flexible y permitir un mayor control sobre la cantidad de información que se representa en un agregador. Un agregador es una aplicación software que permite suscribirse a una fuente en formato RSS o Atom.

Especificación de Atom 1.0

Documento con las directrices para crear un *feed* de datos de producto XML en formato Atom 0.1. <https://support.google.com/merchants/answer/160593?hl=es>

Diferencias entre RSS y Atom

RSS es una estructura sencilla donde se almacena el contenido de una página web. Al igual que con Atom, no necesita de un navegador, ya que podemos utilizar herramientas como *readers* o agregadores.

Estructura de documento RSS:

- Archivo XML contenedor del RSS para el formato 2.0.
- Por lo menos debe contener un canal.
- El canal ofrece algunos artículos o datos.

```
<?xml version="1.0" ?>
<rss version="2.0">
    <channel>
        <title>XUL and XML</title>
        <link>http://www.xul.fr/</link>
        <description>XML graphical interface etc...</description>
        <image>
            <url>http://www.xul.fr/xul-icon.gif</url>
            <link>http://www.xul.fr/index.html</link>
        </image>
        <item>
            <title>News of today</title>
            <link>http://www.xul.fr/xml-RSS.html</link>
            <description>All you need to know about RSS</description>
        </item>
        <item>
            <title>News of tomorrow</title>
            <link>http://www.xul.fr/xml-rdf.html</link>
            <description>And now, all about RDF</description>
        </item>
    </channel>
</rss>
```

Ejemplo RSS.

En el caso de los RSS se puede usar texto plano o HTML, pero no tenemos que indicar cuál de los dos se está usando, a **diferencia de Atom, que permite una gran variedad de tipos**, incluyendo texto plano (HTML, XHTML, XML). También podemos hacer referencias a contenido externo, ya sea de audio, vídeo o documentos como PDF.

Con RSS podemos definir un lenguaje, pero no podemos indicar diferentes idiomas para las entradas. En el caso de Atom utilizamos la normalización *xml:lang* para que sea posible identificar el idioma.

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">

<title>Example Feed</title>
<link href="http://example.org/" />

<updated>2003-12-13T18:30:02Z</updated>
<author>
<name>John Doe</name>
</author>
<id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>

<entry>
<title>Atom-Powered Robots Run Amok</title>
<link href="http://example.org/2003/12/13/atom03"/>
<id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>

