DOCUMENTO RECOPILATORIO EXAMEN FINAL M4

Las preguntas recogidas en este documento son susceptibles (o no) de aparecer en el examen final. En este documento no están recogidas el 100% de las preguntas que pueden aparecer en el examen, las preguntas pueden provenir de dudas en foros, muros, video-tutorías, material didáctico e incluso cualquier recurso que os haya aportado durante el semestre.

No obstante seguir las indicaciones de este documento os prepara para realizar una buen Prueba de Evaluación Final.

Cada UF del Examen constará de 12 preguntas de Tipo test que serán el 60% de la nota del Examen para cada UF.

Cada UF del Examen constará también de 2 preguntas (cortas a resolver en 4-5 líneas o ejercicios prácticos) que serán el 40% de la nota del Examen por cada UF.

Examen/Nota	60%	40%
UF1	12 preguntas tipo test	2 preguntas cortas o ejercicios
UF2	12 preguntas tipo test	2 preguntas cortas o ejercicios
UF3	12 preguntas tipo test	2 preguntas cortas o ejercicios

PREGUNTAS DEL MATERIAL DIDÁCTICO

Estas preguntas pueden aparecer en formato tipo test o como pregunta corta a resolver en 4 líneas. Las preguntas marcadas en AMARILLO son más susceptibles de aparecer como preguntas cortas. Esto no quiere decir que las demás no puedan hacerlo, pero sí que estas debéis repasarlas bien. Pensad que estas preguntas pueden aparecer como tipo test también.

- **TEMA 1:**
  - **Punto 1.1: Concepto y ventajas. (solo página 2)**

Cualquier tipo de lenguaje utilizado por las personas se le está aplicando un determinado formato a la información que transmitimos.

En el caso de los documentos que intercambiamos a través de internet, como las páginas web, son los lenguajes de marcas quienes nos permiten aplicar dicho formato.

El navegador o mejor dicho el agente de usuario, es el encargado de interpretar las marcas de formato y las aplica convenientemente al texto para dar lugar a una página web, que será mucho más agradable de leer que el texto original.

Una marca es una señal colocada dentro de un texto, con el fin de delimitar una parte del mismo y en muchos casos, aplicarle un determinado formato (aunque existen marcas con otros propósitos).

Las marcas más comunes están formadas por una palabra que describe su función encerrada entre los símbolos menor que (<) y mayor que (>) como <html>.

Es muy habitual que aparezcan por parejas, unas de comienzo y otra de fin.

Los lenguajes de marcas no son equivalentes a los lenguajes de programación aunque se llamen también lenguajes. De hecho, no debemos utilizar la palabra “programa” cuando nos referimos a lenguajes de marcas, puesto que no disponen de los elementos típicos como variables, arrays, sentencias de control, funciones, etc.

Sin embargo, los lenguajes de marcas se pueden combinar dentro del mismo documento, con otros lenguajes como JavaScript o PHP, que sí son lenguajes de programación, con el objetivo de aportar funcionalidad y dinamismo a la página web.

Otro aspecto importante a tener en cuenta cuando hablamos de lenguajes de marcas es el destinatario de la información. Quizás lo más habitual, es un usuario final utilizando un navegador web en el PC de su casa, pero tenemos que considerar el resto de opciones que van en aumento. La presentación de la misma página web para cada usuario deber ser lógicamente distinta.

La cuestión es que el lenguaje de marcas debe ser independiente del destinatario final, es el intérprete del lenguaje quien se encarga de representar las marcas de la forma adecuada.

Para independizar aún más la representación de la página web de su contenido, se creó CSS, que no es un lenguaje de marcas sino de estilos. Mediante CSS podemos especificar con mayor precisión y eficacia la representación de la información, para cada intérprete y para diferentes soportes.

Muchas páginas presentan diferentes versiones adaptadas al dispositivo que utilice el usuario, en este caso, se trata de documentos html diferentes o bien del mismo documento html, pero aplicándole una hoja de estilos distinta.

- **Punto 1.2: SGML. El origen. (solo página 5)**

En los años 60 las empresas de publicación y manejo de documentos electrónicos tenían el problema de falta de compatibilidad entre aplicaciones. El problema existente era que cada aplicación utilizaba sus propias marcas para describir los diferentes elementos, esto impedía el intercambio de documentos entre plataformas. Otra carencia importante era la separación entre estructura y aspecto del documento.

IBM, empresa pionera en investigación en informática y electrónica (más de 5.000 patentes en 100 años de historia) intentó resolver estos problemas a través de un lenguaje de marcas denominado GML (Generalized Markup Language).

GML independiza el documento del dispositivo que lo va a utilizar, usando marcas genéricas. Por otro lado GML incorpora marcas descriptivas para la estructura del documento que permiten distinguir el texto, de las listas, las tablas, etc. El mismo documento puede, entonces, ser utilizado por varios dispositivos, simplemente especificando un perfil para cada uno.

En 1986 GML pasó a manos de ISO y se convirtió en SGML(ISO 8879), Standard Generalized Markup Language, software libre y de código abierto.

Es importante tener en cuenta que SGML no es estrictamente un lenguaje sino un metalenguaje, es decir, un conjunto de normas que permiten crear otros lenguajes de marcas.

Esto se hace definiendo un vocabulario o conjunto de elementos a utilizar, y una gramática o conjunto de reglas que rigen el uso de los elementos y sus atributos.
SGML, por tanto, es un metalenguaje que permite definir lenguajes de marcado. HTML, por ejemplo, es uno de los lenguajes creados a partir de SGML.

Ventajas de SGML:
- Reutilización de los datos.
- Integridad.
- Control sobre los datos.
- Portabilidad.
- Adaptabilidad.

Inconvenientes de SGML: Alta complejidad.

Un documento SGML consta de 2 partes:

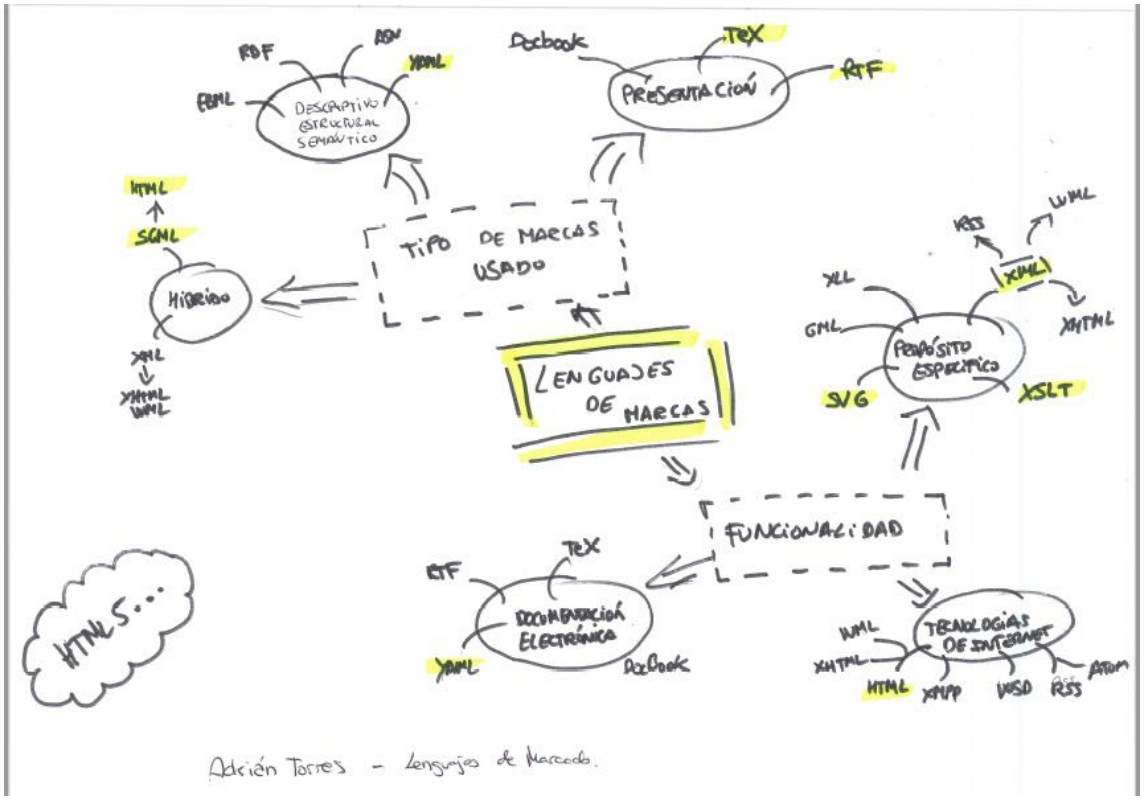
- El prólogo: contiene la estructura.
    - La declaración: indica que el documento es SGML y algunos parámetros.
    - La definición de tipo de documento (DTD): indica la sintaxis particular del lenguaje creado.
- La instancia del documento: contiene los datos.

    - **Punto 1.4: Clasificación de los lenguajes de Marcas. (solo negritas). Esquema en PAC.**

Clasificación según el tipo de marcas que utilizan:
- De presentación.
- Descriptivo, estructural o semántico.
- Híbrido.

Clasificación según su funcionalidad:
- Para crear documentación electrónica.
- Tecnologías de Internet.
- De propósito específico.

El origen de los lenguajes de marcas como ya sabemos es SGML, del cual derivan directamente algunos lenguajes como HTML. También se creó a partir de él por simplificación XML, otro metalenguaje más fácil de usar y entender. A partir de XML se han creado XHTML, RSS y un largo etc.



    - **Punto 1.6.2: Solo características XML.**

XML es una simplificación y adaptación de SGML, que permite definir lenguajes específicos. XML es un metalenguaje. XHTML, SVG y RSS son lenguajes basados en XML.

Las característica són:
- Extensible: se pueden definir nuevas etiquetas.
- Versátil: separa contenido, estructura y presentación.
- Estructurado: se pueden modelar datos a cualquier nivel de complejidad.
- Validable: cada documento se puede validar frente a un DTD / Schema.
- Abierto: independiente de empresas, sistemas operativos, lenguajes de programación o entornos de desarrollo.

- Sencillo: fácil de aprender y de usar.

- **TEMA 2: HTML y XHTML**

    - **Punto 2.1: Solo leer primeros 2 párrafos.**

HTML (Hypertext Markup Language) aparece a principios de los años noventa cuando el investigador Tim Berners Lee, lo desarrolla para potenciar la colaboración entre físicos e investigadores de la energía nuclear.

El **objetivo principal** era enlazar los documentos mediante links (hipervínculo), de forma que al hacer clic sobre algún texto, se abrirá otro documento relacionado con la información seleccionada.

    - **Punto 2.2: Saber estructura de un documento.**

Los documentos HTML son archivos de texto plano formados por etiquetas de apertura y cierre (por ejemplo:<html>...</html>) que permiten dar formato al contenido del documento.

Una página web es el resultado que ofrece un navegador al intepretar dicho documento.

La **estructura básica** de un documento HTML es:

```
<!Declaracion de Tipo de Documento>
<html>
	<head>
		<title>Titulo de la pagina</title>
		Elementos opcionales de cabecera
	</head>
	<body>
		Contenido del cuerpo
	</body>
</html>
```

Tanto la declaración del tipo de documento, como las etiquetas html, head y body son opcionales, pero deben incluirse para que el documento sea correcto y legible.

La etiqueta <html> es la raíz del documento por lo que head (cabecera) y body (cuerpo) deben estar totalmente contenidas en ella.

Ademas head y body no deben solaparse entre sí, por lo tanto, el documento deben tener un gran bloque principal (html) y dos subbloques al mismo nivel.

Estos dos bloques a su vez, pueden tener otros subbloques que tampoco deben solaparse.

Es buena costumbre separar los bloques importantes por líneas de comentarios.

La declaración de tipo de documento se coloca al principio, fuera de todo bloque.

DTD (Document Type Definition – Definición de Tipo de Documento), se trata de un documento externo con **extensión .dtd** y que existen 3 posibilidades: **estricta, de transición y con marcos**.

Al indicar la DTD al principio del documento le decimos al navegador como interpretar adecuadamente el código.

HTML 4.01 recomienda separar por completo el contenido del formato.

DTD estricta:
Si cumplimos escrupulosamente con esta recomendación, almacenando toda la información sobre el formato en una hoja de estilos CSS, entonces podemos utilizar la declaración estricta: **<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"**

"http://www.w3.org/TR/html4/strict.dtd">
DTD de transición:
Si queremos combinar contenido e información de formato en nuestro documento HTML, entonces debemos utilizar la declaración de transición: **<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"** **"http://www.w3.org/TR/html4/loose.dtd">**
DTD con marcos:
Si queremos incluir marcos en nuestro documento, entonces tenemos que usar una DTD que es muy similar a la anterior, pero que sustituye el elemento body por frameset, en este caso la declaración seria: **<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"** **"http://www.w3.org/TR/html4/frameset.dtd">**

En HTML 5 la declaración es más simple: **<!DOCTYPE html>**

Ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
	<head>
		<title>Mi primera pagina</title>
	</head>
	<body>
		<h1>Hola Mundo</h1>
	</body>
</html>
```

- **Punto 2.4: Conocer elementos de la cabecera.**

<head>

Definición: delimita la cabecera de un documento.

Aparición: apertura y cierre opcionales.

Atributos: **lang, dir, profile.**

- **Profile:** indica donde estan los perfiles para interpretar los elementos meta.

  Valores: **URI (Uniform Resource Identifier – Identificador Uniforme de Recursos)**, dirección web completa del recurso.

Contenido de la cabecera: **<title>, <base>, <meta>, <link>, <object>, <script> y <style>.**

Todos los elementos son opcionales excepto <title> que es obligatorio.

El contenido de la cabecera es una información general de todo el documento, por eso se escribe al principio y es lo primero que analiza el navegador.

<title>

Definición: indica el nombre del sitio Web en la barra superior o pestaña del navegador.

Aparición: etiquetas de apertura y cierre obligatorias.

Atributos: **lang, dir.**

Ejemplo:

```
	<head>
		<title>Mi primera página</title>
	</head>
```

<base>

Definición: indica la dirección raíz del sitio Web, la cual permite resolver las direcciones relativas.

Aparición: sin etiqueta de cierre (V).

Atributos: **href**

- **href:** se usa siempre para indicar la dirección raíz del documento.

Ejemplo: **<base href="http://www.juanmacar.es/">**

<meta>

Definición: indica un conjunto de propiedades generales del documento como el autor, la descripción, palabras clave, herramienta utilizada, tipo de contenido, etc.

Estas propiedades evolucionan continuamente, es decir, se crean nuevas propiedades para ampliar la información asociada al documento, las redes sociales son un ejemplo.

Normalmente estas propiedades pasan desapercibidas para el usuario habitual.

Aparición: sin etiqueta de cierre (V)

Atributos: **lang, dir, name, content, http-equiv, scheme.**

- **name:** indica el nombre de una propiedad, no hay lista oficial de propiedades pero podemos citar los siguientes valores: *abstract, autor, copyright, date, description, distribution, expires, generator, google-site-verification, keywords, language, no-email-collection, organization, rating, reply-to, revisit-after, robots* y la familia de etiquetas *Dublin Core*.

- **content:** contiene el valor de la propiedad indicada antes por name, por tanto, estos dos atributos funcionan simultáneamente proporcionando parejas de valores.

  Ejemplo: **<meta name="description" content="Lenguaje de Marcas">**

- **http-equiv:** indica una propiedad al navegador en forma de cabecera http, como si el propio servidor http (Web) hubiera generado dicha cabecera, de ahí viene el nombre "http-equivalent".

  Valores: *cache-control, content-type, set-cookie, content-disposition, pics-label, pragma, refresh, resource-type, content-script-type, content-style-type, window-target*.

  Tiene muchas utilidades como por ejemplo, indicar el conjunto de caracteres a utilizar en nuestro sitio web, sin necesidad de modificar el servidor. También funciona en coordinación con el atributo content para proporcionar parejas de valores.

  Ejemplo:

  <meta http-equiv="content-type" content="text/html ; charset=iso-8895-15">

  Esto indica que el contenido de la página es texto plano HTML con el conjunto de caracteres iso-8895-15, que corresponde al alfabeto latino nº9.

- **scheme:** indica al navegador que tiene que interpretar los metadatos antendiendo al perfil especificado, para evitar ambigüedad.

  Ejemplo:

  <meta scheme="Europe" name="date" content="24-10-2012">

Esto indica que la fecha se lee segun el perfil “Europe”, es decir, “dd-mm-aaaa”.

Ahora vamos a estudiar las principales meta etiquetas en función de su utilidad.

Se clasifican en dos grupos (por sus dos atributos principales aunque hay mas).

1) **Meta name:** se utiliza principalmente para optimizar el resultado de los motores de búsqueda, es decir, para ayudar a los buscadores de paginas web.

   La información que manejan no se presenta en pantalla.

   Informacion general sobre la página: **abstract, autor, copyrigth, date, generator.**

   Información especifica para buscadores: **description, distribution, keywords, locality, rating, revisit-after, robots.**

   Para administración del sitio: **google-site-verification, reply-to.**

2) **Meta http-equiv:** se utilizan para dar instrucciones a los navegadores, es decir, ejercen cierto control sobre ellos.

   Tienen influencia sobre lo que aparece en pantalla.

   Tipo de contenido: **content-type, content-language, content-script-type.**

   Manejo de cookie: **set-cookie.**

   Actualización/Redireccion de la pagina: **refresh.**

   Transiciones de pagina: **page-enter, page-exit.**

   Control parental: **pics-label.**

   Manejo de la cache: **cache-control, pragma, last modified, expires.**

3) **Protocolo Open Graph** es un protocolo que permite identificar los elementos que contiene nuestra pagina web dentro de una red social.

   Las propiedades básicas son:

   og:title - El titulo tal y como aparecerá en la red.
   og:type - El tipo de objeto, por ejemplo “article”.
   og:image - La URL de una imagen representativa del objeto.
   og:url – La URL canónica del objeto que será su identificador en la red.

   Ejemplo:

```
<meta property=”og:title” content=”mipagina”/>
<meta property=”og:url” content=”http://mipagina.es”/>
```

<link>

Definicion: define un vincula a otro documento indicado por href.

Aparicion: sin etiqueta de cierre (V)

Atributos: **lang, dir, class, id, style, title, eventos intrinsecos, charset, href, hreflang, type, rel, rev y media.**

- **charset:** establece el conjunto de caracteres usado para el documento.
- **href:** indica la URI del documento vinculado.
- **hreflang:** indica el lenguaje del documento vinculado.

- **type:** indica el tipo de contenido del documento vinculado, para que el navegador no lo abra en caso de no tener soporte.

  Los tipos mas comunes son: **text/html, text/css, image/png, image/gif, image/x-icon, video/mpeg y audio/basic.**

  Tipos para documentos xml: **application/rss+xml, application/atom+xml**

  Otras: **application/x-shockwave-flash, application/opensearchdescription+xml**

- **rel:** establece la relación entre los documentos origen y destino.

- **rev:** indica un vincula inverso, es decir, desde el destino al origen.

- **media:** indica el medio al que se refiere los datos de estilo en el documento destino.

  Tipos de medios: **screen, tty, tv, projection, handheld, print, braille, aural, all.**

Ejemplo:

Para indicar que hay una versión de la pagina en otro idioma:
<link rel="alternate" hreflang="en" title="Mi pagina en Ingles" type="text/html" href="http://mipagina.es/ingles.html">

Para indicar que hay un archivo rss:
<link rel="alternate" type="application/rss+xml" title="Resumen de todas las secciones" href="/feed.xml">

Para insertar un icono en la barra de titulo o pestaña del navegador:
<link rel= "shorcut icon" type= "image/x-icon" href= "http://mipagina.es/imagenes/logo.ico">

Para vincular una hoja de estilos con el documento actual:
<link rel="stylesheet" type="text/css" href="http://mipagina.es/estilos.css">

Para especificar la pagina principal cuando hay duplicación en diferentes idiomas:
<link rel="canonical" href="http://mipagina.es/index.html">

En todos los casos se puede especificar una ruta relative en href dependiendo de la estructura de directories.

<script>

Definicion: inserta un script dentro del documento.

Puede aparecer varias veces, tanto en la cabecera como en el cuerpo y puede ser interno al documento o bien un fichero externo.

Aparición: las etiquetas de inicio y cierre son obligatorias.

Atributos: **charset, type, src, defer.**

- **charset:** codificación de caracteres del script indicado por **src.**

- **type:** indica el lenguaje de programación en el cual esta escrito el script.

  Como ya sabemos, con la etiqueta **<meta>** se puede determinar el lenguaje script por defecto para el documento. Si no se hace asi, es conveniente indicar el lenguaje cada vez mediante el atributo **type**.

- **src:** indica la URI donde esta alojado el script externo (del ingles source).

- **defer:** es un booleano, su presencia indica al navegador que el script no va a generar ningún contenido en el documento.

Ejemplos:

- Interno:

  <script type="text/javascript"> ... </script>

- Externo:

  <script type="text/javascript" src="/js/banner.js"> ... </script>

  <script type="text/javascript" src="https://www.google.com/jaspi">
  ...
  </script>

<style>

Definición: permite inserter una hoja de estilo interna en la cabecera del document.

Aparición: etiquetas de inicio y fin obligatorias.

Atributos: **lang, dir, type, media, title.**

- **type:** indica el tipo del lenguaje de estilos. Normalmente será **text/css.**

- **media:** indica el tipo de medio al que se dirige la información de estilo.

  Por defecto será **"screen".** Otros valores: **tty, tv, handheld** (para móviles), **print** (para impresoras), **braille, aural, all.**

- **title:** es un texto descriptivo de la información de estilo.

Ejemplo:

```
<style type="text/css">
        body {
                margin-left:40px;
                margin-top:40px;
                margin-right:40px;
        }
</style>
```

- **Punto 2.4.3.1: Proviene de PAC, saber que significan los símbolos de Copyright (figura 2.1 o similar).**

creative commons españa | CC ES | PREGUNTAS FRECUENTES | LICENCIAS

Licencias

Usar una licencia Creative Commons es muy fácil

Poner vuestras obras bajo una licencia Creative Commons no significa que no tengan copyright. Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones. ¿Qué condiciones? Esta web os permite escoger o unir las condiciones de la siguiente lista »

Reconocimiento (Attribution): En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.

No Comercial (Non commercial): La explotación de la obra queda limitada a usos no comerciales.

Sin obras derivadas (No Derivate Works): La autorización para explotar la obra no incluye la transformación para crear una obra derivada.

Compartir Igual (Share alike): La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Con estas condiciones se pueden generar las seis combinaciones que producen las licencias Creative Commons:

Reconocimiento (by): Se permite cualquier explotación de la obra, incluyendo una finalidad comercial, así como la creación de obras derivadas, la distribución de las cuales también está permitida sin ninguna restricción.

Reconocimiento – NoComercial (by-nc): Se permite la generación de obras derivadas siempre que no se haga un uso comercial. Tampoco se puede utilizar la obra original con finalidades comerciales.

Reconocimiento – NoComercial – CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Reconocimiento – NoComercial – SinObraDerivada (by-nc-nd): No se permite un uso comercial de la obra original ni la generación de obras derivadas.

Reconocimiento – CompartirIgual (by-sa): Se permite el uso comercial de la obra y de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Reconocimiento – SinObraDerivada (by-nd): Se permite el uso comercial de la obra pero no la generación de obras derivadas.

- **Punto 2.5.1: Manejo del texto, conocer H1, H2,...**

<h1 | h2 | h3 | h4 | h5 | h6>

Sirven para establecer encabezados, es decir, el tamaño de los títulos.

Hay seis niveles de encabezados: **h1,h2,h3,h4,h5 y h6** ordenados estrictamente por su tamaño. Los encabezados son **elementos de bloque.**

Aparicion: las etiquetas de apertura y cierre son obligatorias.

Atributos: **%attrs, align.**

Ejemplo:

```
<body>
       <h1>Encabezado H1</h1> <h2>Encabezado H2</h2>
       <h3>Encabezado h3</h3> <h4>Encabezado h4</h4>
       <h5>Encabezado h5</h5> <h6>Encabezado h6</h6>
</body>
```

<p>

Definicion: Otro de los elementos de bloque mas típicos es el párrafo.

De la misma forma que distribuimos el texto mediante párrafos en un documento Word, la líneas de texto de un documento HTML deben agruparse dentro de <p>. Cada párrafo en HTML se compone de una única línea de texto, salvo que se introduzca un retorno de carro explicito usando el **elemento
.** El navegador se encarga de que la línea de texto ocupe todo el ancho disponible, y si el ancho cambia, el texto se ajusta al nuevo ancho ocupando mas o menos líneas.

Todo texto de un documento HTML, que no sea encabezado, lista o celda de tabla debe estar dentro del elemento <p>.

Aparicion: la etiqueta de cierre es opcional. No puede contener elementos de bloque.

Atributos: **%attrs, align.**

Un asunto a tener en cuenta es el espacio en blanco, si queremos insertar un espacio mas del normal entre dos palabras, no vale utilizar la barra espaciadora porque es ignorada por el navegador. Es necesario insertar el **carácter   (no-break space)** por cada espacio que queramos añadir.

**
**

A veces es preciso introducir un salto de línea dentro del mismo párrafo, en ese caso debemos insertar la etiqueta
 de “break” que sirve para “romper” la línea de texto. No sirve insertar un salto de línea usando enter porque también es ignorado por el navegador.

Usando el ejemplo anterior, para evitar que una frase quede partida y mejorar la legibilidad del texto, podríamos introducir saltos de línea detrás de punto y seguido.

Aparicion: sin etiqueta de cierre (V).

<hr>

Otra forma posible de separar el texto es utilizar una línea visible horizontal.

Aparicion: sin etiqueta de cierre (V).

Atributos: **%attrs, align, noshade, size y width.**

<b> <i> <big> <small> <s> <strike> <tt> <u>

Definicion: son los llamados estilos de Fuente.

b: negrita, **i:** itálica o cursiva; **big:** tamaño grande; **small:** tamaño pequeño; **s y strike:** texto tachado mediante una línea a media altura; **tt:** texto con apariencia de teletipo; **u:** subrayado con una línea por debajo.

Aparicion: las etiquetas de apertura y cierre son obligatorias.

Atributos: **&attrs.**

Se trata de elementos de línea, por tanto, no hacen salto de línea y se pueden introducir dentro de un párrafo <p>. Tambien pueden contener a otros combinándose entre si.

<font> Esta desaprobado, no usar.

Definicion: sirve para establecer el tamaño, el color y la fuente para el texto, esta desaprobado.

Aparicion: apertura y cierre obligatorios.

Ejemplo:

```
<body>
       <h1>Estilos de Fuente</h1> <hr>
       <ul>
          <li>
              <h2>Estilos simples</h2>
              <b>Negrita</b><i>Cursiva</i><big>Tamaño grande</big>
              <small>Tamaño pequeño</small><tt>Teletipo</tt>
              <strik>Tachada</strike><u>Subrayado</u>
          </li>
          <li>
              <font face=”Verdana” size=”4”>Font face Verdana</font>
              <font face=”Courier New”>Font face Courier</font>
          </li>
       </ul>
</body>
```

Las posibilidades usando hojas de estilos son mucho mayores, por ello, todos los estilos de fuente estan en desuso.

<pre> Texto con formato previo

Definición: sirve para que el navegador visualice el texto tal y como aparece en el contenido de esta etiqueta, respetando tabuladores, espacios y saltos de línea.

La única excepcion es que introduzcamos en el contenido, otros elementos HTML, los cuales se ejecutaran también.

Aparicion: etiqueta incial y final obligatorias.

Atributos: **%attrs y width.**

- **Width:** indica el ancho del bloque con formato previo.

Ejemplo:

```
<body>
       <h1>Programacion en Java</h1>
       <h2>
         <pre>
              class HolaMundo {
                     public static void main (String[] args) {
                       System.out.println(<&lt;Hola Mundo&gt;>)
                     }
              }
         </pre>
       </h2>
</body>
```

Para evitar que los caracteres “<” y “>” se interpreten como etiqueta debemos utilizar:

< = “<” (less than)
> = “>” (greater than)

Si no utilizamos <pre> se visualizaría todo el texto en una sola línea.

- **Punto 2.5.2: Conocer tipos de listas.**

Tenemos 2 tipos fundamentales de listas: **listas de elementos y listas de definiciones.**

A su vez las **listas de elementos pueden estar ordenadas o no.**

Las listas son un recurso muy común en los textos, como se puede apreciar en este esquema, en el que se han utilizado listas no ordenadas.

Listas
- **Listas de elementos**
  - **Listas no ordenadas (<ul>,<li>)**
  - **Listas ordenadas (<ol>,<li>)**
- **Listas de definiciones (<dl>,<dt>,<dd>)**

Listas de elementos.

** Listas de elementos no ordenados.**

Definición: se construyen con el elemento <ul> y su contenido está **formado exclusivamente por elementos <li>.**

Los **elementos ** son los que contienen a su vez el texto.

Aparicion: las etiquetas de apertura y cierre son obligatorias.

Atributos: **%attrs, type y compact.**
- **type:** indica el tipo de símbolo utilizado y puede tomar los valores “disc” (disco), “square” (cuadrado) y “circle” (circulo). Desaprobado.
- **compact:** si aparece indica al navegador que la lista debe compactarse.

** Listas de elementos ordenados**

Definición: se construyen con el elemento <ol> y su contenido esta formado exclusivamente por elementos <li>.

Aparición: las etiquetas de apertura y cierre son obligatorias.

Atributos: **%attrs, type y start.**
- **type:** indica el tipo de secuencia de lista y puede tomar los valores 1 (numeros decimales), a (letras minúsculas), A (letras mayúsculas), i (números romanos minúsculas) y I (números romanos mayúsculas). Desaprobado.
- **start:** indica el numero del primer elemento de la lista. Desaprobado.

** Elementos de lista.**

Definición: objeto de lista, que puede contener otros elementos de bloque como párrafos o directamente el texto.

Aparición: El cierre es opcional.

Atributos: **%attrs y value.**
- **value:** establece el numero del objeto de lista actual. Desaprobado.

Ejemplo:

```
<body>
   <h1>Lenguajes de Marcas</h1>
   <h2>Lista ordenada</h2>
   <ol>
       <li>HTML</li>
       <li>XHTML</li>
       <li>XML</li>
   </ol>
   <h2>Lista no ordenada</h2>
   <ul>
       <li>HTML</li>
       <li>XHTML</li>
       <li>XML</li>
   </ul>
</body>
```

Listas anidadas: La mayor utilidad de las listas resulta cuando se combinan, es decir, cuando cada elemento de una lista es a su vez otra lista. En estos casos los niveles de anidación deber ser de tipos distintos, para evitar la confusión del lector.

Ejemplo:

```
<body>
   <ol>
       <li>Paginas Web
              <ul>
                     <li>HTML</li>
                     <li>XHTML</li>
                     <li>XML</li>
              </ul>
       </li>
   </ol>
</body>
```

Listas de Definiciones:

<dl>

Se trata de listas que no tienen índices ni símbolos de ninguna clase como las anteriores, sino que se componen de dos elementos: **el término y la definición,** a modo de diccionario. Todo el texto aparece con sangría.

Definición: se construyen con el elemento <dl> y **contienen solamente elementos de tipo <dt> y <dd> en numero variable.**

Aparición: etiquetas de apertura y cierre obligatorio.

Atributos: **%attrs.**

<dt>

Definición: es el término que se quiere definir, normalmente una palabra o un texto breve y se trata de un elemento de línea.

Aparición: etiqueta de cierre opcional.

Atributos: **%attrs.**

<dd>

Definición: es la definición del término anterior, normalmente un texto largo y se trata de un elemento de bloque.

Aparición: etiqueta de cierre opcional.

Atributos: **%attrs.**

Ejemplo:

```
<body>
  <dl>
       <dt>HTML</dt>
       <dd>HyperText Markup Language</dd>
       <dd>Lenguaje de Marcas de Hipertexto</dd>
       <dt>XHTML</dt>
       <dd>Extensible HyperText Markup Language</dd>
  </dl>
</body>
```

- **Punto 2.5.4 y 2.5.6: Proviene de PAC. Diferenciar entre DIV y TABLE.**

El elemento HTML div es un contenedor a nivel de bloque para otros elementos de línea y de bloque. Por sí mismo, no tiene significado alguno a nivel presentacional o semántico, exceptuando que, al ser un elemento a nivel de bloque, los navegadores mostrarán un quiebre de línea antes y después de su contenido.

Los elementos HTML div adquieren su potencial al ser usados conjuntamente con hojas de estilo, ya que resultan muy útiles para asignar atributos presentacionales a bloques enteros de contenido.

Otro uso útil para este elemento, y tal vez el más importante, es el de establecer la distribución o el diseño (en inglés "layout") del documento. Los elementos DIV han venido a reemplazar la antigua forma de establecer el diseño del documento, que usaba tablas para organizar la distribución del contenido. Estos diseños con tablas hacían uso erróneo del elemento HTML table, cuyo propósito no es otro que representar información tabulada.

La costumbre de usar las tablas en lugar de divs para el lagyout, cada día más se va perdiendo, y es así como debe de ser, ya que ofrece muchas limitaciones en cuanto a darle un formato que uno desee, ya sea en colores, fuentes, diseños, etc.

Por otra parte, las div cada vez se hacen más atractivas y versátiles, debido a que permite un diseño web mucho más vistoso y con una gran facilidad de interpretación y escritura en código, ya que se le da formato mediante el uso del lenguaje CSS.

Diferencias:

- Las tablas son ideales para presentar datos, pero en cuestión de diseño y maquetación los divs son más recomendados ya que puede ahorrar bastante código.
- Los divs pueden separar el contenido en “bloques” diferenciados e independientes entre si, sin obviar la posibilidad de depender o interactuar entre ellos.
- El código queda mucho mas limpio y manejable con divs que con tablas, por lo que es mas sencillo localizar errores y cambiar el diseño.
- Los divs son mas flexibles y manejables que las tablas, tienen mas usos y mas propiedades.

- **Pagina 65: Figura 2.17 (Contenido de la tabla, Definición de siglas).**

Cod	Nombre	Significado
1	SGML	Standard Generalized Markup Language
2	HTML	HyperText Markup Language
3	XML	eXtended Markup Language

- **Punto 2.7: HTML5 (solo página 86).**

Esta versión aporta nuevos elementos enfocados a distintas tareas como la presentación, el diseño de la pagina (layout), facilitar la inclusión de audio y video, mejorar los formularios, almacenar los datos de sesión (evitando usar

cookies), generar eventos "server-sent" desde el servidor, permitir la geolocalización del sitio web, construir una superficie de dibujo llamada "canvas", arrastrar objetos de un lugar a otro de la pagina, entre otras cosas.

Pero no se trata solamente de incluir nuevos elementos, esta versión tiene el objetivo mas ambicioso de convertirse en una plataforma de desarrollo, que integre todas las tecnologías web. Hoy en dia para desarrollar una web deben combinarse diferentes lenguajes (como HTML, CSS, Javascript...), diferentes formatos de audio y video, animaciones, etc. A veces esto crea problemas de compatibilidad con algunos navegadores y diferentes resultados al visualizar una pagina según el navegador o versión utilizada. Ademas, exige al usuario la descarga y actualización de los plug-in para reproducción de animaciones y videos.
HTML 5, pretende ser una solución a esto incorporando una API (Application Program Interface), es decir, una biblioteca de funciones lo bastante amplia para que todos los navegadores se comporten de igual forma.

Tambien se produce una simplificación importante a nivel de atributos, todos los elementos de HTML5 tiene un conjunto común llamado **global attributes.**

En cuanto a la sintaxis, es muy flexible y pone fin a la bifurcación entre HTML y XHTML convirtiéndose en un estándar para los dos lenguajes.

Caracteristicas:

- Se permiten las mayúsculas en las etiquetas.
- La etiqueta de cierre es opcional en los elementos vacios.
- Es opcional establecer un valor a los atributos.
- Las comillas son opcionales en los atributos.

Dado que HTML5 todavia es un lenguaje en construcción, algunos de sus elementos o atributos no son soportados por todos los navegadores, lo cual significa que algunos objetos no se viusalicen de igual forma.

- **TEMA 3: CSS. Hojas de estilo**

    - **Punto 3.1: CSS (solo página 112 y primeros 4 párrafos)**

CSS (Cascading Style Sheets – Hojas de Estilo en Cascada) no es un lenguaje de marcas sino un **lenguaje de presentación**, por tanto, su estructura y sintaxis son diferentes.

CSS está íntimamente ligado a los lenguajes de marcas, hasta el punto que resulta imprescindible para crear sitios web con calidad.

El objetivo principal de este lenguaje es manejar el aspecto y formato de los documentos, liberando de esta forma el HTML de las tareas de presentación.

Esta separación entre contenidos y presentación supone muchas **ventajas:**
- Evita la repetición innecesaria de código.
- Disminuye el código a escribir.
- Facilita la generación de código y su mantenimiento.
- Mejora la legibilidad de los documentos.

    - **Punto 3.5: Leer/Repasar.**

Las **principales propiedades CSS relacionadas con el texto** se pueden ver en la siguiente tabla:

Propiedad:valor	**Significado**
color: <color> | inherit	Color del texto
font-family: <fuente> | inherit	Tipo de fuente de letra
font-size: <absoluto> | <relativo> | | <%> | inherit	Tamaño de letra
font-weight: normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | Inherit	Grosor de letra

font-style: normal | italic | oblique | inherit	Estilo de letra
font-variant: normal | small-caps | inherit	Mayúsculas en pequeño
text-align: left | right | center | justify | inherit	Alineación horizontal
line-height: normal | <numero> | <medida> | <porcentaje> | inherit	Interlineado
text-decoration: none | (underline || overline || line-through || blink) | inherit	Decoración
text-transform: capitalize | uppercase | lowercase | none | inherit	Transformación
text-shadow: none | h-shadow | v-shadow | blur | color	Sombreado del texto
vertical-align: baseline | sub | super | top | text-top | middle | bottom | text-bottom | <porcentaje> | <medida> | inherit	Alineación vertical
text-indent: <medida> | porcentaje | inherit	Tabula las primeras lineas
letter-spacing: <medida> | porcentaje | inherit	Espaciado entre las lineas
white-space: normal | nowrap | pre | pre-line | pre-wrap | inherit	Tratamiento de los espacios en blanco
word-spacing: normal | <medida> | inherit	Espaciado entre palabras

Para incluir **tipos de fuentes especiales,** se puede agregar el archivo de fuentes en la carpeta apropiada del sistema (C:\Windows\fonts normalmente). Pero en el caso de contratar un hosting externo no tendremos acceso a dicha carpeta, ni falta que hace, puesto que con CSS podemos importar los archivos de fuentes deseados mediante la **regla @font-face.**

Formato de uso:
@font-face {font-family: *mifuente*; src: url(*'archivo de fuentes'*); }

Ejemplo de uso:

```
<html>
   <head>
       <style type="text/css">
          @font-face {font-family: chopin; src: url(ChopinScript.otf);}
          @font-face {font-family: allstar; src: url(Allstar.ttf);}
          .chopin {font-family:chopin;}
          .allstar {font-family:allstar;}
       </style>
   </head>
   <body>
       <h1>Cambiar la tipografia con CSS3:</h1>
       <p class="chopin">Este texto utiliza la fuente Chopin</p>
       <p class="allstar">Este texto utiliza la fuente Allstar</p>
   </body>
</html>
```

En http://www.fonts2u.com/ dispones de fuentes bajo licencia Freeware.

- **Punto 3.7: Leer/Repasar.**

Propiedades CSS para tablas:

Propiedad:valor	**Significado**
border-collapse: collapse | separate | inherit	Determina la fusión de bordes
border-spacing: <medida> <medida> | inherit	Separación de bordes horizontal y vertical
empty-cells: show | hide | inherit	Celdas vacías
caption-side: top | bottom | inherit	Posición del título

Ejemplo. Vamos a construir una tabla con 3 colores, 1 color para la fila head y el resto de filas intercambiando el color blanco y verde o gris muy claro.

```
<html>
   <head>
       <style type="text/css">
          #clientes {font-family:Arial, Helvetica; text-align:center; width:100%;
                     border-collapse: collapse;}
          #clientes th {font-size:1.2em; padding-top:5px;  padding-bottom:4px;
                        background-color:#AC4; color:#fff;}
          #clientes td {font-size:1em; border:1px solid #98b;
                        padding:4px 7px 2px 7px;}
          #clientes .alt {color:#000; background-color:#EFD;}
       </style>
   </head>
   <body>
       <table id="clientes">
          <tr>
              <th>Empresa</th><th>Contacto</th><th>Pais</th>
          </tr>
          <tr>
              <td>Mercedes Benz</td><td>Maria Naidistch</td><td>Alemania</td>
          </tr>
          <tr class="alt">
              <td>Volvo</td><td>Cristina Carlsen</td><td>Suecia</td>
          </tr>
          <tr>
              <td>Seat</td><td>Francisco Valle</td><td>España</td>
          </tr>
          <tr class="alt">
              <td>Rover</td><td>Elena Short</td><td>Reino Unido</td>
          </tr>
       </table>
    </body>
</html>
```

Empresa	Contacto	Pais
Mercedes Benz	Maria Naidistch	Alemania
Volvo	Cristina Carlsen	Suecia
Seat	Francisco Valle	EspaÃ±a
Rover	Elena Short	Reino Unido

- **TEMA 4: XML. Almacenamiento de datos**

XML (eXtensible Markup Language – Lenguaje de Marcado eXtensible) tiene las siguientes características:

- Es un estándar (o norma), no es una implementación concreta.
- Es un metalenguaje de marcas, lo que significa que no dispone de un conjunto fijo de etiqueta que todo el mundo debe conocer.
- Permite definir a los desarrolladores los elementos que necesiten y con la estructura que mejor les convenga.
- Define una sintaxis general para maquetar datos con etiquetas sencillas y comprensibles para cualquier humano.
- Provee un formato estándar para documentos informáticos.
- Es un formato flexible, de manera que puede ser adaptado al campo de aplicación que se desee.

    - **Página 167: ¿Qué no es XML?**

- XML no es un lenguaje de programación, de manera que no existen compiladores de XML que generen ejecutables a partir de un documento XML.
- XML no es un protocolo de comunicación, de manera que no enviará datos por nosotros a través de internet (como tampoco lo hace HTML). Protocolos de comunicación son HTTP (Hyper-Text Transfer Protocol – Protocolo de Transferencia de Hipertexto), FTP (File Transfer Protocol – Protocolo de

Transferencia de Ficheros), etc. Estos y otros protocolos de comunicación pueden enviar documentos con formato XML.

- XML no es un sistema gestor de base de datos. Una base de datos relacional puede contener campos del tipo XML. Existen, incluso, bases de datos XML nativas, que todo lo que almacenan son documentos con formato XML. Pero XML en sí mismo no es una base de datos.
- No es propietario, es decir, no pertenece ninguna compañía, como sucede con otros formatos.

- **Página 175: Espacio de nombres (solo página 175).**

Es un mecanismo para para evitar conflictos de nombres, de manera que se pueden diferenciar elementos o atributos dentro de un mismo documento XML que tengan idénticos nombres pero diferentes definiciones. No aparecieron en la primera especificación de XML, pero si pronto después.

Se declaran como atributo de elementos de la forma:

<nombre_elemento xmlns:prefijo="URI_del espacio_de_nombres">

Y se usan anteponiendo a elementos y atributos con el prefijo asociado al espacio de nombres además del carácter dos puntos ":".

Ejemplo:

```
<info:pedido xmlns:info="empresa:espacios:info">
    <info:ítem info:id="i_13">Afeitadora eléctrica</info:ítem>
    ...
</info:pedido>
```

Cualquier cadena de texto puede ser usada como prefijo del espacio de nombres. El URI del espacio de nombres sí deber ser único, aunque realmente no se comprueba mediante conexión alguna. El URI no es más que un nombre lógico del espacio de nombres. A veces se usa como URI http//~.

- **Página 179: ¿Qué es un Parser XML?**

Un **analizador XML** (llamado en ingles ***parser***), es un procesador que lee un documento XML y determina la estructura y propiedades de los datos en él contenidos. Un analizador estándar lee el documento XML y genera el árbol jerárquico asociado, lo que permite ver los datos en un navegador o ser tratados por cualquier aplicación.

Si el analizador comprueba las reglas de buena información y además valida el documento contra un DTD o esquema, se dice que se trata de un **analizador validador**. Esto analizadores también comprueban la semántica del documento e informan de los errores existentes.

Existen validadores XML en línea, como XML Validation (http://www.xmlvalidation.com).

- **Punto 4.6: Documentos XML válidos.**

Hasta ahora se ha visto qué componentes forman un documento XML (elementos, atributos, comentarios ...) y qué reglas deben de cumplir para que el documento sea bien formado, es decir, sea sintácticamente correcto.

Nada se ha dicho sobre qué elementos o atributos concretos pueden aparecer, en qué orden, qué valores pueden tomar, etc.

Si se hiciera esto, se estaría definiendo un lenguaje XML concreto, con su vocabulario (elementos y atributos permitidos), las relaciones entre elementos y atributos, los valores permitidos para ellos, etc.

Se dice que un documento XML es válido si existen unas reglas de validación (por ejemplo en forma de definición de tipo de documento (DTD)) asociadas a él, de manera que el documento deba cumplir con dichas reglas. Estas reglas especifican la estructura gramatical y sintaxis que debe tener el documento XML.

Todo documento XML válido está bien formado, pero no a la inversa.

Ejemplo: Para entender la diferencia entre un documento bien formado y un documento válido, veremos el equivalente con una frase en castellano.

montaña La nevada está <- -> La montaña está nevada.

Ambas frases están bien formadas en castellano desde el punto de vista léxico, es decir, las palabras que en ellas aparecen son correctas en castellano. Ahora bien, desde el punto de vista gramatical, la primera se trata de una frase incorrecta (no se cumple la clásica regla de sujeto + predicado). Así, sólo la segunda frase es válida con respecto a la gramática castellana.

- **Página 188 en adelante. Componentes del DTD.**

Hay cuatro tipos posibles de componentes que se pueden declarar en un DTD:

- **Elemento <!ELEMENT>.**
- **Atributo <!ATTLIST>.**
- **Entidad <!ENTITY>.**
- **Notación <!NOTATION>.**

<!ELEMENT>

Es una declaración de tipo de elemento. Indica la existencia de un elemento en el documento XML.

Sintaxis general: **<!ELEMENT nombre_elemento modelo_contenido>**

El nombre del elemento tiene que ser el mismo que el correspondiente del documento XML sin los signos de menor (<) y mayor (>) propios de las etiquetas.

El modelo de contenido puede indicar:

- Una **regla**, en cuyo caso será:
  - **ANY** (cualquier cosa): se puede utilizar al construir el DTD para dejar la descripción de un elemento como válida en cualquier caso, eliminando cualquier comprobación sintáctica. Es un comodín que no debe aparecer en el DTD definitivo.
  - **EMPTY** (elemento vacío): describe un elemento sin descendientes.
- **Datos (caracteres)**, sean textuales, numéricos o cualquier otro formato que no contenga marcas (etiquetas). Se describe cómo **#PCDATA (Parsed Character Data – Datos de Caracteres Procesados)** y debe aparecer entre paréntesis.
- **Elementos descendientes.** Su descripción debe aparecer entre paréntesis y se basa en las siguientes reglas.

  Cardinalidad de los elementos:

  Para indicar el numero de veces que puede un elemento o una secuencia de elementos existen ciertos símbolos:

Símbolo	Significado
?	El elemento (o secuencia de elementos) puede aparecer 0 o 1 vez
*	El elemento (o secuencia de elementos) puede aparecer de 0 a N veces.
+	El elemento (o secuencia de elementos) puede aparecer de 1 a N veces.

  Secuencias de elementos:

  En las secuencias de elementos se utilizan símbolos para indicar el orden en que un elemento debe aparecer, o bien si aparece como alternativa a otro:

Símbolo	Significado
A,B	El elemento B aparecerá a continuación del elemento A.
A|B	Aparecerá el elemento A o B, pero no ambos.

Se pueden combinar el uso de los símbolos de cardinalidad de los elementos con los de las secuencias de elementos.

- **Contenido mixto**, se mezcla de texto más elementos descendientes. No se puede usar el cuantificador + con un contenido mixto, por eso se debe usar el *.

<!ATTLIST>

Es una declaración de tipo de atributo. Indica la existencia de atributos de un elemento en el documento XML. En general se utiliza un solo ATTLIST para declarar todos los atributos de un elemento (aunque se podría usar un ATTLIST para cada atributo).

Sintaxis general: **<!ATTLIST nombre_elemento**
nombre_atributo tipo_atributo carácter
nombre_atributo tipo_atributo carácter
...>

El **nombre** del atributo tiene que ser un nombre XML válido.

El **carácter** del atributo puede ser:
- Un **valor textual entre comillas**, que representa un valor por defecto para el atributo.
- **#IMPLIED,** el atributo es de carácter opcional y no se le asigna ningún valor por defecto.
- **#REQUIRED**, el atributo es de carácter obligatorio, pero no se le asigna un valor por defecto.
- **#FIXED**, el atributo es de carácter obligatorio y se le asigna un valor por defecto que además es el único valor que puede tener el atributo.

Los posibles **tipos de atributo** son:
- **CDATA:** caracteres que no contienen etiquetas.
- **ENTITY:** el nombre de una entidad (que debe declararse en el DTD).
- **ENTITIES:** una lista de nombres de entidades (que deben declararse en el DTD), separadas por espacios.
- **Enumerado:** una lista de valores de entre los cuales, el atributo debe tomar uno.
- **ID:** un identificado único. Se usa para identificar elementos, es decir, caracterizadas de manera única. Por ello, dos elementos no pueden tener el mismo valor en atributos de tipo ID. Además, un elemento puede tener a lo sumo un atributo de tipo ID. El valor asignado a un atributo de este tipo debe ser un nombre XML válido.
- **IDREF:** representa el valor de un atributo ID de otro elemento, es decir, para que sea válido, debe existir otro elemento en el documento XML que tenga un atributo de tipo ID y cuyo valor sea el mismo que el del atributo de tipo IDREF del primer elemento.
- **IDREFS:** representa múltiples IDs de otros elementos, separados por espacios.
- **NMTOKEN:** cualquier nombre sin espacios en blanco en su interior. Los espacios en blanco anteriores o posteriores se ignorarán.
- **NMTOKENS:** una lista de nombres, sin espacios en blanco en su interior (los espacios en blanco anteriores o posteriores se ignoraran), separados por espacios.
- **NOTATION:** un nombre de notación, que debe estar declarada en el DTD.

<!ENTITY>

Este es un elemento avanzado en el diseño de DTDs. Es una declaración de tipo de entidad.

Hay diferentes tipos de entidades y, en función del tipo, la sintaxis varia:
- Referencia a entidades generales (internas o externas).
- Referencia a entidades parámetro (internas o externas).
- Entidades no procesadas (unparsed).

- **Referencias a entidades generales**, se utilizaran dentro del documento XML.

Sintaxis general: **<!ENTITY nombre_entidad definición_entidad>**

A continuación, se usa en el XML anteponiendo al nombre de la entidad el carácter (&) y a continuación un carácter punto y coma (;). El programa analizador del documento realizará la sustitución.

Sintaxis general: **&nombre_entidad;**

- **Referencias a entidades generales externas,** ubicadas en otros archivos.

  Sintaxis general: **<!ENTITY nombre_entidad tipo_uso url_archivo>**

  Siendo el tipo de uso privado **(SYSTEM)** o público **(PUBLIC).**

- **Referencias a entidades parámetro,** que se usarán en el propio DTD y funcionan cuando la definición de las reglas del DTD se realiza en un archivo externo.

  Sintaxis general: **<!ENTITY % nombre_entidad definición_entidad>**

- **Referencias a entidades parámetro externas,** ubicadas en otros archivos.

  Sintaxis general: **<!ENTITY % nombre_entidad tipo_uso fpi url_archivo>**

  Siendo el tipo de uso privado (SYSTEM) o público (PUBLIC). Si es PUBLIC, es necesario definir el FPI que, recordemos, es el nombre por el cual se identifica públicamente un determinado elemento (sea DOCTYPE, un ELEMENT o una ENTITY), en este caso la entidad.

- **Entidades no procesadas,** referencian a datos que no deben ser procesados por el analizador XML, sino por la aplicación que lo use (como una imagen).

  Sintaxis general: **<!ENTITY % nombre_entidad tipo_uso fpi valor_entidad NDATA tipo>**

  NOTA: En ocasiones se utiliza otra tecnología, llamada XLink, en sustitución de las entidades no procesadas.

<!NOTATION>

Este es un elemento avanzado en el diseño de DTDs. Es una declaración del tipo de atributos NOTATION. Una notación se usa para especificar un formato de datos que no sea XML. Se usa con frecuencia para describir tipos MIME, como image/gif o image/jpg.

Se utiliza para indicar un tipo de atributo al que se le permite usar un valor que haya sido declarado como notación en el DTD.
Sintaxis general de la notación:

<!NOTATION nombre_notacion SYSTEM "identificador_externo">

Sintaxis general del atributo que la usa:

<!ATTLIST nombre_elemento nombre_atributo NOTATION valor_defecto>

SECCIONES CONDICIONALES: Permite incluir o excluir reglas en un DTD en función de condiciones. Sólo se pueden ubicar en DTDs externos. Su uso tiene sentido al combinarlas con referencias a entidades parámetro. Las secciones condicionales son IGNORE e INCLUDE, teniendo la primera precedencia sobre la segunda.

- **Página 190: Ejercicio 4.4**

Se quiere que el elemento <grupoSanguineo> tenga como único elemento descendiente a uno solo de los cuatro siguientes A, B, AB ó O. Indica cuál de las siguientes es una declaración correcta del citado elemento.

a. **<!ELEMENT grupoSanguineo (A ? B ? AB ? O) >**

b. **<!ELEMENT grupoSanguineo (A , B , AB , O) >**
c. **<!ELEMENT grupoSanguineo (A | B | AB | O) >**
d. **<!ELEMENT grupoSanguineo (A + B + AB + O) >**

- **Punto 4.10: Lenguajes basados en XML. Conocer las siglas y que son.**

Existen lenguajes basado en XML que se usan para propósitos específicos. Estos lenguajes tendrán su mecanismo de validación (DTD, esquema XML...) que fijará qué elemento y atributos concretos pueden aparecer, con qué valores, en que orden...

SVG (Scalable Vector Graphics – Gráficos Vectoriales Escalables) es un lenguaje de representación de gráficos vectoriales bidimensionales. Desde el año 2001 es una recomendación del W3C, por lo que muchos navegadores son capaces de mostrar gráficos en este formato.

WML (Wireless Mark-up Language – Lenguaje de Marcado Inalámbrico) es un lenguaje de representación de la información que se visualiza en las pantallas de los dispositivos móviles que utilicen el **protocolo WPA (Wireless Application Protocol – Protocolo de Aplicaciones Inalambricas).**

RSS (Really Simple Syndication – Sindicación Realmente Simple) es una familia de formatos de semilla web usadas para publicar información que se actualiza con frecuencia.

Atom (Atom Syndication Format – Formato Atomico de Sindicación) es un lenguaje usado como semillas web. Es una alternativa a RSS.

DocBook. Es un lenguaje de definición de documentos. Una herramienta interesante es XMLMind XML Editor (http://www.xmlmind.com/xmleditor).

XBRL (eXtensible Business Reportin Language – Lenguaje Extensible de Informes Financieros) es un estandar que permite representar la información y la expresión de la semántica requerida en los informes financieros.

- **Punto 4.11. Lenguajes de almacenado de información. Conocer siglas y que son.**

JSON (JavaScript Object Notation – Notación de Objetos JavaScript) es, de igual manera que XML, un formato en modo texto para el almacenamiento y transmisión de información.

Se usa frecuentemente como alternativa a XML en el envio de datos, por ejemplo en ***AJAX (Asynchronous JavaScript And XML – JavaScript Asíncrono y XML).*** El motivo fundamental es que el procesado de un documento en formato JSON es mas sencillo que el de un documento XML.
Permite representar objetos en aplicaciones ***RIA (Rich Internet Application – Aplicaciones Ricas de Internet)*** que usan JavaScript.

El término JSON se sustituye en ocasiones por el de ***LJS (Literal JavaScript)***.

JSON se apoya en dos estructuras de datos:
- Una **colección de pares nombres/valor**. En algunos lenguajes de programación a esto se le denomina objetos, registro, diccionario, tabla hash o array asociativo.
- Una **lista ordenada de valores**, lo que en otros lenguajes se denomina array, vector, lista o secuencia.

Un **objeto** es un conjunto desordenado de pares etiqueta/valor. La definición de un objeto comienza con la llave de apertura "{" y termina con la llave de cierre "}". Cada nombre va seguido del carácter dos puntos ":" y los pares nombre/valor se separan por el carácter coma ",".

Un **array** es una colección ordenada de valores. Comienza por el carácter corchete de apertura "[" y termina con el carácter corchete de cierre "]". Los valores se separan por el carácter coma ",".

Un **valor** puede ser:

- Una cadena de texto entre comillas dobles
- Un número.
- True, false o null.
- Un objeto.
- Un array.

Una **cadena de texto** es una secuencia de ceros o mas caracteres Unicode, rodeados por dobles comillas. Se usa el carácter barra invertida (\) como marca de escape. Un carácter se representa como una cadena de un solo carácter. Es equivalente al String de Java.

Un **número** es como los número en Java. Puede tener parte entera, entera + decimal, entera + exponente y entera + decimal + exponente.

Sintaxis:

objeto ::= {} | {miembros}
miembros ::= par | par,miembros
par ::= string : valor
array ::= [] | [elementos]
elementos ::= valor | valor,elementos
valor ::= string | numero | objeto | array | true | false | null

Existen herramientas de conversión automática entre JSON y XML:

- http://json.online-toolz.com/tools/xml-json-convertor.php
- http://jsontoxml.utilities-online.info

YAML (acrónimo recursivo de YAML Aint't Markup Language – YAML no es otro lenguaje de marcas, aunque también se refiere como Yet Another Markup Language), es un formato de almacenamiento de información o serialización de datos, ligero y de fácil lectura para el ojo humano. Comparte con JSON ciertos elementos como las **listas** y los **arrays asociativos.**

Una **lista en formato bloque** donde cada elemento se denota por un guion "-".

- **Elemento1**
- **Elemento2**
- **Elemento3**

Una **lista en formato lineal**, donde los elementos se rodean de un corchete de apertura "[" y uno de cierre "]" y se separan entre sí por comas ",".

[Elemento1,Elemento2,Elemento3]

Un **array asociativo en formato de bloque**, en el que aparece la etiqueta, el carácter dos puntos ":" y el valor.

Etiqueta1:ValorEtiqueta1
Etiqueta2:ValorEtiqueta2

Un **array asociativo en formato lineal,** donde los pares etiquetas/valor se rodean por una llave de apertura "{" y una de cierre "}" y se separan entre si por comas ",".

{Etiqueta1:ValorEtiqueta1 , Etiqueta2:ValorEtiqueta2}

A partir de estos elementos básicos se pueden construir estructura más complejas: listas de arrays asociativos, arrays asociativos de listas ...

- **Repasar ejercicios de las PACs, XML y DTD.**

- **TEMA 5: Tratamiento y recuperación de datos**

  - **Punto 5.3: XPath. ¿Qué es y para que se utiliza?**

XPath forma junto con XSLT y XSL-FO, una familia de lenguajes llamada XSL, diseñados para acceder, transformar y dar formato de salida a documentos XML.

Un documento XML tiene una estructura de árbol: un elemento raíz que tiene hijos, que a su vez tienes otros hijos, etc. A su vez, cada elemento puede tener atributos y/o contenido textual.

Más en concreto se asemeja a un árbol genealógico, donde aparecerán conceptos como padre (se usará de manera genérica el género masculino, pero es intercambiable por madre), hijo, hermano, antepasado (o ascendiente), descendiente...

O también parecida a un sistema de ficheros: elementos que contienen otros elementos o datos, frente a directorios que contienen otros directorios o archivos. Para recorrer y listar la estructura de directorios usaríamos los comandos del sistema operativo: cd, dir o ls... Para recorrer un documento XML y extraer la información contenida en sus elementos se usará XPath.

XPath es un lenguaje de expresiones que no se usa de manera independiente, sino que se emplea en el contexto de un lenguaje anfitrión. Se trata de un lenguaje bastante sofisticado.

Existen diferentes versiones de XPath. La versión más reciente es la 2.0, que introduce algunas novedades respecto a la 1.0.

- **Página 268. Direccionamiento.**

Las expresiones en XPath puede ser absolutas (empiezan por /) o relativas (con respecto a un nodo determinado, llamado nodo de contexto)

XPath trata un documento XML como un árbol de nodos. Los tipos de nodos son:
- Raiz del documento (uno por documento): contiene todo el documento y se representa por el símbolo /. No tiene nodo padre y tiene al menos un nodo hijo, el nodo documento). Contiene también los comentarios e instrucciones de procesamiento, que no forman parte del documento en si.
- Elemento: todo elemento de un documento XML es un nodo de XPath.
- Atributo: todo atributo de un documento XML, es un nodo XPath.
- Texto: todo contenido textual de un elemento es un nodo XPath.
- Comentario.
- Instrucciones de procesamiento.
- Espacio de nombres.

Los tipo de datos básicos son:
- string.
- number.
- boolean.
- node-set (conjunto de nodos).

Las expresiones XPath más comunes son:

Expresion XPath	**Coincidencia**
elemento	Elemento de nombre *elemento*
/elemento	Elemento de nombre *elemento* ubicado en la raíz del documento
e1/e2	Elemento e2 hijo directo de elemento e1
e1//e2	Elemento e2 descendiente (hijo, nieto, bisnieto ...) del elemento e1
//elemento	Elemento de nombre *elemento* ubicado en cualquier nivel por debajo de la raíz del documento
@atributo	Atributo de nombre *atributo*
*	Cualquier elemento (todos los elementos)
@*	Cualquier atributo (todos los atributos)
.	Nodo actual
..	Nodo padre
espNom:*	Todos los elementos en el espacio de nombre de prefijo espNom
@espNom:*	Todos los atributos en el espacio de nombres de prefijo espNom

Para probar las expresiones XPath, se podrá usar:
- Un entorno de desarrollo como el que ofrece BaseX, con ayudas al usuario sobre que elemento descienden de cuales. Se recomienda esta opción.

- Un analizador de expresiones XPath en línea, como por ejemplo: http://www.whitebeam.org/library/guide/TechNotes/xpath.rhtm. El elemento FirePath de Firebug de MozillaFirefox.

- **Punto 5.4: XQuery. ¿Qué es y para que se utiliza?**

XQuery es un lenguaje de consulta que permite extraer y procesar información almacenada en formato XML, habitualmente en bases de datos nativas XML o en tablas y campos de tipo XML en bases de datos relacionales.

Se parece a SQL (Standard Query Language – Lenguaje de Consultas Estandar) en algunas de las clausulas empleadas (where, order by) comunes en ambos lenguajes.

También se asemeja a XPath, con el que comparte modelo de datos y soporta las mismas funciones y operadores. Se podría considerar a XQuery como un superconjunto de Xpath, ya que toda expresión XPath es una expresión XQuery válida.

- **Punto 5.5: ¿Qué es y para que se utilizan?**

  - **SAX.**

SAX (Simple API for XML – API Simple para XML), originally a Java-only API. SAX fue la primera API masivamente usada para manejar XML en Java. Es un estándar de facto.

Está basada en eventos y muy orientada a la lectura de documentos XML. Se desarrolló inicialmente para analizadores escritos en Java, aunque luego han aparecido implementaciones para otros lenguajes: C++, Python, Perl ...

El modelo de recorrido de nodos basado en eventos difiere del basado en una estructura en árbol, ya que, según el analizador va leyendo el documento XML, va enviando notificación de ello al programa en tiempo real. No es necesario leer el documento entero para poder trabajar sobre los datos que se encuentran al principio del mismo. De hecho, el documento no reside en memoria, de manea que es la elección idónea para documentos grandes que no caben en la memoria disponible. Sin embargo, no sería adecuado si se quisiera realizar alguna actualización en memoria de los datos para luego volcarlos en el documento.

SAX está compuesto de un conjunto de interfaces contenidas en el paquete org.xml.sax. Una de ellas es XMLReader, que representa un analizador XML.

  - **DOM.**

DOM (Document Object Model – Modelo de Objetos de Documento) es un API para acceder y manipular documentos XML tratándolos como estructura de árbol de nodos. Se define como un conjunto de recomendaciones que describen el modelo de objetos, independiente de cualquier lenguaje de programación, usado para almacenar documentos jerárquicos en memoria (como XML, HTML o XSLT).

La jerarquía de objetos DOM almacena referencias entre los nodos de un documento, de manera que el documento completo debe ser leído y procesado antes de que pueda estar disponible para una aplicación DOM. Esto fuerza, además, a que el documento entero se encuentre en memoria, lo que supone, en ocasiones, una cantidad significativa de carga y convierte a esta técnica en inadecuada para procesar documentos de gran tamaño.

Para aplicaciones que requieren acceder aleatoriamente a diferentes posiciones del árbol y en diferentes momentos, o aplicaciones que requieren la modificación dinámica de la estructura de un documento XML, DOM es una tecnología muy adecuado.

Se utiliza la interfaz genérica Node como elemento principal. Representa un nodo del árbol con tres enlaces definidos con su nodo padre, a sus nodos hijos y a sus nodos hermanos.

También existen interfaces específicas para los distintos tipos de nodo: elemento, atributo, instrucciones de procesamiento, etc. Todas ellas derivan de Node.

Realmente la interfaz Node no se instancia directamente, pero al ser la interfaz raíz de todas las específicas, permite extraer información de cualquier objeto DOM sin saber su tipo.

- **JAXP.**

JAXP (Java API for XML Processing – API de Java para Procesamiento de XML) es un API que permite a las aplicaciones Java procesar, transformar, validar y consultar documentos.

Viene incluida en JDK 1.6 como parte de los desarrollos del proyecto Xerces de Apache.

Contiene tres grupos de paquetes:

- JAXP
  - Constantes
  - Tipos de datos
  - Espacios de nombres
  - Analizadores
  - Transformaciones
  - Transformaciones DOM
  - Transformaciones SAX
  - Transformaciones de flujos
  - XPath
- DOM
  - Interfaces DOM
  - Cargar y grabar
  - Eventos
  - HTML
  - Transversal
  - XPath
- SAX
  - Interfaces SAX
  - Clases helper
  - Extensiones

- **XPOINTER.**

XPointer (XML Pointer Language – Lenguaje de Punteros XML) es una especificación del W3C para vincular fragmentos de documentos XML (http://www.w3.org/TR/WD-xptr). Al igual que el XLink, con el que conjuntamente se utiliza, es una tecnología que no ha tenido un gran desarrollo e implantación. Se apoya en XPath para identificar los fragmentos del documento XML que se enlazaran.

Existen diversas funciones predefinidas que permiten acceder a diversos puntos en un documento XML: here(), origin(), point(), start-point(), range() ...

- **XLINK.**

XLink (XML Linking Languaje – Lenguaje de Vinculación para XML) es un estándar del W3C para vincular documentos XML (http://www.w3.org/TR/xlink). Es una tecnoloqia no ha tenido un gran desarrollo e implantación.

Con XLink se pueden vincular documento XML mediante **enlaces simples**, equivalentes a la etiqueta <a> de HTML que permiten vincular un documento con otro, y mediante **enlaces extendidos** que permiten vincular varios documentos entre sí.

- **JAXB.**

JAXB (Java Architecture for XML Binding – Arquitectura Java para manejo de XML) es otra API, alternativa a JAXP, que pretende facilitar el manejo de documentos XML así como optimizar el rendimiento.

Los paquetes que incluyen las interfaces y clases que constituyen esta API no forman parte de la propia distribución de Java y deben descargarse de manera independiente (http://jaxb.java.net). La última versión en el momento de la publicación del libro es la 2.2.5.

De manera simplificada, las tareas que se desarrollarán con JAXB serán:

- Leer un documento XML a memoria (deserializar).
- Recorrer y/o manipular la estructura creada.
- Guardar en un archivo la estructura de memoria.

Los pasos para leer un documento XML a memoria son:

- Vincular (**bind**) el esquema XSD:

  JAXB requiere que el documento XML que se quiera procesar tenga un esquema XSD asociado que lo valide.

  La vinculación del esquema consiste en la generación de una serie de clases en Java que representan el esquema y es realizada por una aplicación llamada compilador de vinculación, que viene con la distribución de JAXB. Se ejecuta independientemente desde la línea de comandos.

  La respuesta del compilador de vinculación son una serie de archivos .java, que contienen interfaces y clases que implementan estas interfaces.

- Convertir el documento XML en objetos de Java, lo que se conoce como **deserialización o unmarshal**.

  Se crea un **árbol de objetos de contenido** que representa al documento, pero no es un árbol DOM, sino que es una estructura más eficiente en el uso de la memoria.

  Los **objetos de contenido** son instancias de las clases producidas por el compilador de vinculación.

Si lo que se quisiera es escribir el contenido de un documento XML ya en memoria en un archivo y suponiendo que la vinculación del esquema ya se ha realizado para llegar hasta aquí:

- Crear el árbol de contenido:

  Se puede hacer a partir de un documento XML existente por el proceso de unmarshal, o desde cero con la clase ObjectFactory, que permite crear los objetos del árbol de contenido.

- Volcar el árbol de contenido en un documento XML, lo que se conoce como **serialización o marshal**. Este proceso es el opuesto al unmarshal.

Al usar esta API, además de los propios programas, se generan muchos archivos auxiliares, como las interfaces que representan los elementos del esquema XML y las clases que las implementan.

    - **Repasar Recursos sobre ejercicios ejemplo XPath.**

- **TEMA 6: Transformación de documentos XSLT**

    - **Punto 6.2: XSLT. Todo página 306-307.**

XSLT (eXtensible Sylesheet Language for Transformations) es un estandar del W3C que aparece recomendado por primera vez por este organismo en 1999. La versión actual de XSLT en el momento de publicación de este libro es la 2.0.

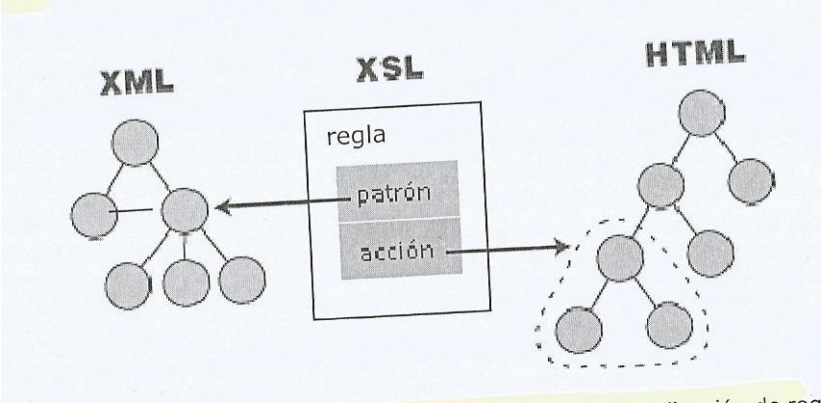
XSLT está basado en XML, de manera que cada hoja de transformaciones es un documento XML bien formado.

Se trata de un lenguaje de programación declarativo que permite generar documentos, en diferentes formatos de salida (XML, HTML, texto, PDF, RTF, etc), a partir de un documento XML.

Se describe como declarativo porque consiste en la declaración de una serie de reglas o plantillas que hay que aplicar a un documento XML para transformarlo en la salida deseada. Una regla asocia un patrón (expresión) con elementos del documento XML de partida y les aplica una serie de acciones.

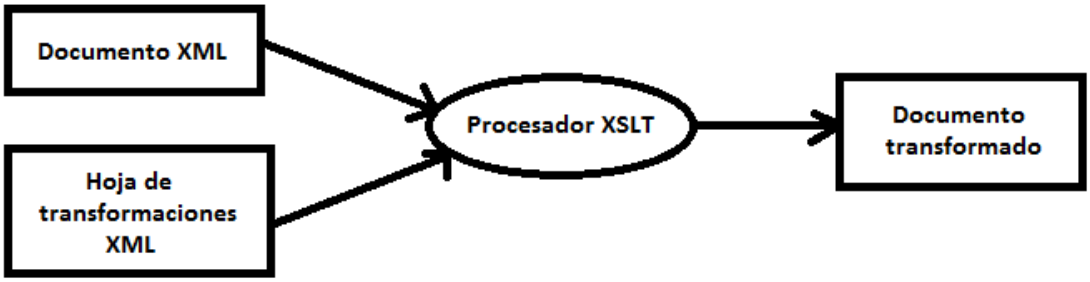
Estas reglas se almacenarán en un documento de texto, que habitualmente tiene la extensión .xsl que, junto con el documento .xml de partida serán pasados como parámetros a un procesador XSLT, que generará como salida el nuevo documento transformado.

- **Figura 6.1:**



Transformación de XML en HTML mediante la aplicación de reglas

- **Figura 6.2.**



Procesamiento XSLT

- **Página 307: CSS vs XSLT.**

El W3C ha definido dos familias de normas para hojas de estilos. La más antigua y simple es CSS, que permite definir propiedades de elementos de marcado (una letra en negrita o de un cierto tamaño, un recuadro con bordes verdes discontinuos...).

Sin embargo, hay ciertas **tareas que no se pueden acometer con CSS:**

- **No se puede cambiar el orden en el que los elementos de un documento HTML se visualizan** (no puede ordenar elementos o filtrarlos según algún criterio).
- **No se pueden realizar operaciones**, como sumar los valores de todos los elementos <precio> de un documento.
- **No se pueden combinar múltiples elementos**, como todas las nóminas anuales de un empleado con el fin de obtener un total de ingresos y retenciones.

Estas limitaciones que aparecen con el uso de CSS se superan con XSLT. De hecho, **en muchas ocasiones se usará CSS y XSLT de manera complementaria**. Con XSLT se

determinará qué contenido de un documento XML, se quiere mostrar y en qué orden, realizándose si es necesario algún cálculo sencillo; con CSS se dará un formato visual agradable a la información mostrada.

- **Página 307: Procesador XSLT.**

Son aplicaciones capaces de procesar documentos XML mediante hojas de transformaciones XSLT.

Cuando el procesador XSLT lee un documento XML genera una representación de dicho documento como un árbol de nodos. Los tipos de nodo son: elemento, atributo, texto, comentario, instrucción de procesamiento y espacio de nombres. Las relaciones entre los nodos son las mismas que en un árbol genealógico: padres, hijos, ascendientes, descendientes...

Los comentarios y las instrucciones de procesamiento son parte del árbol de nodos y deben ser tenidos en cuentas a la hora de contar nodos o iterar entre ellos.

El procesador XSLT va recorriendo y procesando nodo a nodo. El nodo que se está tratando en un momento dado se denomina **nodo de contexto**.

Existen diferentes tipos de procesadores:

- Procesadores en línea.
- Procesadores instalables, como Xalan, desarrollado en Java y ejecutable desde la línea de comandos.
- Aplicaciones que se han introducido ya como editores de XML, de DTDs, de esquemas XML, validadores XML... también permiten realizar transformaciones. Algunas de la más destacadas son:
  - Altova StyleVision
  - <oXigen/>
  - XML Copy Editor

- **Página 310: Estructura básica. Mirar ejemplo recuadro punto 2.**

Una hoja de transformaciones estará compuesta por:

1. Una declaración de documento XML, puesto que lo es:

   <?xml versión=”1.0”?>

2. Un elemento raíz, llamado <xsl:stylesheet> (sinónimo a <xsl:transform> que es igualmente válido, pero en los ejemplos no se usará).

