ILERNA Online

Módulo 4

Lenguajes de marcas y sistemas de gestión de la información

UF1. PROGRAMACIÓN EN XML 4

1. Reconocimiento de las características de los lenguajes de marcas 4
1.1. Clasificación 6
1.2. Herramientas de edición 8
1.3. XML: Estructura, sintaxis y etiquetas. 9
1.4. Elaboración de documentos XML bien formados 13
1.5. Modelo de datos de un documento XML. Nodos. 14

2. Utilización de lenguajes de marcas en entornos web: 17
2.1. Evolución Histórica 17
2.2. Identificación de etiquetas y atributos de HTML 20
2.3. Estructura Básica 23
2.4. Elementos de un HTML 25
2.5. Versiones de HTML y XHTML 46
2.6. Herramientas de diseño web 47
2.7. Hojas de estilo 49

3. Definición de esquemas y vocabularios de XML: 75
3.1. Validación. Documentos XML válidos 75
3.2. Lenguajes basados en XML 94

UF2. ÁMBITOS DE APLICACIÓN DE XML 95

1. Aplicación de los lenguajes de marcas a la sindicación de contenidos 95
1.1. Ámbitos de aplicación 96
1.2. Estructura de los canales de contenidos 96
1.3. Elementos principales de un RSS 98
1.4. Tecnologías de creación de canales de contenidos 100
1.5. Validación y publicación 101
1.6. Directorios de canales de contenidos 102
1.7. Agregación 103

2. Conversión y adaptación de documentos XML 104
2.1. Técnicas de transformación de documentos XML 105
2.2. Descripción y utilización de estructura, sintaxis y plantillas 107
2.3. Elaboración de documentación. 112
2.4. XSL- FO 114

3. Almacenamiento de información: 123
3.1. Sistemas de almacenamiento de información XML nativo. 123
3.2. XPath 124
3.3. Lenguajes de consulta y manipulación 132
3.4. Otras tecnologías complementarias 134
3.5. Bases de datos relacionales con XML 135
3.6. Tratamiento de XML desde JAVA 136

UF3. SISTEMAS DE GESTIÓN DE INFORMACIÓN EMPRESARIAL 137

1. Sistema de gestión empresarial: 137
1.1. Inteligencia del negocio 137
1.2. ERP 139
1.3. CRM 141
1.4. Instalación, configuración e integración de módulos 144

BIBLIOGRAFÍA **145**

WEBGRAFÍA **145**

UF1. Programación en XML

1. Reconocimiento de las características de los lenguajes de marcas

A continuación, vamos a definir las principales características de los lenguajes de marcas:

- **Texto plano**

Los archivos de texto plano se refieren a los que están formados solo por caracteres de texto, no pueden contener imágenes, sonido, archivos comprimidos ni otros tipos.

Los caracteres se codifican mediante la utilización de diferentes códigos dependiendo del idioma o del alfabeto necesario, como pueden ser: ASCII, UTF-8, ISO- 8859-15.

Entre sus principales ventajas se encuentra que los archivos de texto plano se pueden interpretar mediante un simple editor de texto. Sin embargo, los archivos binarios necesitan un software específico, por ejemplo, descompresores, visores multimedia, compiladores.

De esta forma, conseguimos que los documentos lleguen a ser independientes del sistema operativo o del programa que los ha creado.

- **Compacidad**

Permite mezclar las instrucciones de marcado con el propio contenido, por ejemplo, <h2>Contenido</h2>.

El código que aparece entre corchetes es lo que denominamos instrucciones de marcado o etiquetas. Haciendo referencia al ejemplo, podemos apreciar que la etiqueta es una etiqueta de presentación que nos indica que el texto seleccionado se corresponde con el formato asignado a la cabecera nº 2.

El texto situado entre las marcas hace referencia al contenido propio del documento.

- **Independencia del dispositivo final**

Son las distintas interpretaciones que se pueden hacer de un mismo documento. Dependiendo del dispositivo final, podemos tener distintos resultados si utilizamos un dispositivo móvil o si utilizamos un ordenador.

- **Especialización**

Con la especialización nos referimos a que, con el paso de los años, han ido evolucionando, de tal manera que, podemos hacer uso de los lenguajes de marcas en un gran número de áreas, como pueden ser los gráficos vectoriales, notación científica, interfaces de usuario, etc.

- **Flexibilidad**

Podemos combinar el mismo tipo de archivo con otros lenguajes diferentes, como pueden ser HTML con PHP y con JavaScript. Además, existen unas etiquetas que ya viene especificadas para tal fin, como son los <script>.

1.1. Clasificación

Dependiendo del **tipo de marcas** que utilicemos, podemos dividir los lenguajes de marcas en tres tipos diferentes que detallamos:

- **De presentación:** se refieren al formato del texto que, sin modificar su estructura, permite aumentar el tamaño de la fuente, añadir o eliminar la cursiva. Incorpora diferentes lenguajes de procedimiento que permiten combinar varias marcas de presentación para una macro determinada.
  El software encargado de representar el documento debe ser capaz de poder traducir el código, pero sin transformar el orden en que aparece.
  Las aplicaciones de edición y los procesadores de texto son algunos de los que utilizan este marcado. Algunos ejemplos pueden ser: TeX, Docbook (derivados de SGML), nroff, troff, RTF.
- **Descriptivo, estructural o semántico:** hacen referencia a las diferentes partes en las que se puede dividir un documento sin ofrecer detalles de cómo se representan y en qué orden van a hacerlo.
  XML es un lenguaje diseñado específicamente para generar marcado descriptivo junto con los lenguajes que deriven de XML y tengan este propósito.
  Van a ir creando una serie de documentos que se van a almacenar en forma de árbol por lo que son bases de datos, pero en este caso, no se utilizan tablas, ni bases de datos estructuradas, por tanto, reciben el nombre de bases de datos semiestructuradas. Algunos ejemplos de este tipo pueden ser: ASN. 1, YAML, EBML, RDF, XFML, OWL, XTM.
- **Híbrido:** se refieren a una un conjunto de lenguajes que permiten utilizar los dos tipos de marcas anteriores, como HTML, XHTML y WML.

También podemos clasificar los lenguajes de marcas según su **funcionalidad** en los siguientes tipos:

- **Para crear documentación electrónica:**

  RTF, TeX, troff, nroff

  Wikitexto, DocBook, LinuxDoc

  ASN.1, EBML, YAML

- **Tecnologías de Internet:**

  HTML, XHTML, WML (páginas web)

XMPP (para mensajería instantánea)

WSDL, SOAP, UDDI (servicios web)

RSS, Atom (sindicación de contenidos)

GladeXML, XForms, XAML (interfaces/formularios de usuario)

- **De propósito específico:**

  MathML, CML (fórmulas matemáticas)

  MusicXML (lenguaje musical)

  SSML, SRGS, VoiceXML (síntesis de voz)

  SVG, VML, X3D (gráficos vectoriales)

  XLL (enlaces):

  - XLINK (asociación de recursos)
  - XML Base (URI básico)
  - XPOINTER (localiza recursos)

  XSLT (transformación de documentos)

  XTM (mapas conceptuales)

  RDF, XFML, OWL, XMP (catalogación y clasificación de documentos)

  GML (información geográfica)

  OFX (intercambio de información financiera)

  ebXML (comercio electrónico)

  XML Dsig, XML Enc, SAML, XACML, XKMS, XrML (seguridad)

  XInclude (inclusión de archivos)

1.2. Herramientas de edición

Existen bastantes tipos de herramientas que podemos utilizar en un documento XML, entre las que diferenciamos las siguientes:

- **XMLSpy**, de Altova:
  http://www.altova.com/es
  Podemos definirlo como un editor de XML junto con XLST, XPath, XQuery, etc. Cuenta con un gran número de herramientas diferentes, entre las que podemos diferenciar las de representación gráfica de diferentes documentos. Se trata de un programa de pago, pero que dispone de una versión de prueba de 30 días.

- **<oXygen/>**, de Syncro Soft:
  http://www.oxygenxml.com
  Utilizado para una serie de tecnologías entre las que destacamos: XSD, XSLT o XQuery. Es necesario descargarse el *XML Editor.*

- **XML Copy Editor:**
  http://www.xml-copy-editor.sourceforge.net
  En este caso, nos referimos a una herramienta de software libre con licencia GNU GPL. También es editor de XML, XSD, XSLT, etc.

- **XMLPad Pro Edition**, de WMHelp:
  http://www.wmhelp.com/download.htm
  Herramienta bastante sencilla de utilizar cuya función principal es comprobar si un documento XML está bien formado.

- **Otras aplicaciones:**
  Existen otra serie de aplicaciones, aunque detallaremos solo algunas:

  Exchanger XML Editor:
  http://www.exchangerxml.com
  Liquid XML Editor:
  http://www.exchangerxml.com/editor/downloads.html

Se trata de versión de pago, de la que también existe una versión de prueba.

- **Parser XML:**

Procesador que se encarga de leer un documento XML para, posteriormente, determinar la estructura y propiedades de aquellos datos que se encuentran contenidos en dicho documento.