<updated>2003-12-13T18:30:02Z</updated>
<summary>Some text.</summary>
</entry>

</feed>
```

Ejemplo Atom.

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

- El origen de los estándares de sindicación de contenidos.
- Las versiones de cada estándar.
- Construir un documento RSS.
- Las diferencias entre RSS y Atom.

6.3. Herramientas



Índice

Objetivos	3
Herramientas	4
Directorios de canales de contenidos	4
Lectores de feeds	4
Inoreader	5
Feedly	6
NewsBlur	7
G2Reader	8
CommaFeed	9
Emisores directorios de RSS	9
Validar un documento RSS/Atom	10
Agregadores	11
Despedida	12
Resumen	12

Objetivos

Con esta unidad perseguimos los siguientes objetivos:

- Conocer qué son los directorios de sindicación de contenidos.
- Conocer cómo validar un "feed".
- Conocer el término de agregador.
- Conocer los diferentes tipos de agregadores que existen y cuáles son los más utilizados.

Herramientas

Directorios de canales de contenidos

Los directorios de canales RSS proporcionan un contenido de calidad, pues estas herramientas filtran y evalúan los contenidos antes de ofrecerlos. Por lo tanto, agregan valor a la búsqueda de dichos contenidos en la web.

La **sindicación de contenidos** por medio de canales RSS es un método contemporáneo para distribuir contenidos web que posibilita a los usuarios **acceder solamente a los contenidos de su interés**, sin tener que navegar de una página a otra. El uso por parte de los generadores de un directorio facilita la difusión de estos contenidos y permite su acceso de manera más sencilla.

Los directorios de canal de contenidos permiten que **un fichero RSS/Atom** esté disponible para cualquiera, facilitando a los usuarios la búsqueda de **información**, ya que estructura los canales en una clasificación.

Para esto es necesario registrar el fichero RSS/Atom en un directorio RSS/Atom. Dado que el volumen de información disponible es tan grande, resulta muy interesante el uso de algún tipo de catálogo que permita clasificar y realizar búsquedas sobre los contenidos de distintos *feeds*. Para ello existen unas herramientas llamadas directorios de canales.

Muchos directorios se especializan por temáticas, otras por el canal de difusión; por ejemplo blogs, podcast, etc.

Lectores de feeds

Las aplicaciones consideradas como directorios o motores de búsqueda de RSS indican los ficheros RSS. Los contenidos frecuentemente se adquieren de *weblogs* o de sitios de noticias.

Los suscriptores indican el tema que les interesa y reciben notificación tan pronto como su solicitud coincide con los registros que se añaden al motor de búsqueda.

A continuación se muestran algunos de los lectores de feeds más populares.

Inoreader

Inoreader no es solo otro lector de RSS, es una comunidad de generadores de contenido. Tiene un modo de exploración de contenidos llamado *Discovery* y paquetes de suscripción generados por el usuario o canales de transmisión.