```
<xsl:stylesheet versión=”1.0”
       xmlns:xsl = “http://www.w3.org/1999/XSL/Transform”>
...
</xsl:stylesheet>
```

3. El espacio de nombres es http://www.w3.org/1999/XSL/Transform, siendo xsl el prefijo.
4. Existen otros elementos, llamados de nivel superior que, de aparecer en la hoja de transformaciones, siempre lo harán como hijos de <xsl:stylesheet>

   Son <xsl:import> (si aparece, debe ser el primer hijo del elemento raíz) , <xsl:include> , <xsl:namespace-alias>, <xsl:output>, <xsl:strip-space>, <xsl:preserve-space>, <xsl:attribute-set>, <xsl:key>, <xsl:param> (puede ser parámetro global o local), <xsl:variable> (puede ser variable global o local) y <xsl:template>. Algunos se verán muy en detalle y otros levemente.

- **Punto 6.2.1: ¿Cómo se enlaza?**

Se puede asociar de forma permanente una hoja de estilo, sea CSS o XSLT a un documento XML mediante la instrucción de procesamiento <?xml-stylesheet ?>. Esta instrucción se ubica al principio del documento XML, después de la declaración <?xml...?>

Cuando se visualiza en un navegador web un documento XML enlazado con un hoja de estilo CSS, se muestra el resultado de la aplicación de los estilos a los elementos existentes.

Ejemplo:

En el código del documento componentes.xml, en cuya segunda línea, después de la declaración de documento XML, se enlaza con la hoja de estilos componentes.css, es:

```
<?xml-stylesheet type=”text/css” href=”componentes.css”?>
```

- **Página 319: Actividad 6.1.**

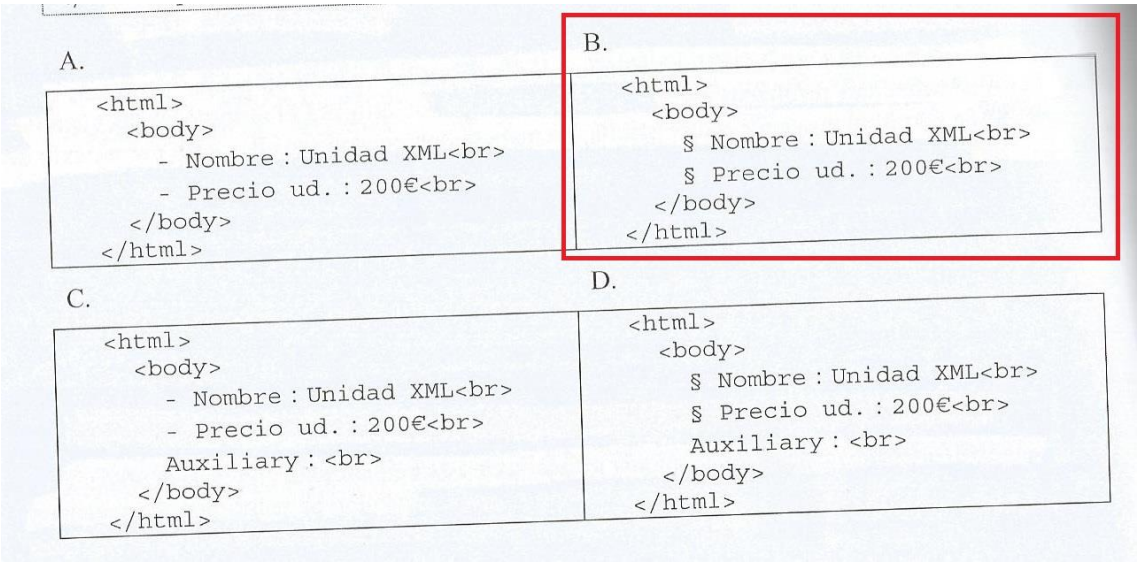
¿Cuál de los siguientes A, B, C o D, es el resultado adecuado de una transformación al aplicar al siguiente documento XML la siguiente hoja de transformaciones XSLT?

documento.css:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="documento.xsl"?>
<ListaProductos>
	<Titulo>Series XML</Titulo>
	<Producto>
		<Nombre>Unidad XML</Nombre>
		<PrecioUd>200</PrecioUd>
	</Producto>
</ListaProductos>
```

documento.xsl:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
	<xsl:template match="/">
		<html>
		  <body>
			<xsl:apply-templates select="ListaProductos/Producto"/>
			<xsl:apply-templates select="ListaProductos/Auxiliar"/>
		  </body>
		</html>
	</xsl:template>
	<xsl:template match="/ListaProductos/Producto">
		- Nombre: <xsl:value-of select="Nombre"/><br/>
		- Precio ud.: <xsl:value-of select="PrecioUd"/>e<br/>
	</xsl:template>
	<xsl:template match="/ListaProductos/Producto">
		$ Nombre: <xsl:value-of select="Nombre"/><br/>
		$ Precio ud.: <xsl:value-of select="PrecioUd"/>e<br/>
	</xsl:template>
	<xsl:template match="Auxiliar">
		Auxiliar: <br/>
		<xsl:value-of select="."/><br/>
	</xsl:template>
</xsl:stylesheet>
```



- **Página 326: Ejemplo.**

Se dispone de un documento XML de partida que representa una tabla con palabras en castellano y en inglés. Se va a convertir en un documento HTML de salida que dibuje la tabla, con las palabras ya mencionadas precedidas de la posición de las mismas en la tabla.

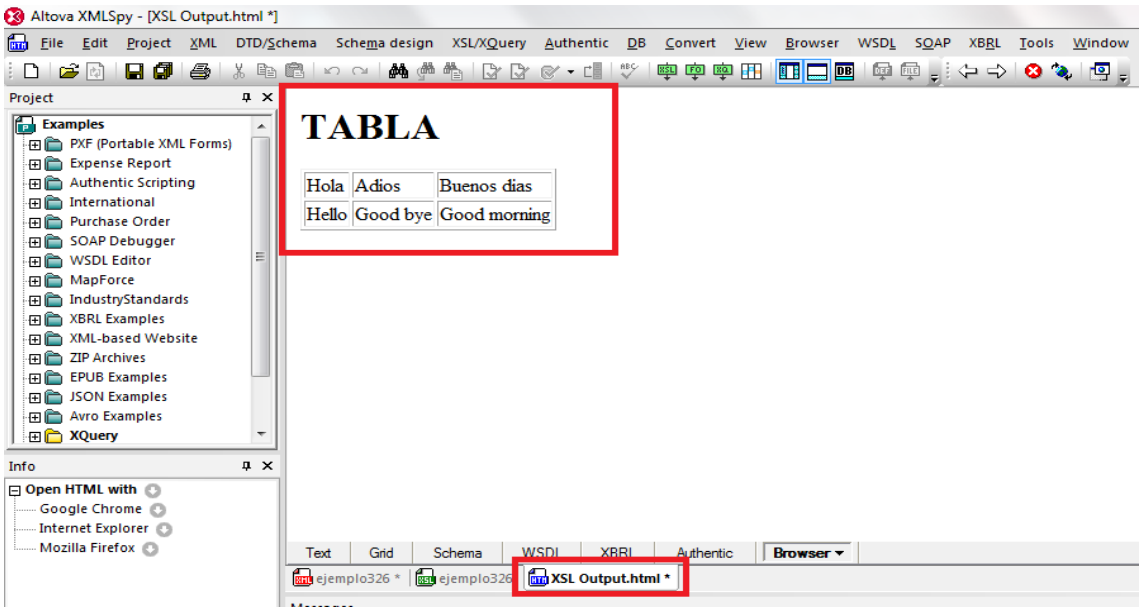
```
<?xml version="1.0" encoding="iso-8859-1"?>
<tabla>
	<fila>
		<columna valor="Hola"/>
		<columna valor="Adiós"/>
		<columna valor="Buenos días"/>
	</fila>
	<fila>
		<columna valor="Hello"/>
		<columna valor="Good bye"/>
		<columna valor="Good morning"/>
	</fila>
</tabla>
```

La hoja de transformaciones será:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
	<xsl:output method="html" encoding="iso-8859-1"/>
	<xsl:template match="/tabla">
		<html>
		    <body>
		       <h1>TABLA</h1>
		       <!-- Empieza a generar la tabla si al menos tiene una fila -->
		       <xsl:if test="count(fila) &gt; 1">
		           <table border="1">
		              <xsl:for-each select="fila">
		                  <tr>
		                     <xsl:for-each select="columna">
		                         <td> <xsl:value-of select="./@valor"/> </td>
		                     </xsl:for-each>
		                  </tr>
		              </xsl:for-each>
		           </table>
		       </xsl:if>
		    </body>
		</html>
	</xsl:template>
</xsl:stylesheet>
```

La salida se queda:

```
<html>
<body>
<h1>TABLA</h1>
<table border=”1”>
<tr>
	<td>Hola</td>
	<td>Adios</td>
	<td>Buenos dias</td>
</tr>
<tr>
	<td>Hello</td>
	<td>Good bye</td>
	<td>Good morning</td>
</tr>
</table>
</body>
<html>
```



El mismo efecto se hubiera conseguido al aplicar una hoja de transformaciones con dos plantillas una que se aplique para cada fila y otra que se aplique para cada columna dentro de una fila.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
	<xsl:output method="html" encoding="iso-8859-1"/>
	<xsl:template match="/tabla">
		<html><body>
			<h1>TABLA</h1>
			<!--Empieza a generar la tabla si al menos tiene una fila-->
			<xsl:if test="count(fila) &gt; 1">
				<table border="1">
					<!--Se invoca la plantilla para cada
						elemento/tabla/fila-->
					<xsl:apply-templates select="fila"/>
				</table>
			</xsl:if>
		</body></html>
	</xsl:template>
	<xsl:template match="fila">
		<tr>
```