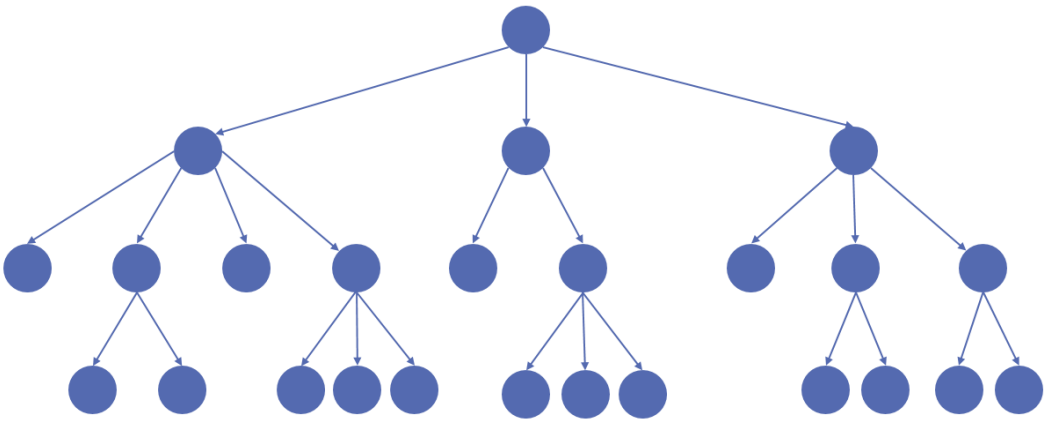
Este procesador o analizador va a comprobar que se han seguido las reglas de forma correcta y, a continuación, puede validar el documento e informar, en caso de que existan, de los errores producidos.

1.3. XML: Estructura, sintaxis y etiquetas.

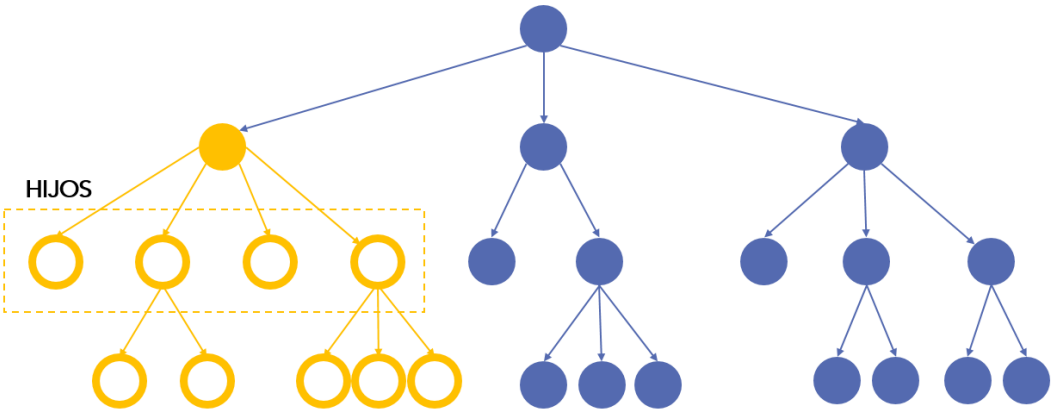
La información de un documento XML se organiza de forma jerárquica, de tal forma que los diferentes elementos del documento se van a relacionar entre sí mediante relaciones de padres, hijos, hermanos, ascendentes, etc.

Las diferentes relaciones que se pueden dar entre los diferentes nodos son:

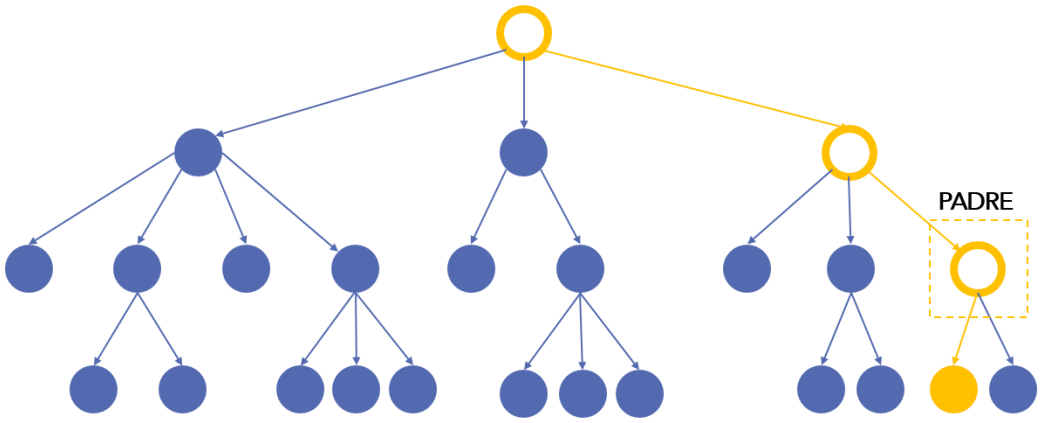
Original



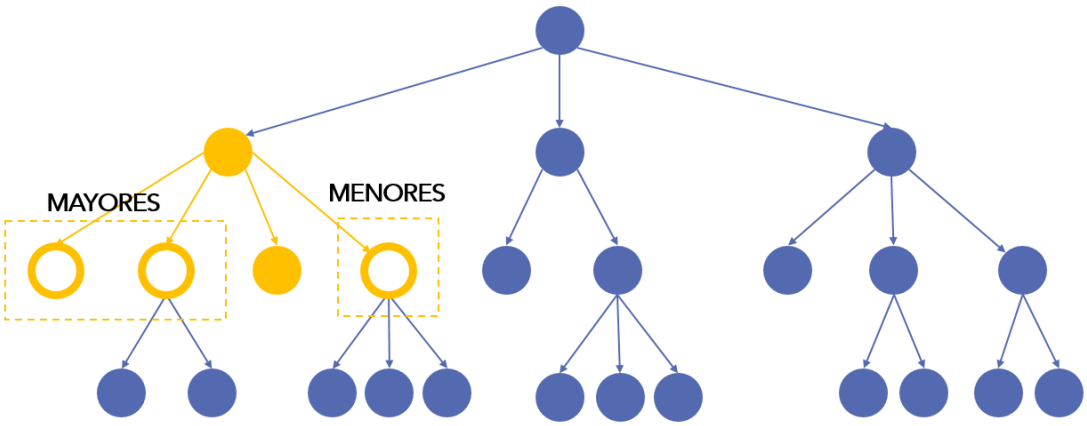
Descendentes



Ascendentes



Hermanos

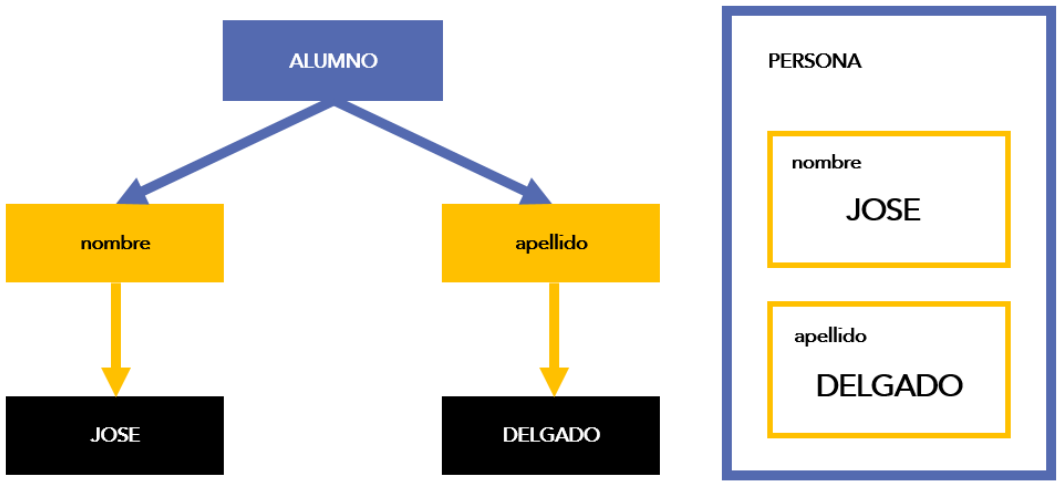


- A las partes del árbol que tienen hijos se les denomina nodos intermedios o ramas.
- Y a las partes que **NO** tienen hijos, las denominamos nodos finales u hojas.

Como ejemplo, veamos un elemento <alumno> que es "padre" de los elementos <nombre> y <apellido> que son "hermanos" entre sí.

```
<alumno>
    <nombre>Jose</nombre>
    <apellido>Delgado</apellido>
</alumno>
```

Si lo representamos, nos queda de la siguiente forma:



1.4. Elaboración de documentos XML bien formados

A la hora de especificar la sintaxis del lenguaje XML, definimos:

- Cómo delimitar los elementos con etiquetas.
- Qué formato podemos asignar a una etiqueta.
- Qué nombres son aceptables para los diferentes elementos.
- Dónde se pueden colocar los atributos.