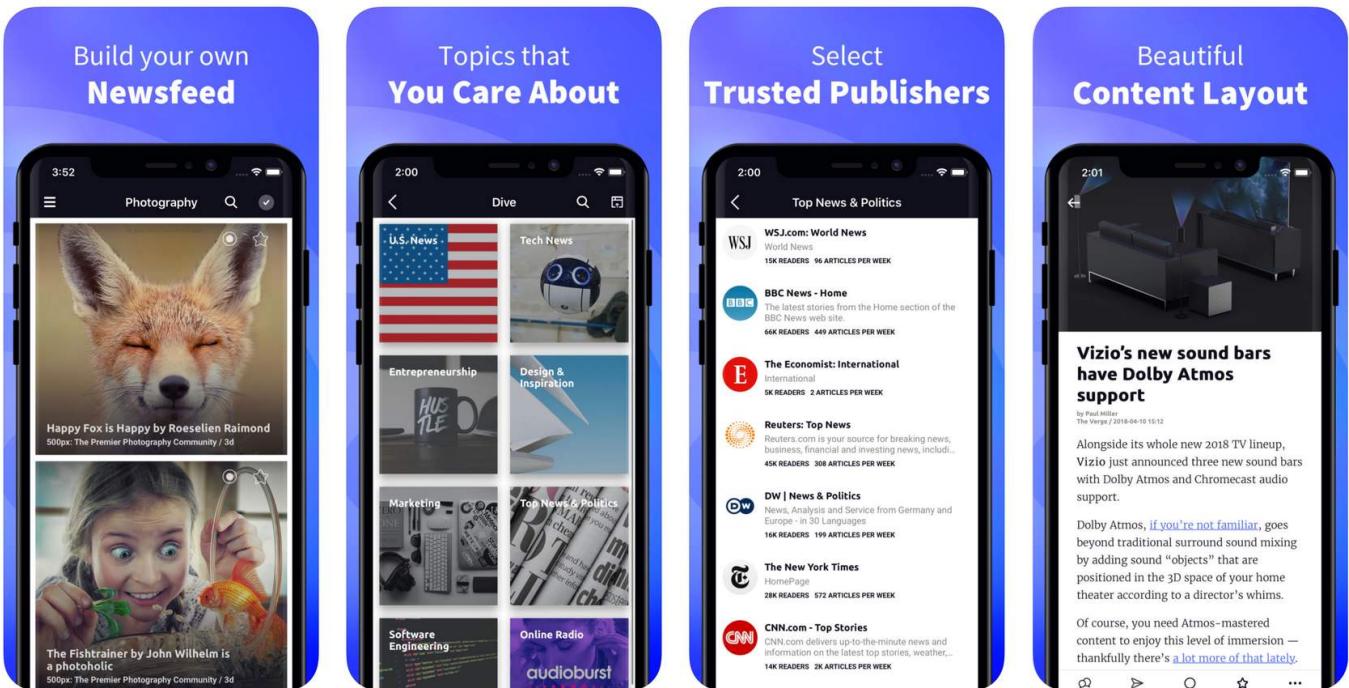


Imagen de [Apple Store](#).

<https://itunes.apple.com/us/app/inoreader/id892355414?platform=ipad&preserveScrollPosition=true&platform=ipad>

Inoreader

Lector RSS. <https://www.inoreader.com/>

Feedly

Lector inteligente que conecta con la información desde múltiples plataformas. Con Feedly puedes organizar fácilmente por categorías las publicaciones, blogs, canales de YouTube y más en un solo lugar.

The screenshot shows the Feedly web interface. On the left is a sidebar with the following structure:

- Home**
- Saved For Later**
- Shared Collections** (NEW)
- Add Content**
- All** (734 items)
 - Tech** (484 items)
 - MacRumors (35)
 - TechCrunch (295)
 - The Verge (154)
 - Decoration** (15 items)
 - Fresh Home (6)
 - Home Designing (9)
 - Food** (61 items)
 - Food52 (38)
 - Lady and Pups (3)
 - Love and Lemons (10)
 - molly yeh (8)
 - smitten kitchen (2)
 - Competition** (2 items)
 - edwin@feedly.com via Google / Logout

On the right, the main content area shows several news items:

- Apple TV Gains Updated NFL Channel With Game Pass Integration**
The Apple TV's existing NFL Now channel was today revamped, changing the name to "NFL" and adding support for Game Pass subscriptions. Through the updated channel, NFL fans who have a Game Pass subscription can watch on 100+ MacRumors / by Juli Clover / 2h
- Tep Is An Adorable Fitness Tracking App That Works Like A Tamagotchi**
Remember the Tamagotchi? Those little monsters were great. A new iOS app called Tep created a Tamagotchi-like app for your phone to help you stay motivated when it comes to working out. Move around if you want to feed your 400+ TechCrunch / by Romain Dillet / 4h
- Apple Seeds Eighth Beta of OS X El Capitan to Developers, Sixth Beta to Public Testers**
Apple today released the eighth beta of OS X El Capitan to developers for testing purposes, nearly two weeks after releasing the seventh El Capitan beta and more than two months after unveiling the operating system at its 2015 300+ MacRumors / by Juli Clover / 4h
- Are you still using Apple Music?**
Apple Music has officially been available for two months now, and in that time it's had a few ups and downs. Despite some pesky, persisting bugs, Apple Music has quickly gained 11 million subscribers and counting during the trial 1K The Verge / by Micah Singleton / 5h
- Eatsa, A Futuristic Restaurant Where Robot Cubbies Serve Quinoa**
500+ TechCrunch / by Josh Constine / 5h

Captura de pantalla de [Feedly](https://feedly.com/i/welcome). <https://feedly.com/i/welcome>

Feedly

Lector inteligente que conecta con la información desde múltiples plataformas.

<https://feedly.com/i/welcome>

NewsBlur

Software gratis en la web, iPad, iPhone y Android. Al suscribirse a una cuenta *premium* desbloquea algunas restricciones.