```
                    <!—Se invoca la plantilla para cada elemento
                           /tabla/fila/columna-->
                    <xsl:apply-templates select="columna"/>
             </tr>
      </xsl:template>
      <xsl:template match="columna">
             <td> <xsl:value-of select="./@valor"/> </td>
      </xsl:template>
</xsl:stylesheet>
```

- **Punto 6.3. XSL-FO. ¿Qué es y para que se utiliza?**

XSL-FO (XSL – Formatting Objects Processor – Procesador de Objetos de Formateo) es un lenguaje de marcas que permite especificar el aspecto que tendría una serie de datos (en ocasiones extraídos de un documento XML de partida) al mostrarlos en un determinado formato de salida, habitualmente un documento **PDF (Portable Document Format – Formato Portable de Documento),** pero también **RTF (Rich Text Format – Formato de Texto Enriquecido)**, PostScript y otros.

- **FOP. ¿Qué es y para que se utiliza?**

FOP (Formatting Objects Processor – Procesador de Objetos de Formateo) es una aplicación desarrollada en Java, dentro del proyecto de código abierto de Apache que, en su uso más frecuente, toma como entrada un documento XSL-FO y genera una salida en algún formato "visible" como PDF.

Se usará una sintaxis muy simplificada en función de la tarea que se quiere llevar a cabo:

- En un primer momento se usará para generar una salida en formato PDF a partir de un documento con formato FO. La sintaxis aplicar es:

  fop –fo <doc_existente.fo> -pdf <doc_generado.pdf>

- Más adelante, lo que se pretende es generar el mismo documento de salida en formato PDF, pero el documento de entrada en formato FO se construirá dinámicamente a partir de un documento XML y una hoja de transformaciones XSLT. La sintaxis a aplicar en esta caso es:

  fop –xml <doc_existente.xml> -xsl <doc_existente.xsl> -pdf <doc_generado.pdf>

  Tambien se prodria optar por generar la salida en dos pasos:

  1. Se genera el documento en formato FO a partir de un XML y una XSLT.

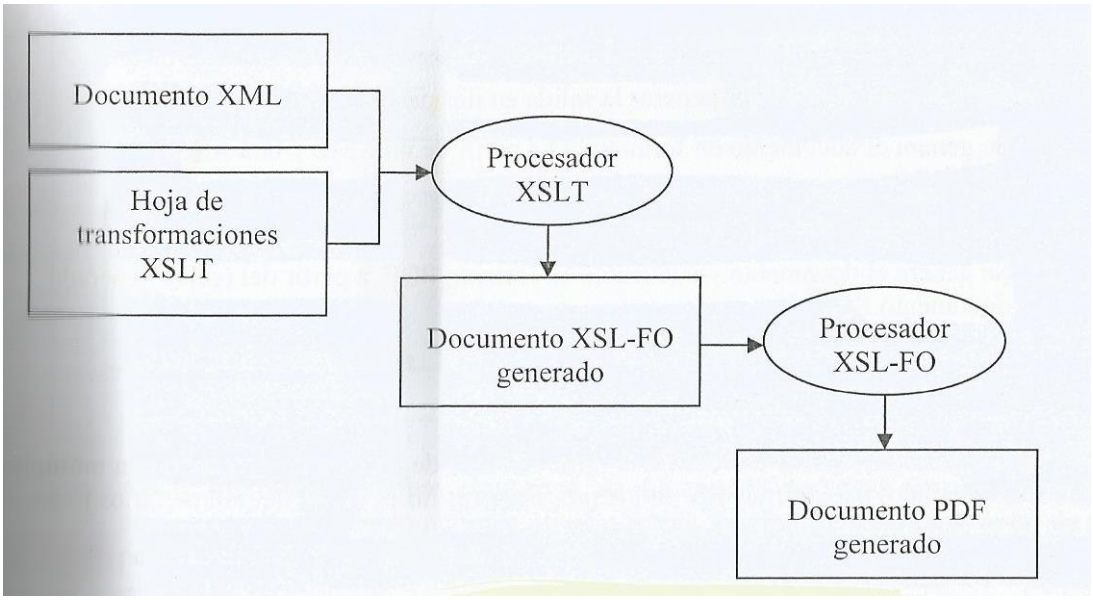
     fop –xml <doc_existente.xml> -xsl <doc_existente.xsl> -foout <doc_generado_paso1.fo>
  2. Se genera el documento visualizable en formato PFD a partir del recién generado documento FO.

     fop –fo <doc_generado_paso1.fo> -pdf <doc_generado.pdf>

NOTA: Las norma XSL-FO define un gran número de objetos de formateo con múltiples propiedades cada uno, pero la mayoría de los procesadores no son capaces de interpretarlos todos, o no son capaces de interpretar alguna de sus propiedades.

Alternativas a FOP son:
- RenderX, con licencia propietaria, aunque existe una edición Personal para uso no comercial que se obtiene de manera gratuita.
- Antenna House, con licencia propietaria.
- PassiveTex, con licencia de software libre.
- Altova VisionStyle, con licencia propietaria.

- **Página 349: Figura 6.5.**



Transformación XSLT y procesamiento XSLT-FO

Procesadores XSL-FO: son programas que reciben como entrada un documento XSL-FO y generan como salida un documento con posibles diversos formatos. Los mas habituales son PFD, RTF, PostScript, texto plano...

Un uso muy frecuente, que sirve como nexo con los conocimientos de XSLT recién adquiridos, es la transformación de un documento de partida XML, mediante una hoja de transformaciones XSLT, en un documento XSL-FO. A su vez, este documento podrá actuar como entrada de un procesador XSL-FO y generar como salida un documento en el formato de salida que elijamos entre los ya citados (pongamos PDF). Figura 6.5

- **TEMA 7: Sindicación de contenidos. RSS**
    - **Punto 7.1: RSS. ¿Qué es y para que se utiliza?**

RSS son las siglas de **Really Simple Sindication – Sindicación Realmente Simple**, y se trata de una tecnología que sirve para compartir o distribuir información en la Web. Cuando un usuario esta interesado en un determinado tema, quiere recibir información de forma continua y actualizada, para lo cual, se suscribe a la fuente de los contenidos de dicho tema. De esta forma, el usuario no tiene que preocuparse de consultar periódicamente la fuente para comprobar si hay información nueva.

Inicialmente era necesario utilizar un software específico diseñado para leer estos contenidos RSS, lo que se conoce como **feer reader, agregador de noticias o lector RSS**, pero actualmente los navegadores incorporan el software necesario para leer los RSS.

La información de un sitio Web puede ser compartida de varias formas, una manera sencilla es como hemos descrito antes, mediante suscripción a la fuente con un agregador de noticias. Pero también se puede compartir insertando la información en otros sitios Web, de esta forma el receptor de las noticias se convierte a su vez en emisor, a esto se le conoce como redifusión Web. Este nuevo servicio es uno de los pilares básicos de lo que se ha llamado Web 2.0, formada por ese conjunto de aplicaciones Web que mejoran la difusión de información como las redes sociales, wikis, blogs, etc.

Resumiendo, **RSS es una técnica para difundir de forma más eficaz los contenidos de un sitio Web. Pero también es un lenguaje derivado de XML, para construir los archivos que guardan el contenido a difundir.** Sin embargo, no es el único lenguaje existente con este propósito, posteriormente apareció Atom, también derivado de XML pero con algunas ventajas sobre su predecesor. Por tanto, tenemos dos lenguajes, es decir, dos formatos de archivo que sirven para almacenar los contenidos a distribuir. No debemos

confundir los dos significados de RSS, el primero hace referencia al concepto general de sindicación o redifusión Web y el segundo a un formato en particular de archivo que contiene la información a difundir.

Ventajas de la sindicación de contenidos:

- Los usuarios no necesitan comprobar si la información ha sido actualizada en los sitios donde se encuentran los contenidos de su interés.
- El formato de los datos es texto plano, por tanto, ligero y rápido a la hora de ser transmitido, esto lo hace idóneo para dispositivos móviles.
- Mediante la sindicación se puede filtrar la información que nos interesa de cada sitio y no perder tiempo con el resto de información.
- Protege la cuenta de correo puesto que no es necesario utilizarla, asi evitaremos correo no deseado.

    - **Punto 7.2: Estructura de un RSS.**

Como siempre empezamos con el prólogo:

```
<?xml versión=”1.0” encoding=”utf-8”?>
<?rss versión=”2.0”>
```

La primera línea es la declaración XML, que define la versión de XML y la codificación de caracteres utilizados. En este caso, el documento cumple con la especificación 1.0 de XML y utiliza el conjunto de caracteres Unicode.

La siguiente línea es la declaración de tipo de documento, que identifica el lenguaje derivado de XML que estamos usando, en este caso se trata de un documento RSS versión 2.0.

Si queremos aumentar el número de etiquetas disponibles debemos incluir espacios de nombres, algunas posibilidades son:

```
<rss versión=”2.0” xmlns:atom=”http://www.w3.org/2005/Atom”>
<rss versión=”2.0” xmlns:g=”http://base.google.com/ns/1.0”>
<rss versión=”2.0” xmlns:media=”http://search.yahoo.com/mrss/”>
```

También se trata de la raíz del documento, por ello, el contenido debe estar comprendido entre las etiquetas **<rss>** contenido **</rss>**.

Después del prólogo viene el contenido del documento.

La siguiente línea contiene el elemento **<channel>** que describe la fuente o canal RSS.

El elemento **<channel>** puede tener uno o más elementos **<ítem>** y cada elemento **<ítem>** define un artículo en el canal RSS.

Por último, las dos últimas líneas consisten en cerrar la etiqueta **<channel>** y la raíz **<rss>.**

Ejemplo:

```
<?xml versión=”1.0” encoding=”utf-8”?>
<rss versión=”2.0”>
   <channel>
       <title>Bienvenidos a mipagina</title>
       <link>http://www.juanmacr.es</link>
       <description>Ayuda al estudiante de informática</description>
       <ítem>
              <title>ASIR</title>
              <link>http://www.juanmacr.es/rss/asir.rss</link>
              <description>Ciclo de administración</description>
       </ítem>
   </channel>
</rss>
```

- **Punto 7.3: Elemento <channel>, Elemento <ítem>.**

<channel>

Este elemento describe la fuente RSS y tiene tres elementos obligatorios:

<title>: Define el nombre del canal.
<link>: Define el hipervínculo para el canal.
<description>: Describe el contenido del canal.

El elemento **<channel>** puede contener uno o más elemento **<ítem>**

Los elementos opcionales en **<channel>** son:

Elemento	Descripción
<category>	Define una o más categorías para el canal
<cloud>	Permite ser informado inmediatamente de los cambios en el canal
<copyright>	Notifica sobre el contenido con derechos de autor
<docs>	Indica una dirección para la documentación del formato utilizado
<generator>	Especifica el programa utilizado para generar el canal
<image>	Presenta una imagen cunado los agregadores muestren un canal
<language>	Especifica el idioma en que está escrito el canal
<lastBuildDate>	Define la fecha de la última modificación del contenido del canal
<link>	Define el hipervínculo para el canal
<pubDate>	Define la última fecha de publicación en el canal
<rating>	La valoración PICS del canal
<skipDays> **<skipHours>**	Especifica los días/horas durante los cuales los agregadores deben omitir la actualización del canal
<textInput>	Especifica un campo de entrada de texto que aparece con el canal
<ttl>	Especifica el tiempo en minutos, que el canal puede permanecer en la caché, antes de actualizarse desde la fuente (“time to live”
<webMaster>	Define la dirección e-mail del webmaster del canal

Vamos a explicar algunos de los más conocidos:

(.) El elemento **<category>**

Permite a los agregadores de RSS agrupar sitios basándose en la categoría. Se puede indicar jerarquía en las categorías usando barras.

Ejemplo: **<category>Noticias/economía/rescate</category>**

(.) El elemento **<copyright>**

Permite identificar el material con derechos de autor.

Ejemplo: **<copyright>Garceta-2012. Todos los derechos reservados</copyright>**

(.) El elemento **<image>**

Permite que se muestre una imagen cuando los agregadores presentan un canal.

El elemento **<image>** tiene tres elementos secundarios obligatorios:

<url>: Indica la URL de la imagen.
<title>: Indica el texto que se mostrará si la imagen no se pudo cargar.
<link>: Indica el hipervínculo a la página web que ofrece el canal.

Ejemplo:

```
<image>
        <url>http://www.mipagina.es/imagenes/logo.jpg</url>
        <title>miempresa</title>
        <link>http://www.mipagina.es</link>
</image>
```

(.) El elemento **<language>**

Permite especificar el lenguaje utilizado para escribir el documento RSS y agrupar los sitios basándose en el lenguaje.

Ejemplo: **<language>es-es</language>**

(.) El elemento **<generator>**

Es muy común cuando el canal es generado automáticamente por alguna herramienta.

Ejemplo: **<generator>Joomla! – Open Source Content Management</generator>**

<item>

Como se mencionó antes, cada element <item> define un artículo en el canal RSS.

El elemento <ítem> contiene tres elementos necesarios:

<title>: Define el título del artículo.
<link>: Define el hipervínculo al articulo.
<description>: Describe el artiuclo.

Los elementos opcionales de <ítem> son:

Elemento	Descripcion
<autor>	Especifica el email del autor del artículo.
<category>	Define la categoría/s a las que pertenece el artículo.
<comments>	Permite enlazar a los comentarios sobre ese tema.
<enclosure>	Permite incluir un archivo multimedia.
<guid>	Define un identificador único para el artículo
<pubDate>	Define la fecha de la última publicación para el artículo
<source>	Especifica una fuente para el articulo mediante un link

(.) El elemento <comments>

Permite un elemento para vincular a los comentarios sobre ese tema.

Ejemplo: **<comments>http://www.viva_pilar.com/comentarios</comments>**

(.) El elemento <enclousure>

Permite incluir un archivo multimedia en el artículo.

Tiene tres atributos obligatorios:

- **url:** Define la URI del archivo.
- **longitud:** Define el tamaño en bytes del archivo.
- **type:** Define el tipo de archivo multimedia.

Ejemplo:

<enclousure url=”http://www.mipagina/rss/noticias.mp3” length=”1200” type=”audio/mpeg”/>

- **Proviene de PAC. ¿Lectores RSS?**

Una vez escrito y validado nuestro archivo RSS hay que subirlo al servidor Web.

Ahora debemos hacer 3 cosas:

1. Insertar un enlace en la pagina de inicio que apunte al archivo RSS para poder consultarlo a través del navegador.
2. Insertar un elemento **<link>** para que los **lectores o agregadores RSS** puedan encontrar nuestro archivo RSS y leerlo, es decir, suscribirse a nuestro feed.

3. Enviar la dirección URI del archivo RSS a sitios Web, llamados **directorios RSS**, que se dedican a catalogar y almacenar feeds para que los buscadores los visiten.

Empezamos por incluir el enlace, para lo cual usamos el icono estándar de sindicación de contenidos:

Ahora viene lo mas importante, la suscripción a nuestro feed, para lo cual se necesita un lector RSS. Los lectores pueden ser programas específicos para agregar contenidos, por ejemplo, **Feedreader** o aplicaciones de un buscador, por ejemplo, **Feedly**, ya que Google Reader se ha discontinuado.

Lo primero es añadir un enlace en nuestro sitio para localizar el archivo RSS:

<link href="archivo.rss" rel="alternate" type="application/rss+xml" title="RSS 2.0"/>

Ahora abrimos nuestro lector RSS y agregamos la dirección del archivo RSS, en Feereader se hace mediante el botón Nuevo Canal.

En Feedly, es necesario crearse un cuenta, se hace con el botón FOLLOW.

Los lectores permiten crear carpetas para organizar las suscripciones por temas, ordenar por fechas o fuentes, también tienen otras opciones como ver los artículos en forma de lista o completos, etc.

Para consultar la fuente de donde proceden, basta con hacer clic sobre un artículo y nos redirecciona al sitio Web de origen.

Por último, podemos enviar la dirección de nuestro feed a directorios RSS, con el fin de mejorar los resultados de búsqueda.

También podemos enviar la dirección de nuestro sitio Web a buscadores.

Esto se conoce como **técnicas SEO, Search Engine Optimizacion**, que significa, posicionamiento en buscadores).

- **TEMA 8: Sistemas de gestión de información. ERP**

  - **Leer páginas 384, 385, 386, 387, 392, 393.**

Herramientas de **BI (Busniness Intelligence – Inteligencia del Negocio)** que permite registrar información sobre distintos aspectos de una determinada actividad. Uno de los exponentes más claros de estas aplicaciones de inteligencia de negocio son los **ERP, sistemas modulares integrados de gestión de empresas.**

Inteligencia del negocio (BI):

En el pasado, las empresas y organizaciones realizaban tareas de una manera artesanal, aprendiendo realmente poco del funcionamiento de sus propios procesos. Esto era debido a la dificultad de almacenar la información destacada de estos procesos y, sobre todo, a procesarla y recuperarla con rapidez.

Desde la aparición de los ordenadores personales, individuos y organizaciones han sido capaces de registrar datos en sistemas de información más o menos sofisticados. Desde simples archivos de texto hasta sofisticadísimos sistemas gestores de bases de datos, sistemas expertos, sistemas **CBR (Cased Based Reasoning – Razonamiento Basado en Casos)**, etc.

Estos pasos que se han ido dando en la evolución de los sistemas de información han permitido la automatización de muchas tareas y la extracción de conclusiones muy interesantes.

En el momento en el que se ha dispuesto de una gran cantidad de datos, fácilmente procesables y recuperables, se han podido estudiar los diferentes procesos de cualquier actividad, sus casos de éxito y sus casos de fracaso. Esto ha permitido

aprender en profundidad el funcionamiento de estos procesos, optimizarlos, transformarlos y, en ocasiones, eliminarlos. Se ha generado asi la llamado **inteligencia de negocio.**

Se entienden entonces por **aplicaciones de inteligencia de negocio aquellas que automatizan y agilizan procesos, registran resultados, favorecen la extracción de conclusiones, etc.**

Aunque inicialmente estas herramientas fueron desarrollándose por las propias empresas para uso interno, poco a poco se fue generalizando su uso y aparecieron compañías que desarrollaban aplicaciones de inteligencia de negocio genéricas, pero adaptables o parametrizables a las necesidades de cada compañía que las usara.

En cuanto a su ámbito de aplicación, ha habido herramientas muy generales, que cubrían muchos aspectos de un negocio y otras más específicas, centradas en alguna actividad más concreta.

Se pueden citar una serie de **familias de aplicaciones** con sus particularidades:

- **ERP (Enterprise Resource Planning – Planificación de Recursos Empresariales), son aplicaciones integrales** (pueden llegar a cubrir las necesidades de un negocio en casi todas las áreas) **y modulares** (se suelen distribuir por paquetes o módulos que interaccionan entre sí de manera transparente al usuario).
- **CRM (Customer Relatioships Management – Gestión de Relaciones con Clientes),** son aplicaciones que, aunque en ocasiones se integran en sistemas ERP, han ido teniendo un amplio rango de diferentes funcionalidades, lo que ha derivado en su desarrollo independiente de aquéllos.
- **CM (Change Management – Gestión del Cambio),** son herramientas cuyo fin es facilitar a los empleados de una compañía los cambios estructurales que puedan tener lugar. Recogen información sobre los propios procesos de cambio.
- **BA (Business Analytics – Analítica del Negocio),** son herramientas destinadas a analizar datos, resultados, tendencias y poder así sacar conclusiones sobre la evolución del negocio.
- **ETL (Extract, Transform and Load data – Extraer Transformar y cargar datos) o integración de datos,** son aplicaciones orientadas a la fusión y homogeneización de datos de orígenes variados.
- **SFA (Sales Force Automation system – sistemas de Automatización de Ventas),** son aplicaciones que suelen formar parte de un CRM y permiten registrar todas las fases del proceso de una venta.
- **BPM (Business Process Management – Gestion de Procesos de Negocio).**
- **MRP (Material Resource Planning – Planificación de los Recursos Materiales).**
- **SRM (Supplier Relationship Manager – Gestión de las Relaciones con proveedores).**
- **KM (Knowledge Management – Gestion del Conocimiento).**

ERP:

ERP es la denominación genérica de cualquier sistema de gestión de la información interna y externa de una organización, incluyendo finanzas, contabilidad, ventas/compras, relaciones con clientes, gestión de almacenes...

El objetivo de estos sistemas es facilitar el **flujo de información** entre los diferentes estamentos de la empresa, más aún en empresas muy grandes y compartimentadas, sin el cual se producirían duplicidades en los esfuerzos realizados.

Las ERP suelen ser muy **versátiles,** funcionando en distintos sistemas operativos con distintas configuraciones de hardware. Habitualmente se apoyan en un sistema gestor de base de datos para almacenar su información

Características de un ERP:

- **Integral:** cubre todas las necesidades de gestión de una empresa.
- **Modular:** las aplicaciones se organizan por módulos independientes pero combinables.

- **Parametrizable:** con capacidad de adaptarse a las necesidades concretas de cada empresa.
- **Escalable:** con capacidad de crecimiento.

Ejemplos de ERP:
- **SAP.**
- **Navision.**
- **PeopleSoft,** de Oracle.
- **Compiere.**
- **OpenBravo.**

SAP:

SAP (Systeme Anwendugen un Produkte – Sistemas, Aplicaciones y Productos) es el nombre de la empresa alemana que desarrolla el ERP comercial del mismo nombre más usado en la actualidad. La empresa SAP es el desarrollador de software mas potente de Europa y de los mayores del mundo.

El nombre actual del ERP es **ECC (Enterprise Core Component – Componente Central de la Empresa)** y su versión la 6.0. Anteriormente recibió otros nombres como R/1, después R/2 y el mas conocido R/3 (sistema en tiempo real de 3 capas).

Esta compuesto de multitud de módulos que cubren la mayoría de los procesos de negocio que puede necesitar cualquier empresa. Los **módulos más destacados** (los cuales tienen a su vez múltiples sub-módulos) son:
- **FI (Financial) Finanzas.**
- **CO (Controlling) Costos y control.**
- **LO (Logístics) Logística.**
- **SD (Sales and Distribution) Ventas y distribución.**
- **MM (Materials Management) Gestión de materiales.**
- **LE (Logistics Execution) Ejecución logística.**
- **PP (Production Planning) Planificación de la producción.**
- **HR (Human Resources) Recursos Humanos.**
- **BC (Basics Components) Componentes básicos.**
- **IS (Industrial Solutions) Soluciones industriales.**
- **CRM (Customer Resource Management) Gestión de clientes.**
- **QM (Quality Management) Gestión de calidad.**

Los ERPs se pueden adaptar a las necesidades específicas de una empresa. Esto se puede conseguir mediante la codificación de programas que realicen determinadas tareas necesarias para la empresa. SAP integra un lenguaje de programación de cuarta generación propio, **ABAP (Advanced Business Application Programming – Programación de Aplicaciones Avanzadas de Negocio)**, que apareció con la versión R/2. Permite trabajar con distintos tipos de datos, acceder a diversas bases de datos, tratar archivos. Permite, asi mismo, realizar llamadas a procesos remotos para comunicarse con otros sistemas.

La última versión de SAP permite también codificar módulos en Java.

OpenBravo:

Es una alternativa a SAP en software libre. Se ejecuta a través de un cliente web y dispone de una interfaz **RIA (Rich Internet Application – Aplicación Rica en Internet)** de fácil manejo.

Dispone igualmente de una serie de **módulos** que se integran entre si:

- **Gestión de datos maestros.**
- **Gestión de aprovisionamientos.**
- **Gestión de almacenes.**
- **Gestión de proyectos y servicios.**
- **Gestión de la producción.**
- **Gestion comercial y CRM.**
- **Finanzas y contabilidad**

- **Inteligencia de negocio.**
- **Retail POS – (Point Of Sale – Punto de venta).**

Actualmente se encuentra en la **versión 3 (OB3).** Se autodenomina el ERP **ágil,** porque se supone que su aprendizaje y posterior manejo son rápidos y fáciles.

CRM: (Customer Relationship Management – Gestión de Relaciones con Clientes)

Es una aplicación informática, en ocasiones integrada en un ERP, orientada al registro de información de clientes, ventas y marketing. Su objetivo fundamental es ganar y mantener clientes.

Un ejemplo de CRM podría ser la aplicación que se usa en un concesionario de automóviles donde se haga un seguimiento de clientes, ofertas, campañas publicitarias...

Existen una serie de **tareas** que suelen cubrir los CRMs:
- **Almacenando de datos de clientes.**
- **Ofertas comerciales.**
- **Informes.**
- **Campañas publicitarias y de marketing.**

Alguno de los **CRM existentes** actualmente en el mercado son:
- **Salesforce,** de Salesforce.com Inc, con licencia propietaria.
- **CiviCRM,** alternativa de software libre.
- **HiperGate,** software libre.
- **SugarCRM,** es otra alternativa de software libre.

SugarCRM:

Es un CRM de software libre. Actualmente se encuentra en su versión 6.5.

Al igual que los ERPs, dispone de una serie de **modulos. Los más importantes son:**
- **Ventas.**
- **Marketing.**
- **Servicio al cliente.**

Las **características de la arquitectura del servidor** son:
- Desarrollado inicialmente para LAMP (Linux, Apache, MySQL y PHP).
- Toda la lógica de negocio está implementada en PHP, por lo que no usa procedimientos almacenados ni triggers de la base de datos.
- Actualmente tiene soporte para diferentes SOs: Windows, Unix, Mac OS.
- Los sistemas gestores de bases de datos para los que hay versiones son MS SQL Server y Oracle.
- Modelo – Vista – Controlador.

La aplicación puede ejecutarse:
- En la nube: de la compañía, pública o en la de un socio.
- En local.
- En teléfonos inteligentes y tabletas.
- Flexible e integrable con otras aplicaciones.

Las **funciones principales** son:
- Mantenimiento de cuentas de empresas.
- Mantenimiento de contactos de personas.
- Mantenimiento de oportunidades de negocio.
- Creación de informes.
    - Tabular.
    - Sumarizado.
    - Sumarizado detallado.
    - Matricial.
- Mantenimiento de documentos.
- Mantenimiento de casos.
- Registro de llamadas.

- Enviar correo.
- Planificar reunión.
- Mantenimiento de tareas.
- Mantenimiento de notas o adjuntos.
- Crear caso.
- Crear campaña publicitaria:
    - Via correo electrónico.
    - Via correo ordinorio.
- Creación de artículos para la base de conocimiento.
- Mantenimiento de empleados.

    - **Leer ampliación-complemento aportado al final de este documento.**
    - **Entra todo lo mencionado inmediatamente anterior a este punto para el tema 8.**

AMPLIACIÓN-COMPLEMENTO UF3

ERP

Los ERP forman parte de las herramientas BI (Bussines Intelligence) que registran la información sobre aspectos de una determinada actividad. Un componente estrella de los ERP son los CRM que se dedican a la gestión de contactos con clientes.

Un ERP (en inglés, Entreprise Resource Planning) es un sistema para incrementar la eficacia de una empresa, mejorando su sistema de ventas, distribución, logística... en resumidas cuentas un ERP es un sistema de gestión global para la empresa y en el que se trabaja por separado (módulos) cada área. Los ERPs adquieren estructuras modulares y destacan por sus Menús modulares con base de datos central y por su posibilidad de implantar el sistema mediante etapas o partes. Son aplicaciones integrales.

Los módulos de un ERP común són: producción, distribución, logística, envíos, inventario, módulos de compras y ventas, facturación y contabilidad en la empresa. Pero los ERP también pueden intervenir en áreas como la gestión de proyectos o control de pedidos, recursos humanos, calidad de servicio, control de almacenes.

Esquematizando las características de un ERP son:

- Acceso a una base de datos centralizada.
- Captura de datos automáticamente.
- Estructura de trabajo modular.
- Configurables. Cada parte se puede especificar con una configuración propia.
- Interactivo. Los componentes destacan por su interacción interna.

HISTORIA DE LAS SIGLAS ERP

Nos remontamos entre los años 1939 y 1945, justamente cuando tuvo lugar la Segunda Guerra Mundial. Como suele suceder, el armamento y la estrategia suelen ir de la mano, y más si se trata de uno de los conflictos bélicos más importantes del siglo XX. El gobierno de los Estados Unidos ideó programas especializados para conseguir el control de la logística militar y organizar a las tropas en el frente conocidos como sistemas de **Planificación de Requerimiento de Materiales** (Material Requeriments Planning Systems o MRP-I Systems).

Los sistemas MRP empezaron a aplicarse a la gestión de grandes empresas para el control de sus sistemas productivos a finales de los años 50. Los **MRP** fueron evolucionando también con los cambios tecnológicos e informáticos y creándose programas avanzados de gestión especializados que integraban todas las funciones que se desarrollan en una empresa y que la empresa de tecnología informática Gartner bautizaría como ERP.

Actualmente los sistemas ERP son los sistemas de negocio principales de una empresa, complementándose con otros sistemas de información auxiliares como **Customer RelationShip Management – CRM, Enterprise Content Management – ECM, Businnes Process Management – BPM o Business Intelligence –BI.**

ESTRUCTURA BÁSICA DE UN ERP

Existen muchas aproximaciones distintas para describir cómo se compone un sistema ERP. Una de las más comunes, consiste en analizar el ERP siguiendo las diferentes capas que lo componen:

- La **infraestructura hardware** viene determinada por los requisitos mínimos del fabricante del ERP y por las funcionalidades y utilidad que dará la empresa al programa. Es una de las estructuras que se deben planificar con más cuidado, ya que depende mucho del funcionamiento de la aplicación.

- La **infraestructura software** es absolutamente dependiente de la capa anterior. Una vez se han implantado las máquinas y comunicaciones, se tiene que estudiar qué software se escogerá.

- La **base de datos** variará dependiendo del volumen que se desee almacenar. Suele ser común que en el momento de seleccionar el software que se desea integrar, puedas escoger la base de datos que mejor se ajuste a las necesidades.

- La **aplicación** es el corazón del sistema, ya que refleja los procesos internos de la empresa. El fabricante adapta las necesidades y funcionalidades que la empresa necesita en el momento de implantar la aplicación. Integra los módulos de la empresa desde donde después se gestionará la información. Entre ellos encontramos: Gestión financiera, ventas y compras, fabricación o recursos humanos entre otros.
- La **interfaz** es la encargada de conectar los componentes anteriores y permitir así a los usuarios trabajar con la aplicación.

INTERFAZ DE ERP

1. **Cliente Estándar.** Consiste en una aplicación con toda la funcionalidad disponible que puede estar ejecutando las reglas de negocio o tareas en las áreas funcionales citadas anteriormente, en el mismo ordenador en el que se está ejecutando la propia aplicación Cliente Estándar. Ello implica una mayor necesidad de recursos tanto de hardware, ya que no cualquier ordenador servirá, como de comunicaciones porque el ancho de banda disponible deberá ser grande.

2. **Cliente Ligero.** Es una aplicación especialmente diseñada para que el uso de recursos de hardware y comunicaciones se minimice. La tendencia general es que esta aplicación especial sea el navegador web. Con ello se obtiene varios beneficios:
    - No hay que instalar nada adicional
    - Consumo reducido de ancho de banda
    - Posibilidad de usar dispositivos móviles.
    - etc

    A todo esto se puede sumar que las posibilidades de estos clientes ligeros están ya muy cera de los clientes estándar.

3. **Aplicaciones de hoja de cálculo.** Si bien no se suelen considerar como parte del ERP, la experiencia demuestra que gran número de los usuarios de estos sistemas utilizan las hojas de cálculo especialmente para presentar informes complejos, gráficos, análisis de datos, etc.

4. **Clientes 100% Web.** Se ejecutan desde el navegador.