Un documento XML **bien formado** sí cumple una serie de reglas establecidas por W3C en las especificaciones correspondientes para XML.

Estas reglas son bastantes y muy significativas. En algunos casos también son bastante complejas de entender, por lo que vamos a nombrar aquellas más significativas ya que son las que más se van a utilizar.

- Podemos comenzar un documento por una instrucción de procesamiento XML que indique cuál es la versión del XML que se está utilizando y, algunas veces, indica también la codificación de caracteres (*encoding*), que por defecto es UTF-8 y si ya se encuentra preparado para procesarse o si necesita una serie de archivos.
  Instrucción simplificada *<? xml version="1.0" ?>*
- Solo debe haber un elemento raíz del que van a colgar los demás elementos.
- Los elementos se encuentran entre los símbolos de apertura < > y cierre </>.
- Existes diferencias entre mayúsculas y minúsculas.
- Dos atributos no pueden tener el mismo nombre.
- Los atributos pueden llevar el valor "simples" o "dobles".

1.5. Modelo de datos de un documento XML. Nodos.

Un documento XML tiene una estructura formada por:

- **Raíz.** El nodo raíz es el primer elemento que encontramos en la estructura y lo designamos como "/".
- **Elementos.** Unidad básica de los documentos XML. Tienen como función principal almacenar información, y los etiquetamos de la misma forma que los elementos de HTML, por ejemplo: <ejemplo>...</ejemplo>. Podemos diferenciar dos tipos de elementos especiales:
    - **Elemento raíz.** Debe formar parte de cualquier documento XML. Si el documento está bien formado, debe tener solo un elemento raíz
    - **Elementos sin contenido.** Tenga atributos o no, debe abrirse y cerrarse con una sola instrucción.
      <ejemplo />
- **Atributos.** Permiten asignar un valor a un atributo haciendo uso del signo igual. Se van a tratar como tipo texto, de tal forma que el valor debe aparecer entre comillas.
  <persona DNI="35427641M">...</persona>
- **Texto.** Puede aparecer de dos formas diferentes, como contenido de un elemento determinado o como valor correspondiente de un atributo.
- **Comentarios.** Se utilizan de la misma forma que en HTML.
- **Espacio de nombres.** Utilizado para diferenciar las distintas etiquetas cuando se están utilizando varios vocabularios.
- **Instrucciones de procesamiento.** Comienzan por <? Y finalizan con ¿>. Engloban un conjunto de instrucciones que se envían al procesador.
- **Entidades predefinidas.** Utilizadas para poder representar caracteres especiales de marcado, aunque se interpreten por el procesador como si fuera un texto.

Entidad	Carácter
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

- **Secciones CDATA.** Grupo de varios caracteres que no necesitan ser analizados por el procesador.
- **Definición Tipo de Documento (DTD).** Se definen una serie de reglas que asignan una serie de restricciones sobre la estructura del documento XML.

1.5.1. Utilización de espacios de nombres en XML

- **Nombres XML**

Los nombres en XML deben seguir una serie de reglas para conseguir utilizar nombres correctos. Estas reglas son las mismas para nombres que para los atributos.

Los nombres en XML se denominan de la siguiente forma:

- Tienen permitido empezar con una letra (puede tener tilde o no), que puede pertenecer al alfabeto no latino, subrayado o con dos puntos.
- Los caracteres que aparezcan a continuación pueden ser: letras, dígitos, guiones bajos, subrayados, comas o dos puntos.
- Aquellos nombres que empiecen por las letras XML (mayúscula o minúscula) están reservadas.
- No deben contener espacios en blanco.

- **Uso de elementos frente a uso de atributos:**
  - **Elementos:**
    - Se pueden utilizar para representaciones jerárquicas.
    - Pueden tener atributos.
    - Puede que un elemento tenga distintas ocurrencias.
    - Aparecen en orden representativo.
    - Tienen posibilidad de poder extender con algunos elementos en su interior.
  - **Atributos:**
    - Deben asociar a los diferentes elementos.
    - Permiten alterar la información.
    - Aparecen en orden no representativo.
    - Permiten realizar el registro de metadatos.
    - No tienen posibilidad de poder extender con otros elementos en su interior.
    - Un atributo no dispone de múltiples ocurrencias.

- **Espacios de nombres**

Los espacios de nombres son una serie de mecanismos utilizados para que no se desarrollen conflictos sobre los nombres, de tal forma que, se puedan diferenciar los elementos que están dentro de un mismo documento XML y tengan el mismo nombre, pero pertenezcan a diferentes distribuciones.

Declaración:

<nombre_elemento ***xmlns:prefijo="URI del espacio_de_nombres"****>*

El URI correspondiente al espacio de nombres tiene que ser único, aunque no exista ninguna conexión que lo compruebe.

El URI es un nombre lógico del espacio de nombres.

Definimos el **espacio de nombres** como los diferentes lugares que pueden hacer referencia a un espacio de nombres en concreto. Todos los elementos llevan antepuesto el prefijo del espacio de nombres.

Además, también podemos declarar un espacio de nombres distinto para un elemento que descienda de otro, que ya haya declarado otro espacio de nombres.

- **Espacio de nombres por defecto**

En estos espacios de nombres no tenemos la obligación de definir un prefijo. Su ámbito de aplicación correspondiente es el del elemento en el que se ha declarado junto con sus elementos descendientes. No a sus atributos.

Para añadir un espacio de nombres a un documento XML que ya está creado, iremos insertando el prefijo a los elementos y atributos. Este proceso resulta un poco pesado y puede provocar una serie de errores. Así podemos evitar escribir prefijos.

Uno de los mayores inconvenientes que se puede dar en el espacio de nombres por defecto es que el espacio de nombre al que nos referimos solo afecte al elemento que está declarado y a sus descendientes. No a sus atributos.

- **Atributos especiales**
  - **xml: space**. Utilizado para indicar a la aplicación XML si debe tener en cuenta los espacios en blanco.
  - **xml: Lang**. Ofrece la posibilidad de indicar el idioma en el que está escrito, diferenciando entre:
    - Código que contiene solo dos letras
    - Identificador para el idioma.
    - Identificador que define el usuario
  - **xml: base**. Puede definir una URI diferente a la existente en el documento.

2. Utilización de lenguajes de marcas en entornos web

2.1. Evolución histórica

Cuando hablamos del origen de HTML nos remontamos al año 1980, que fue cuando el físico Tim Berners-Lee, que trabajaba para CERN (Organización Europea para la Investigación Nuclear), propone un nuevo sistema de "hipertexto" para que de pudieran compartir diferentes documentos.

Estos sistemas de "hipertexto" ya se habían desarrollado con anterioridad, aunque en el ámbito de la informática; cuando se habla de "hipertexto" se refiere a que los usuarios puedan acceder a la información que esté relacionada con aquellos documentos electrónicos que están visibles. Así, los "hipertextos" iniciales, se asimilaban a los enlaces a las distintas páginas web.

Años posteriores, Tim Bereners-Lee se une con el ingeniero de sistemas Robert Cailliau, con el que gana la propuesta WorldWideWeb (W3):

- En 1991 se presenta el primer documento con descripción de HTML y bajo el nombre *HTML Tags* (Etiquetas HTML).

- En 1993 se presenta la primera propuesta oficial para convertir HTML en un estándar. Aunque existieron avances muy significativos, porque se definieron las etiquetas de imágenes, las tablas y los formularios, no se llegó a conseguir ser el estándar oficial.

- Ya en 1995, es el organismo IETF quien se encarga de poner en marcha un grupo para trabajar con HTML, y es cuando se consigue publicar, el 22 de septiembre de 1995, el estándar HTML 2.0, siendo este el primer estándar oficial de HTML.

- A partir del año 1996, los diferentes estándares de HTML los publica otro organismo distinto denominado W3C (World Wide Web Consortium), llegando a publicar la versión HTML 3.2 el 14 de enero de 1997. Esta fue la primera recomendación de HTML que publicó W3C.

- Con la versión HTML 4.0, publicada el 24 de abril de 1998, se consiguieron numerosos avances sobre las versiones anteriores, y aparecía la posibilidad de añadir pequeños programas (scripts) en las páginas web con lo que se conseguía mejorar la accesibilidad de las páginas que ya estaban diseñadas,

trabajar mediante la utilización de tablas más complejas y mejora de los formularios.

- La publicación de HTML 4.01 (publicada en 1999), basada, sobre todo, en revisar publicaciones anteriores, pero no añade avances significativos. Detuvo un poco el desarrollo de HTML para centrarse más en el estándar XHTML.

- Sobre el año 2004 y debido a este parón que existió, algunas empresas como Appel, Mozilla y Opera empezaron a mostrar su preocupación por la falta de interés que estaba existiendo del W3C DE HTML, y fue entonces cuando empezaron a organizar una nueva asociación denominada WHATWG (*Web Hypertext Application Technology Working Group*).

- El 22 de enero de 2008 se publica el primer borrador oficial del estándar HTML5.

- En marzo del 2007 se publican distintos borradores de HTML5.0.