Real-time RSS

Stories are pushed directly to you, so you can read news as it comes in.



Original site

Read the content in context, the way it was meant to be seen.



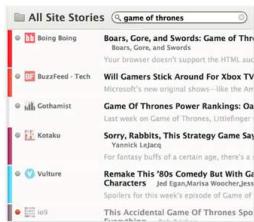
Shared stories

Reading news is better with friends. Share stories on your public blurblog.



Training

Hide the stories you don't like and highlight the stories you do.



Full Text Search

Quickly find stories across all of your subscriptions.



Story Tagging

Save stories with custom tags for fast references.



Blurblog Privacy

Share stories with the world or only with your friends.



Third-party Apps

Supports **Reeder**, **ReadKit**, **Unread**, and loads more.

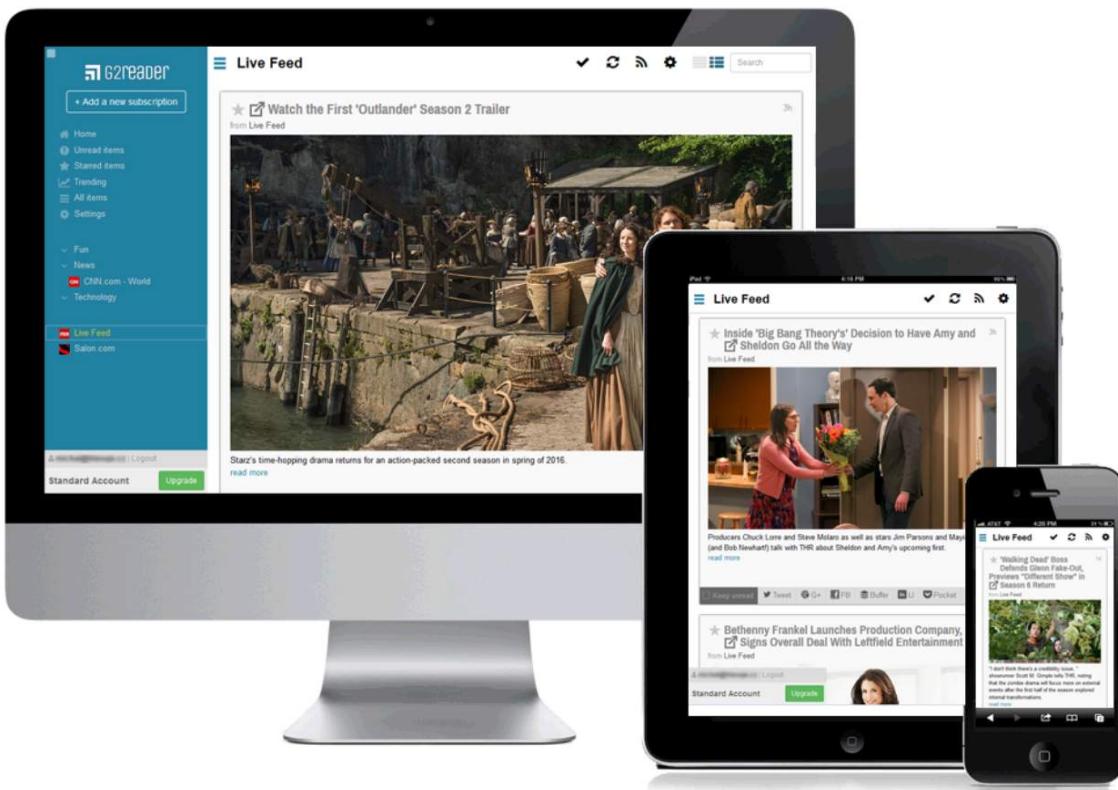
Captura de newsblur.com.

NewsBlur

Unir a las personas para hablar sobre el mundo. <https://newsblur.com/>

G2Reader

Parecido a Feedly, con dos columnas para organizar el contenido a la izquierda y mostrarlo a la derecha. Potente herramienta con opción de pago para obtener más funcionalidades.



Captura de [g2reader.com](https://g2reader.com/en/). <https://g2reader.com/en/>

G2Reader

Manténgase en contacto con los sitios que visita, léalos en un solo lugar, es tan fácil como consultar su correo electrónico. <https://newsblur.com/>

CommaFeed

Dispone solo de la versión web, aunque nos permite importar archivos OPML (formato XML para esquemas), guardar noticias para leerlas después y compartir las que más nos interesen en nuestras redes sociales.



CommaFeed

Buena alternativa si buscamos que no falte lo básico.

<https://www.commafeed.com/#/welcome>

Emisores directorios de RSS

Como los directorios permiten que el fichero RSS esté clasificado, el emisor define la temática del canal.

Para ello es necesario registrar el fichero RSS en un directorio RSS. El proceso es similar a registrar un sitio en un motor de búsqueda.

Es posible que a la hora de sindicar un canal puedan surgir problemas debido al uso de tildes o la letra ñ en los contenidos del canal. Es recomendable usar la codificación UTF-8 y las entidades XML que les sustituyen, es decir, á en lugar de á, é en lugar de é, < en lugar de <, > en lugar de >, etc.

Además de sindicar un canal, también podemos promocionarlo en los buscadores y directorios más populares.

Algunos ejemplos de estos directorios son:

- **Buscazoom RSS.** Reúne una importante colección de feeds RSS clasificados por categorías. <http://www.buscazoom.com/directorio-rss/>
- **Euroresidentes.** Canales de titulares, noticias, contenidos, servicios, newsfeed... https://www.euroresidentes.com/Diversion/Internet/rss_canales.htm
- **Feedage.** Directorio de canales RSS en inglés. Clasifica la gran cantidad de fuentes RSS disponibles. <http://www.feedage.com/>
- **Boosterblog.** Directorio español con más de 11 600 blogs activos. <http://www.boosterblog.es/>