- De forma paralela, se seguía avanzando sobre la estandarización de XHTML (versión avanzada de HTML basada en XML), publicando su primera versión en enero del 2000.

- XHTML 1.0 está basado en la adaptación de HTML 4.01 al lenguaje XML por lo que utiliza sus mismas etiquetas y muchas de sus características, aunque añade algunas nuevas.

- La versión XHTML 1.1 y XHTML 2.0 se publicaron en forma de borrador con la intención de poder modularizar XHTML.

AÑO	EVENTO
1991	Nacimiento. "HTML tags"
1993	HTML 1.2
1995	HTML 2.0
1994	Se funda W3C
1997	HTML 3.2
1998	HTML 4.0 y XML
1999	HTNK 4.01
2000	XHTML 1.0
2001	XHTML 1.1
2002	XHTML 2.0
2003	XFORMS
2004	Se funda WHATWG
2008	Borrador de HTML 5

Una vez conocido el origen y la evolución a lo largo de los años de este lenguaje de marcas, nos pararemos un poco en conocer más a fondo este lenguaje.

2.2. Identificación de etiquetas y atributos de HTML

HTML es un lenguaje de marcas que nos permite desarrollar diferentes páginas web. Para ello, necesitamos:

- Un editor de textos ASCII mediante el que vamos a poder añadir el contenido que pretendemos mostrar.
- Un navegador web con el que podemos visualizar el contenido de la página.

Todos aquellos ficheros que contengan documentos HTML van a tener como extensión .HTML o .htm.

Otro editor HTML (aunque un poco más específico), puede ser WYSIWYG (***W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et*), que se traduce como "Lo que ves, es lo que obtienes". De esta forma, podemos escribir diferentes documentos HTML y ver, de forma simultánea, cómo quedaría el resultado final de la página web, de la misma forma que se vería cuando la publicásemos en Internet.

También existen otras herramientas de edición, que podemos utilizar para llevar a cabo la realización de diferentes páginas web de forma más profesional, como pueden ser, entre otras, Aptana, Amaya, Dreamweaver y Kompozer. Esta serie de editores utilizan diferentes menús e iconos en los que podemos añadir:

- Algunas etiquetas (directivas) de HTML sin que las tengamos que teclear.
- Reglas estilo CSS.
- Diferentes funciones destinadas a la creación y mantenimiento de la página web.

Cuando tengamos la página web lista para su posterior publicación en Internet, vamos a necesitar un servidor de páginas web en los que podamos almacenar las distintas páginas. El servidor web es un software que se encuentra en el propio ordenador y debe estar conectado siempre a Internet. Cuando pongamos las páginas en el servidor, ya serán accesibles a todos los usuarios que pertenezcan a la misma red.

Existen proveedores de servicios de Internet que ofrecen a sus clientes sitios webs gratuitos para que pueden publicar sus páginas web personales o corporativas y, de esta forma, evitamos instalar un servidor web propio.

2.2.1. Etiquetas y atributos

Antes de comenzar a utilizar cualquier lenguaje de marcas es conveniente familiarizarnos con una serie de normas básicas que debemos tener en cuenta:

- **Etiquetas**

Las etiquetas, también conocidas como marcas, definen una serie de elementos que forman el léxico del lenguaje HTML. Se encuentran entre los signos de menor que (<) y mayor que (>). Podemos diferenciar entre dos tipos de etiquetas: cerradas y abiertas.

- **Etiquetas cerradas**

Consta de una etiqueta para la apertura que indica el comienzo de la etiqueta y otra de cierre que indica que hemos terminado de trabajar con la etiqueta en cuestión y lleva el símbolo "/" antes del nombre.

Por ejemplo:

* *

...

* *

- **Etiquetas abiertas**

Cuentan con una única palabra reservada para indicar el inicio y fin de la etiqueta a la vez.

Por ejemplo:

<hr>

*
*

<img>

- **Atributos**

Los atributos, al igual que las etiquetas, se pueden definir tanto en mayúsculas como en minúsculas, aunque los valores que se asignen a los atributos sí son sensibles a mayúsculas o minúsculas. Por ello, es recomendable utilizar siempre minúsculas para etiquetas y para los atributos y, de esta forma, evitar confusiones.

Las etiquetas pueden contener atributos si necesitan realizar alguna configuración sobre alguna característica determinada. Estos atributos se definen a continuación de la palabra reservada en la etiqueta de apertura separada por un espacio en blanco y antes del signo de cierre. Asignaremos el valor correspondiente al atributo a través del signo "=".

Cada comando cuenta con una serie de atributos con sus correspondientes valores, como, por ejemplo:

<p ALIGN = "left">

Definición de un párrafo y alineación del texto a la izquierda.

Es recomendable poner el valor entre dobles comillas para que sea más legible.

2.3. Estructura básica

Como ya hemos indicado, las etiquetas que utilicemos en HTML siempre van a ir entre los símbolos "<" y ">". Y cada vez que tengamos que cerrar una etiqueta, pondremos el nombre correspondiente comenzando con el símbolo "/".

Todas las etiquetas afectan al código que se encuentre delimitado entre la apertura y cierre de la etiqueta correspondiente.

```
<html>                          Inicio del documento
  <head>                        Comienzo de la cabecera
    <title>                     Inicio título del documento
     Título
    </title>                    Fin título del documento
    ...
   </title>
  </head>                       Fin de cabecera
  <body>                        Inicio del cuerpo
  ...
  </body>                       Fin de cuerpo
</html>                         Fin documento
```

HTML5, mediante unas etiquetas nuevas, añade una serie de características y elementos cuya función es facilitar la tarea a los autores de la aplicación web.

HTML5 se basa principalmente en una estructuración avanzada que se encarga de definir los contenidos agrupándolos en distintas etiquetas que tengan un nombre asignado correspondiente con la tarea que se va a realizar.

Algunas de estas etiquetas serían las que detallamos a continuación, que se refieren a:

- <**header**>. Encabezado de la página.

- <**nav**>. Enlaces de navegación.
- <**article**>. Algún artículo que se haya publicado.
- <**section**> Parte correspondiente a algún artículo.
- <**aside**>. Barras laterales.
- <**footer**>. Pie de página.
- <**dialog**>. Distintos diálogos o comentarios.

Además de estas etiquetas, HTML5 también cuenta con elementos como y , que utilizados para poder agrupar los diferentes elementos hijos haciendo uso de atributos como: *class, id* o *tittle*. De esta manera, se pretende utilizar una misma semántica con un estilo común.

2.3.1. Comentarios

Los comentarios son unas líneas que definen, de cara al usuario, lo que vamos realizando en cada momento, aunque no se interpretan por el navegador.

A lo largo del código fuente, por parte del desarrollador, es conveniente utilizar una serie de comentarios para explicar, con sus palabras, lo que se va realizando en cada momento. De esta forma, se pretende conseguir que, si alguien necesita trabajar con el código, sea capaz de interpretarlo de un simple vistazo gracias a los comentarios que aparecen en él.

Los comentarios van escritos entre los símbolos <!- - y - ->.

Por ejemplo:

<!- - Esto es un comentario - ->

2.3.2. Normas en XHTML

Para que un documento XHTML esté bien formado debe cumplir una serie de normas que detallamos a continuación:

- Todas las etiquetas y atributos deben ir en minúsculas.
- Todas las marcas deben cerrarse.
- No se permiten errores de acoplamiento entre las marcas.
- Los valores de los atributos deben ir entre comillas dobles.

2.4. Elementos de un HTML

En este apartado detallaremos todas las marcas que se pueden utilizar en este lenguaje para diseñar una página web. Como ya hemos visto anteriormente, debemos seguir la estructura básica de un documento HTML.

2.4.1. Cabecera

La cabecera del programa se encuentra siempre entre:

<head>

...

</head>

Y se encuentra dentro de un elemento superior como es <html>.

Las marcas que se pueden utilizar dentro de las cabeceras son:

- <title>, <base>, <meta>, <link>, <object>, <script>, <style>. **Todas son opcionales excepto la primera.**

Dentro de la cabecera es donde vamos a definir los elementos generales, como el título de la página:

<**title**>. Es el título que va a aparecer en el navegador web, en la barra superior.

<**meta**>. Encargada de indicar el contenido de nuestras palabras junto con las palabras clave. Esta directiva suele llevar dos atributos: *name* y *content*, que hacen referencia al nombre de la página y a sus principales contenidos.

<meta name = "description" content = "Página principal ILERNA con alumnos y profesores">

<meta name = "keywords" content = "Nombre y apellidos de alumnos matriculados">

Otro uso diferente de la etiqueta <meta> es el "refresco automático" que, transcurrido un tiempo estimado, pasa a actualizarse de nuevo la misma página. Es bastante recomendable para aquellas páginas en las que el contenido se modifica con mucha frecuencia.

<meta http-equiv = "refresh" content = "10"; "url = http://www.google.es">

Con esta instrucción estamos indicando que, pasados 10 segundos, se acceda a la página web de Google.

Otro elemento de mucha utilidad es el elemento **<base>**, al que le podemos indicar una URL base de algún documento, sonidos, gráficos, etc. Que hagan referencia a una determinada página web.

<**link**>. Define el formato del vínculo a la hora de indicar el comando *href* en el cuerpo.

<**script**>. Lo utilizaremos para insertar, entre su apertura y cierre, un script implementado en otro lenguaje mediante el atributo *type*, indicamos el lenguaje de programación, y mediante el atributo *src* definimos la URL del script en el caso de que sea externo, como, por ejemplo:

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

Esto es un ejemplo de un enlace externo a un archivo javascript que se encuentra en la carpeta js.

<**style**>. Marca que nos permite insertar una hoja de estilo en la cabecera. Mediante el atributo type, indicamos el lenguaje del estilo. En este caso, utilizaremos siempre el CSS.

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

Cuando creamos una página web, es conveniente que, al comienzo, realicemos una planificación de su diseño para, después, ordenar la información y recursos disponibles que se van a ofrecer. Para llevar a cabo esta tarea, es recomendable que hagamos uso de una estructura de directorios.

2.4.2. Cuerpo del documento

El cuerpo del programa se encuentra siempre entre:

<body>

…

</body>

Siempre situado detrás de la cabecera <head>. Va a contener todo el cuerpo correspondiente a una determinada página web junto con los elementos propios de la página como pueden ser gráficos, textos, imágenes, etc.

En el cuerpo de un programa también tenemos la posibilidad de definir una serie de acciones necesarias para eventos más concretos, entre los que podemos destacar:

- **onload**. Cuando se descarga el contenido completo de una página.
- **onunload**. Cuando un documento se va a descargar.
- **ononline**. Cuando hay conexión a Internet.
- **onoffline**. Cuando no hay conexión a Internet.
- **onafterprint**. Cuando terminamos de imprimir un documento.
- **onbeforeprint**. Antes de la ventana de impresión. ~~Cuando finalizamos la impresión de un documento.~~

Además, existen un conjunto de atributos (opcionales) de <body> que permiten realizar diferentes configuraciones sobre la apariencia de un documento. Hoy en día están obsoletos, ya que HTML5 no los soporta. Algunos de ellos son: bgcolor, text, link y vlink.

Existen algunos atributos que sí son recomendables utilizar en la marca <body>, como pueden ser:

id: para identificar el elemento para la hoja de estilo.

class: para asignar un nombre de la clase de la hoja de estilo y, de esta forma, poder mejorar su rendimiento.

title: para agregar un comentario y así lo puedan visualizar los navegadores.

style: aplicamos un estilo a este elemento.

A continuación, veremos todas las marcas que podemos utilizar en este parte del documento para poder aplicar distintos tipos de comandos, y así realizar la página web correspondiente. Los elementos del cuerpo se pueden dividir en:

- Elementos de bloque, cuando en su interior contiene otros elementos de HTML y la propia información, formando así una gran estructura.

- Elementos de línea, estos elementos, solo pueden contener datos y otros elementos de línea.

- **Cabeceras**

Existen seis tipos diferentes de cabeceras (elementos de encabezado). Las marcas de apertura y cierre son obligatorias, y el atributo que podemos utilizar es *align* para alinear este título.

Los tipos de cabecera son:

<h1> </h1>

<h2> </h2>

<h3> </h3>

<h4> </h4>

<h5> </h5>

<h6> </h6>

El texto que se escriba en h1 va a ser el del título de mayor tamaño hasta el de h6, que va a ser el menor.

<h1>Encabezado H1</h1>
<h2>Encabezado H2</h2>
<h3>Encabezado H3</h3>
<h4>Encabezado H4</h4>
<h5>Encabezado H5</h5>
<h6>Encabezado H6</h6>

- **Párrafos**

Es una de las marcas más utilizadas. No puede contener elementos de bloque en su interior. Todos los párrafos del documento web van a ser bloques delimitados por la marca p, dejando al navegador el ajuste al ancho de la página.

A la hora de insertar un espacio es necesario insertarlo mediante el carácter ** ,** ya que habrá navegadores que ignoren el espacio en blanco del teclado.

** ** → Escribe un texto en forma de párrafo.

** **

<blockquote> → Permite visualizar una cita con el margen izquierdo mayor produciéndose un efecto de una sangría.

</blockquote>

<pre> → Devuelve una copia exacta del texto respetando los espacios en blanco, tabulaciones y retorno de carro, es decir, tal y como se ha escrito.

</pre>

- **Saltos de línea**

**
** → Representa un salto de línea entre párrafos, ya que el salto de línea del teclado lo ignora el navegador. No necesita marca de cierre.

<hr> → Existe una forma de separar cada párrafo, utilizando una línea horizontal visible. No necesita etiqueta de cierre y podemos utilizar los siguientes atributos: align, noshade, size, width.

- **Semántica en textos**

Son una serie de elementos HTML que ofrecen un significado a una parte del contenido de un texto, haciendo uso del texto en negrita, subrayado o cursiva (, <u> </u> y).

	Negrita
	Cursiva
<big>	Agrandar el tamaño. Se puede introducir varios comandos para hacerlos más grande
<small>	Pequeño. Funciona de la misma forma que big
<s> o <strike>	Tachado de texto
<u>	Subrayado

Existen algunas etiquetas, bastante parecidas entre sí, que tienen como fin ofrecer un estilo diferente para alguna letra especial de un texto determinado.

- **Colores**

Podemos representar los colores mediante el símbolo de "#" seguido de tres pares de dígitos hexadecimales. El rango de colores indica la intensidad de los colores primarios (rojo, verde y azul), que en dígitos hexadecimales son (#RRVVAA).

Los diferentes pares de cifras hexadecimal van a oscilar desde 00 hasta FF proporcionando un rango que va desde 0 hasta 255 valores diferentes.

Otra forma de expresar colores en HTML es haciendo uso de una notación hexadecimal más corta, que utiliza un dígito para cada color (#JVA). En este caso, el rango de valores va a oscilar entre 0 y 15 0, si lo preferimos, podemos indicar el color de forma directa: *red*, *green*, *blue*.

- **Hipervínculos**

Los hipervínculos son elementos del lenguaje HTML que permiten acceder a otro recurso; es decir, lo podemos definir como un enlace que apunta a otro sitio web, un fichero, una imagen, etc.

La sintaxis que debemos utilizar a la hora de incluir un hipervínculo es:

<a>

de tal forma que el texto que se encuentre en esa directiva se puede convertir en un hipervínculo. Si hacemos clic con el ratón, nos debe llevar al sitio referenciado.

En el caso en el que el sitio web esté referenciado por un texto, debe aparecer subrayado y de otro color.

El atributo *href* es el que nos ofrece la posibilidad de crear un hiperenlace. Debemos indicar una URL que va a ser a la que queramos acceder al hacer clic en el hiperenlace. **A continuación, si en el elemento <a> no indicamos el atributo *href*, entonces el elemento representa un marcador de posición que va a ser a la que referencie otro hipervínculo en su atributo href.**

Disponemos de otro atributo *target* (opcional) que va a hacer referencia al destino en el que se va a mostrar la información disponible en esa dirección a la que nos lleva.

- **Anclas y vínculos internos**

Los vínculos internos permiten acceder a un sitio concreto dentro de una página web. Aunque, si queremos hacer uso de los vínculos internos, antes debemos establecer un ancla que es el punto fijo de posición al que accederemos tras un vínculo interno.

- **Rutas relativas y absolutas**

Absolutas. Aquellas que enlazan con páginas, cuya dirección absoluta se indica en el atributo *href* del comando *a*. Suelen ser páginas web externas a nuestro proyecto.

<a href = "http://www.google.es">

Las direcciones absolutas empiezan a direccionarse desde el comienzo de la ruta que indicamos.

Relativas. Aquellos enlaces cuya dirección relativa se indica en el atributo *href* del comando *a*. Suelen ser enlaces a páginas internas al mismo proyecto.

<a href = "./pagina2/pagina2.html">

Empiezan a direccionarse a partir del directorio actual.

- **Imágenes**

Las imágenes pueden estar en diferentes formatos como pueden ser los mapas de bits (archivos PNG, GIF, JPEG), documentos vectoriales de una página (archivos PDF, XML), mapas de bits animados, gráficos de bits animados, etc.

Aunque existen otros tipos de archivos que no se consideran imágenes, como pueden ser los archivos PDF de varias páginas, interactivos, documentos HTML, documentos sin formato, archivos SVG.

Gracias al elemento **<img>** podemos representar una determinada imagen ayudados por su atributo (obligatorio) *src*. En este caso, indicamos:

- La dirección válida en la que está la imagen que queremos visualizar.
- Una ruta relativa si es que la imagen está en alguna parte local.
- Una URL si se refiere a una imagen externa que se encuentra almacenada en una página web diferente.

El atributo ***alt*** nos permite que indiquemos un texto alternativo que sea capaz de representar el contenido de una imagen. Podemos utilizar esta forma en el caso en el que el navegador no pueda visualizar o descargar las distintas imágenes.

Las etiquetas **<figure>** y **<figcaption>** son novedosas para HTML5, y nos ofrece la posibilidad de agrupar una imagen junto con su información o leyenda.