Validar un documento RSS/Atom

Una vez generado el fichero con el canal de contenidos, hay que verificar que su codificación es correcta. Para ello es necesario utilizar un validador de fuentes de contenidos que podemos encontrar en Internet.

En general basta con introducir la URL del canal en el validador para comprobar si todo es correcto.

Muchos de los validadores, a parte de comprobar si el "feed" es correcto, ofrecen un conjunto de recomendaciones para mejorar la interoperabilidad con la mayoría de "feed readers" o lectores de RSS/Atom.

Una vez que se ha validado el documento, la mayoría de validadores ofrecen una imagen de color naranja que se puede incluir en la página principal y sirve para enlazar a la dirección del fichero alojado en el dominio. De esta forma, cuando un visitante del sitio web pulse sobre este icono naranja, se accederá directamente al contenido actual de la fuente y podrá navegar a través de él.

Existen múltiples validadores en la web que permiten realizar el trabajo mencionado anteriormente. Entre los más utilizados se encuentran:

RSS Advisory Board

Para validar RSS. <http://www.rssboard.org/rss-validator/>

FeedValidator

Para RSS, Atom y KML. <http://www.feedvalidator.org/>

W3C Feed Validation Service

Validador de la W3C que permite validar tanto a través de la URI como de un código fuente. https://validator.w3.org/feed/#validate_by_input

Agregadores

Una vez tenemos un canal de contenidos validado para ofrecer, o somos el consumidor de este, el siguiente paso sería utilizar un agregador o lector de fuentes.

Este agregador es un software que permite suscribirse a una fuente en formato RSS/Atom.

Esta aplicación avisa al usuario de los sitios web que han incorporado contenido nuevo, de tal forma que podrán ver las actualizaciones que surgieron desde su última lectura.

Podríamos decir que los agregadores se dividen en dos tipos:

Agregadores web:

Los agregadores web se ejecutan a través de la propia web. La ventaja de utilizar agregadores web es que puedes acceder a ellos desde diferentes ordenadores.

- Feedly. <https://feedly.com/i/welcome>
- Netvibes. <https://www.netvibes.com/en>

Agregadores de escritorio

Los agregadores de escritorio se deben instalar en la máquina local y son aconsejables para los usuarios que siempre acceden a Internet desde un mismo ordenador.

Su interfaz suele asemejarse a la de los gestores de correo electrónico.

- FeedDemon. <http://www.feeddemon.com/>
- RSSOwl. <http://www.rssowl.org/>

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.

- Qué son los directorios de sindicación de contenidos.
- Cómo trabajan y cuáles son los directorios de canales de contenidos.
- Validar un "feed".
- Qué servicios web ofrecen validadores "feed".
- Los diferentes tipos de agregadores que existen y cuáles son los más utilizados.