- **Incorporación de imágenes**

Llegado el momento de añadir imágenes a las diferentes páginas web, debemos contar con que los navegadores permiten trabajar con ficheros que tengan formatos JPEG o GIF, ya que son los más recomendables.

Si queremos que una imagen se muestre en una web, en primir lugar necesitamos declarar una etiqueta **<img>**, que no necesita etiqueta de cierre.

Por ejemplo:

<img src="imagen.jpg" alt="Texto">

Aparte de estos dos atributos, también contamos con algunos más que mostramos en la siguiente tabla.

Etiqueta	Atributo	Valor	Significado
img	src	URL	Indica la URL de la imagen.
	alt	Texto	Define un texto alternativo por si no se encontrara la imagen deseada.
	Align	Top, middle, bottom, Left, Right, center	Alinea la imagen respecto al texto, tanto en sentido horizontal como en sentido vertical.
	Border	Número	Pone un borde o marco a la imagen. Se expresa en píxeles.
	Height	Número %	Especifica la altura que debe tener la imagen. Se expresa en píxeles o porcentaje.
	Width	Número %	Especifica el ancho que debe tener la imagen. Se expresa en píxeles o porcentaje.
	hspace	Número	Especifica en píxeles la separación horizontal entre el texto y la imagen.
	vspace	Número	Especifica en píxeles la separación vertical entre el texto y la imagen.

- **Uso de mapas sensibles**

Los mapas de imagen nos permiten definir distintas zonas "pinchables" dentro de una imagen. De tal forma que el usuario puede hacer clic sobre las zonas definidas y, cada una de estas, puede apuntar a una URL distinta.

Las diferentes zonas que se definen en una imagen se van creando mediante rectángulos, círculos y polígonos. A la hora de crear un mapa de imagen:

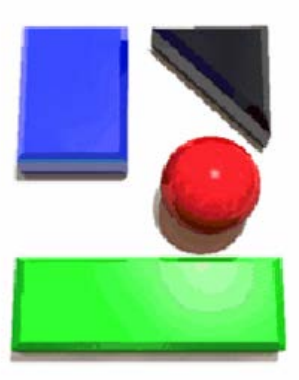
En primer lugar, insertamos la imagen original mediante la etiqueta **<img>.**

A continuación, utilizaremos la etiqueta **<map>** para definir las diferentes zonas o regiones de la imagen. Cada una de estas zonas, la definiremos con la etiqueta **<area>**.

Veamos un ejemplo práctico de los mapas de bits.

<map>	Mapa de imagen
Atributos comunes	Básicos, i18n y eventos.
Atributos específicos	Name = "texto". Nombre que identifica de forma única al mapa definido (es obligatorio indicar un nombre único).
Tipo de elemento	Bloque y en línea.
Descripción	Se emplea para definir mapas de imagen.

<area>	Área de un mapa de imagen
Atributos comunes	Básicos, i18n, eventos y foco
Atributos específicos	href = "url" – URL a la que se accede al pinchar sobre el área. nohref = "nohref" – Se emplea para las áreas que no son seleccionables. shape = "default | rect | circle | poly" – Indica el tipo de área que se define (toda la imagen, rectangular, circular o poligonal). coords = "lista de números" – Se trata de una lista de números separados por comas que representan las coordenadas del área. Rectangular = X1, Y1, X2, Y2 (coordenadas X e Y del vértice superior izquierdo y coordeandas X e Y del vértice inferior derecho). Circular = X1, Y1, R (coordenadas X e Y del centro y radio del círculo). Polígonal = X1, Y1, X2, Y2, ..., XnYn (coordenadas de los vértices del polígono. Si las últimas coordenadas no son iguales que las primeras, se cierra automáticamente el polígono uniendo ambos vértices).
Tipo de elemento	Etiqueta vacía
Descripción	Se emplea para definir las distintas áreas que forman un mapa de imagen.



Mediante este círculo, triángulo y los dos rectángulos, podemos acceder a 4 zonas diferentes de la imagen a través del siguiente código HTML.

```
<img src="imagen.gif" usemap="#mapa_zonas" />

<map name="mapa_zonas">

<area shape="rect" coords="20,25,84,113" href="rectangulo.html" />

<area shape="polygon" coords="90,25,162,26,163,96,89,25,90,24" href="triangulo.html"

<area shape="circle" coords="130,114,29" href="circulo.html" />

<area shape="rect" coords="19,156,170,211" href="mailto:rectangulo@direccion.com" />

<area shape="default" nohref="nohref" />

</map>
```

- **Listas**

— Listas numeradas

Ofrece la posibilidad de poder representar los diferentes elementos de una lista enumerándolos a todos dependiendo del lugar que ocupen.

En las listas numeradas, utilizan las directivas:

```
<ol>
  <li>Apartado 1</li>
  <li>Apartado 2</li>
  ...
</ol>
```

Y se muestra de la siguiente forma:

1. Apartado 1
2. Apartado 2

...

Por ser una lista numerada.

La directiva **<ol>** tiene disponibles los siguientes parámetros:

- ***start*** = numero → Permite seleccionar el número que deseemos que sea el primero de la lista.
- ***Type*** = tipo → Nos permite seleccionar el tipo de numeración que deseemos.

En HTML5, podemos diferenciar entre los siguientes tipos:

- **1**. Expresa números desde 1, 2, 3, etc.
- **a**. Expresa letras minúsculas a partir de a, b, c, etc.
- **A**. Expresa letras mayúsculas a partir de A, B, C, etc.
- **i**. Expresa números romanos desde i, ii, iii, etc.
- **I.** Expresa números romanos en mayúscula desde I, II, III, etc.

— Listas no numeradas

Ofrece la posibilidad de poder representar los diferentes elementos de una lista sin enumerar independientemente del lugar en el que se vayan a almacenar. Lo que sí lleva es una especie de "viñeta" para marcar la lista.

En las listas no numeradas, utilizan las directivas:

```
<ul>
   <li>Apartado 1</li>
   <li>Apartado 2</li>
   ...
</ul>
```

Por ser una lista no numera, se muestra de la siguiente forma:

- Apartado 1
- Apartado 2
- ...

La directiva **<ul>** tiene disponibles los siguientes parámetros:

- ***Type*** = tipo → Nos permite seleccionar el tipo viñetas que deseemos.

En HTML5, podemos diferenciar entre los siguientes tipos:

- **disk**. Expresa una viñeta en forma de disco.
- **circle**. Expresa una viñeta en forma de círculo.
- **square**. Expresa una viñeta en forma de cuadrado.

— Listas de definición o glosario

Ofrece la posibilidad de poder representar los diferentes elementos de un diccionario formado por su término y su definición.

En las listas de definición, se utilizan las directivas:

```
<dl>
   <dt>Coche</dt>
      <dd>Vehículo de cuatro ruedas</dd>
   <dt>Moto</dt>
      <dd>Vehículo de dos ruedas</dd>
   ...
</dl>
```

Y se muestra de la siguiente forma:

Coche

Vehículo de cuatro ruedas

Moto

Vehículo de dos ruedas

- **Agrupación de contenidos**

Esta marca sirve para agrupar varios elementos dentro de un bloque y permite asignarle un identificador

- **Tablas**

Son elementos de HTML que nos ofrecen la posibilidad de representar datos de una o varias dimensiones (matriz) con la información distribuida en filas y columnas.

Uno de los objetivos que pretende HTML5 es conseguir separar el contenido que se pretende mostrar de una página web, de la forma que lo ha presentado, mediante hojas de estilo CSS.

A la hora de definir una tabla, debemos hacerlo con la siguiente directiva:

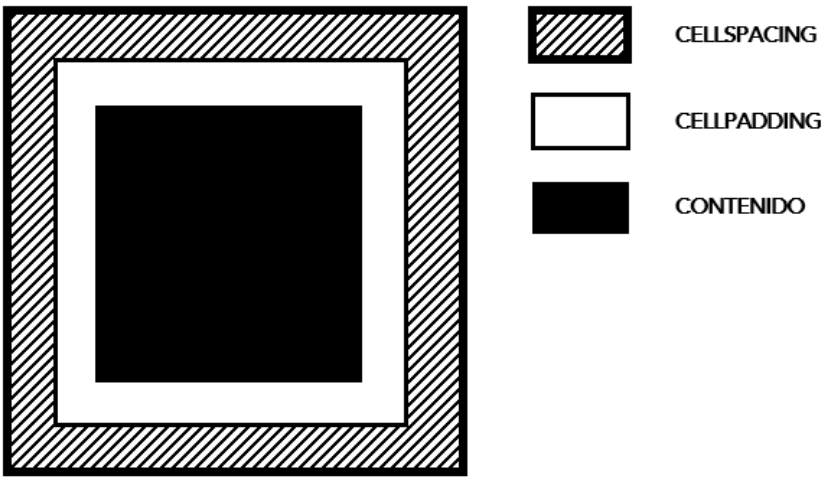
<table>

…

</table>

Hasta el momento, y para versiones anteriores a HTML5, se han podido utilizar distintos atributos como: *align* y *bgcolor*. Sin embargo, desde la aparición de HTML5, soporta los siguientes atributos:

- **summary**, realiza el resumen y la estructura de la tabla para facilitar la tarea de los buscadores.
- **width**, indica el ancho total de la tabla en distintas unidades. Podemos hacerlo en píxel (px) o porcentaje (%).
- **frame**, indica el nombre del marco en el que se puede visualizar la tabla.
- **rules**, especifica las líneas de división que serán visibles.
- **border**, determina el ancho del borde anterior.
- **cellspacing**, especifica el espacio existente entre las distintas celdas.
- **cellpadding**, especifica el espacio existente entre el borde de las celdas y el contenido.



Si tenemos una tabla, el primer elemento que podemos representar es el que aparece dentro de la etiqueta:

<caption>

</caption>

Representa el título de la tabla que lo contiene.

- **Filas, columnas y celdas**

Las tablas se componen de un número de filas y columnas que se pueden representar a través de la directiva:

<tr>

</tr>

Que permiten definir una nueva fila.

Después, cada fila, cuenta con una serie de elementos:

<td>

</td>

Nos permiten ir generando las diferentes columnas con su valor asignado dentro de una determinada fila.

Realiza columnas de cabecera y contiene un texto centrado y en negrita. *<th>*

</th>

Cada dato de una fila y una columna determinada, contiene un dato que denominamos valor.

- **Combinación de celdas**

Es posible combinar un conjunto de celdas de una determinada tabla por lo que van a afectar a su definición:

<colgroup> → Representa las columnas de una tabla

</col> → Indica el número de columnas que tiene una tabla

Disponemos de un conjunto de elementos que nos permiten referirnos a las diferentes partes de una tabla en el lenguaje HTML:

<thead> → Cabecera

<tbody> → Bloques de filas

<tfoot> → Pie

- **Formularios**

Utilizaremos los formularios cuando necesitamos recoger la información específica de un objeto en cuestión.

Los formularios están compuestos, entre otros, por diferentes elementos como las cajas de texto (*textBox*), casillas de verificación (**checkbox**), casillas de opción (*radio button*), listas desplegables y subformularios.

Toda la información que recogen los formularios debe tratarse por archivos que se deben implementar por el propio desarrollador. De esta forma, lo almacenaremos en bases de datos diseñadas previamente que, en este manual, no vamos a detallar ya que solo nos centraremos en la definición y diseño de los formularios.

Esta información también puede ser enviada o procesada mediante email o servidor web a través de un botón de envío (*submit*).

1. Propiedades de los formularios

Un formulario se define mediante el comando:

<form>

…

</form>

Por tanto, dentro de este bloque, debemos implementar todos los elementos necesarios que hemos enumerado previamente.

Esta marca *form* consta de varios atributos bastante importantes, como pueden ser los atributos:

action: indica el lugar al que se envían los datos.

method: indica le método de transferencia de los datos en el servidor web. Los valores que puede tomar este atributo pueden ser:

- **post**. Para el envío de los datos al usuario de forma codificada.
- **get.** Para el envío de los datos a una dirección web.

tarject: se utiliza para indicar o especificar dónde vamos a mostrar la respuesta del formulario. Los diferentes valores que puede tomar este atributo son:

"_blank". Se muestra en una ventana nueva.

"_self". Se muestra en la misma ventana del formulario.

"_parent". Se muestra en la ventana padre, es decir, la que precede al formulario.

name: identifica al formulario mediante el nombre. De esta forma, podemos llamarlo desde otro archivo. Recomendamos que el nombre de formulario sea único para facilitar su tratamiento.

2.Elementos de los formularios

Dentro de un formulario podemos tener distintas directivas que van a componer todo el diseño. Para comenzar, podemos agrupar un conjunto de elementos bajo un nombre, utilizando el comando **<fieldset> </fieldset>**.

A continuación, y mediante las etiquetas:

<input> </input>

Vamos a poder crear varios tipos de elementos dentro de un formulario, dependiendo del valor que asignemos al atributo *type*. Lo primero que recomendamos dentro de esta marca es especificar el valor del atributo *name*, ya que de esta forma, podemos identificar el elemento.

Seguidamente, mediante el atributo *type*, elegiremos el tipo de elemento que deseamos tener en el formulario.

<textarea> </textarea>

En este caso, definiremos un campo de texto de grandes dimensiones para que el usuario tenga la posibilidad de escribir todo lo deseado sin ningún tipo de limitaciones. Este comando se utiliza en muchos formularios cuando escribimos en un campo de observaciones.

Los atributos de esta marca serían los siguientes:

name. Identifica el nombre del área de texto.

cols. Número de caracteres que puede contener cada línea.

rows. Número de líneas del área de texto.

readonly. Para impedir que el usuario pueda editar este campo.

<button></button>

De esta forma insertamos un botón. Aunque también podemos hacerlo con el comando *input*. Podemos añadirle el atributo *type*, donde podemos indicar el tipo de botón que deseemos, (*submit* | *reset* | *button*).

<select>

<options>

</options>

</select>

Este bloque de comandos se utiliza para definir una lista desplegable con opciones. Tendremos tantas marcas *options* como opciones deseemos tener en nuestra lista.

<label></label>

Asignamos un título o etiqueta a un determinado campo del formulario.

<fieldset></fieldset>

Agrupa distintos elementos del formulario por distintas temáticas.

- **Marcos**

1. Definición de marcos (*frames*)

Los marcos ofrecen la posibilidad de poder mostrar varios archivos HTML en la misma ventana del navegador.

También tenemos la posibilidad de que estos *frames* interactúen como, por ejemplo, cuando presionamos un enlace en un *frame* para conseguir acceder a él, contamos con la posibilidad de cargar una página en otro *frame* diferente.

Es recomendable la utilización de *frames* cuando tengamos una situación que lo aconseje. El uso de los *frames* hace que los sitios sean menos accesibles y, por tanto, también es más difícil imprimir el contenido que tengan.

2. Uso de marcos

Vamos a ver la forma de utilizar dos *frames* para poder visualizarlos a la vez:

```
<html>
   <head>
      <title>Prueba de frames</title>
   </head>
   <frameset cols="20%,80%">
      <frame src="página2.html">
      <frame src="página3.html">
   <noframes>
   <p>El navegador no soporta frames</p>
   </noframes>
   </frameset>
</html>
```

En este ejemplo podemos ver la estructura de un documento HTML que realiza la función de esquema de una página web, dividida en dos columnas. La primera, ocuparía un 20% del total de la página, mientras que, la segunda, el resto.

En el primer marco se puede visualizar el documento "pagina2.html", y en el segundo marco lo enlazamos a la "página3.html". Por tanto, para hacer uso de los marcos, vamos a tener que cargar tres archivos html diferentes: el principal y los dos enlaces correspondientes a los marcos.

- **Objetos multimedia**

<iframe> </iframe>

Inserta un marco en un documento. Suele utilizarse para insertar publicidad o páginas de colaboración.

<object></object>

Inserta un objeto en un documento. El tipo del objeto viene determinado por el atributo, pudiendo ser: imágenes, aplicaciones de Java, animaciones u otros documentos HTML.

El atributo *type* es el encargado de determinar el objeto. Dependiendo de su sintaxis, podemos especificar, tanto la categoría como el formato del archivo. Por ejemplo, audio/mp3, vídeo/mpg, etc.

Además de este atributo, también podemos detallar, en los atributos *height* y *width*, tanto el ancho como el alto del objeto.

Y, para finalizar, en el atributo *data*, podemos determinar la ruta del objeto a visualizar, bien sea de audio o de vídeo.

<param>

Este comando no tiene etiqueta de cierre y permite inicializar las variables objeto. Presenta la siguiente sintaxis:

<param name="Nombre del parámetro" value = "Valor del parámetro">

2.5. Versiones de HTML y XHTML

2.5.1. HTML

Debido al rápido crecimiento de la web y a cómo van evolucionando las versiones HTML, aparece la necesidad de estandarizarlo para que autores y navegadores reconozcan el tipo de versión de HTML que pueden utilizar.

HTML llegó a convertirse en estándar en 1995 y, con el paso de los años, ha estado en un desarrollo constante de evolución. Hace algunos años, la versión de HTML recomendada por el W3C era HTML 4.01.

Cuando diseñamos una página web es conveniente especificar la versión de HTML con la que vamos a trabajar, y lo haremos utilizando en la etiqueta <!DOCTYPE> en la primera línea. Gracias a esta información, el navegador puede interpretar de forma correcta. HTML 4.01 cuenta con tres variantes disponibles de DTD:

- **HTML 4.01 Strict (*Strict DTD*).** Es la más restrictiva de todas ya que no permite la utilización de etiquetas que estén anticuadas, solo permite hacer uso de las propias que están definidas en HTML 4.01. Si queremos utilizar esta versión, escribimos en la primera línea la siguiente instrucción:

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

- HTML 4.01 Transitional (*Transitional DTD*).
- HTML 4.01 Frameset (*Frameset DTD*).

2.5.2. XHTML

En 1998 aparece por primera vez XTML 1. 0 de forma oficial. En enero del 2000, y tras una serie de mejoras, se convierte en uno de los lenguajes más recomendados en la creación de páginas web.

XML es un lenguaje que deriva de SGML, aunque resulta bastante más sencillo. Su objetivo principal es transmitir los datos a través de la web con una determinada estructura.

2.6. Herramientas de diseño web

El lenguaje de marcas es el más utilizado para la creación de páginas web por la facilidad de poder escribir el código en cualquier herramienta de procesador de texto.

En este apartado, vamos a detallar los distintos editores de los que disponemos, desde los más básicos hasta los más sofisticados para implementar dicho código.

2.6.1. Editores simples

Cualquier sistema operativo instalado en un equipo informático dispone de unas herramientas básicas para escribir un texto determinado. El sistema operativo Windows suministra la herramienta Bloc de Notas, así como la de NotePad para la creación de un fichero de texto con extensión .txt. Este tipo de fichero es uno de los más básicos que podemos encontrar en un equipo informático ya que no necesita utilizar ninguna herramienta sofisticada para su manipulación y visualización.

Haremos uso de esta herramienta básica para escribir un código en HTML con la salvedad de almacenarlo con la extensión que lo identifica: .html.

De esta forma, podemos tener nuestras páginas web ya codificadas de forma sencilla, intuitiva y fácil.

2.6.2. Editores avanzados

Debido al gran auge y complejidad en el que se ha convertido el desarrollo de las páginas web desde su existencia, cada vez es más complicada la implementación del código en este lenguaje de marcas, ya que en nuevas versiones aparecen nuevos comandos, con sus correspondientes atributos y valores, para tratar nuevos conceptos que el mercado obliga a insertar.

El código de una página web conlleva numerosas líneas de código, interacción con otros lenguajes, incrustaciones de códigos y enlaces externos, etc. Por todas estas características, cada vez resulta más complicada la identificación de las distintas marcas, y para el desarrollador es una tarea árida y complicada.

Para solucionar estos inconvenientes, aparecen los editores de texto avanzados con una interfaz amigable y sencilla de utilizar; mediante distintos colores, podemos diferenciar cada elemento de una parte del código en lenguaje HTML.

De esta forma, facilita bastante la tarea al desarrollador y a cualquier técnico que necesite visualizarlo.

Otra de las ventajas que nos ofrecen los editores avanzados es que, ante cualquier incidencia o error, provee de una serie de opciones para su fácil identificación.

Entre los ejemplos más conocidos, destacamos NotePad++ y Sublime Text, editores muy utilizados por la comunidad desarrolladora de código y con la característica de ser herramientas de código libre. Además, aunque sea de licencia de pago podemos mencionar Adobe Dreamweaver.

Descargas

Podemos descargarlos a través de las páginas:

https://www.sublimetext.com

https://www.notepad-plus-plus.org

2.6.3. Gestores de lenguajes HTML.

Una vez que ya tenemos diseñado todo el código de HTML con cualquier editor de los mencionados en el apartado anterior, el último paso que nos queda es la visualización de dicho código en cualquier navegador de internet.

Para crear cualquier página web, solo nos hace falta un editor y navegador web.

Existen distintos navegadores actualmente en el mercado, entre los que destacamos: Internet Explorer, Mozilla FireFox y Google Chrome.

A medida que avanzamos en nuevas actualizaciones, estos navegadores facilitan la visualización del código de la página y la identificación de algún error que pueda existir en ella.

2.7. Hojas de estilo

CSS (*Cascading Style Sheets*), las Hojas de Estilo en Cascada son un lenguaje de presentación que resultan imprescindibles para poder crear sitios web de calidad.

Uno de sus principales objetivos es poder trabajar el aspecto y formato de los documentos, y así dejar a HTML fuera de las tareas de presentación.

Este lenguaje presenta numerosas ventajas al utilizar separación entre contenidos y presentación.

```
<body>
...
<\body>
```

Mejora, de forma considerable, con esta otra opción:

```
<html>
   <head>
   ...
   <\head>
   <body>
   ...
   <\body>
<\html>
```

Entre las principales ventajas que se dan cuando trabajamos con CSS podemos destacar:

- Necesitamos menos código a la hora de escribir.
- Facilidad a la hora de generar código y mantenerlo.
- Documentos más legibles.

Sintaxis:

- **Incluir CSS en un documento**

Existen tres formas diferentes de añadir hojas de estilo a los documentos HTML.

- **Forma 1**: CSS en línea

```
<body>
    <p style="font-family: Verdana; font-size: medium;">HOLA MUNDO</p>
</body>
```

Puede utilizarse para seleccionar el estilo particular de línea.

- **Forma 2**: CSS interno

```
<html>
  <head>
    <style type="text/css">
    body {font-family: Courier New;}
    h1 {font-family: Arial; font-size: x-large;}
    p {font-family: Verdana; font-size: medium;}
    </style>
  </head>
  <body>
    <h1>Tipo de Fuente Arial y tamaño grande</h1>
    <p>Tipo de Fuente Verdana y tamaño mediano</p>
  </body>
</html>
```

Situado dentro de la etiqueta **<style>**. Podemos utilizarlo para indicar los estilos propios de esta página que se corresponden con los estilos globales propios del sitio web.

- **Forma 3**: CSS externo

Primero, creemos el archivo.css en una carpeta como /css.

```
body {margin: opx;}
td    {color: #oooooo;
       font-size: 12px;}
a     {color: #FF6600;
       font-weigth: bold;}
a:hover   {color:   #3366CC;}
```

Y después, vamos a insertar el enlace <link> en cada documento.

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

Otra forma que tenemos de unir archivos es mediante la regla @import.

```
<style type="text/css">
 @import "/css/estilos.css";
</styles>
```

Podemos utilizarla para indicar aquellos estilos globales que pueden compartir entre todas las páginas web del sitio. Si se produce colisión, la prioridad debe ser:

1º css en línea +prioridad

2º css interno ....

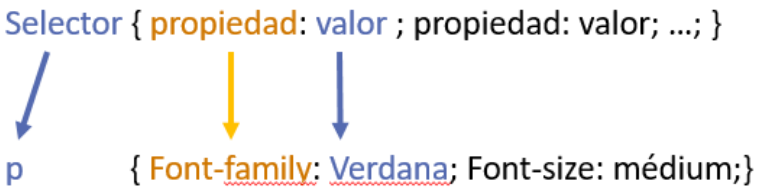
3º css externo -prioridad

En las tres formas que hemos analizado, en caso de que se produzca repetición, debe ser la última regla la que se aplique.

Si queremos dar más prioridad, podemos hacerlo haciendo uso de la palabra reservada **!important.**

- **Construir reglas CSS**

Mediante las reglas podemos unificar selectores más una declaración situada entre llaves. Se componen de una o varias parejas de propiedades con un valor y terminadas en punto y coma.



Podemos agrupar las diferentes reglas si comparten declaración o el selector:

h1 {font-family: Verdana; color: red;}

h2 {font-family: Verdana; color: red;}

Se resumen y queda de la siguiente forma:

h1, h2 { font-family: Verdana; color: red;}

Si tienen el mismo nombre (selector) pero no son iguales sus propiedades, sería:

h1 {font-family: Verdana;}

h1 {color: red;}

Se resumen y queda de la siguiente forma:

h1 { font-family: Verdana; color: red;}

Los comentarios se indican:

/..... */*

Selectores:

Vamos a ver, de forma detallada, una clasificación de los diferentes tipos de selectores:

- **Universal**. Permite seleccionar todos los elementos de una página.

```
* {margin: opx}
```

- **De tipo**. Permite seleccionar todos los elementos de una página cuya etiqueta sea igual que la del selector.

```
p {font-family: Verdana; color: red;}
```

- **Descendiente**. Supongamos que A es un elemento que desciende de B siempre que A se encuentra entre la apertura y cierre de B. Estos selectores se construyen a partir de dos o más selectores simples y separados por un espacio en blanco.

```
p a {font-size: 50px}
```

Si lo aplicamos al código, nos queda:

```
<p>Texto1</p>
<p><a>Texto2</a></p>
<p><span class="nieto"><a>Texto3</a></span></p>
```

Y, a continuación, podemos visualizar:

Texto 1

Texto 2

Texto 3

Si queremos excluir a uno de los hijos para seleccionar a partir de los nietos,

```
p * a {Font-size: 50px}
```

Y de esta forma, visualizamos solo el Texto3 a tamaño 50.

Texto 1

Texto 2

Texto 3

- **Hijo**

Nos permite seleccionar el primer descendiente de un elemento (hijo).

p> a {font-size: 50px}

Y visualizaremos solo el Texto2, tamaño 50 porque Texto3 no sería hijo.

Texto 1

Texto 2

Texto 3

- **Adyacente**

Se utiliza para poder seleccionar aquellos elementos que sean hermanos entre sí, es decir, que todos tienen el mismo padre.

h1 + h2 {Font-size: 50px}

- **Atributos:**

Tiene cuatro formas diferentes de seleccionar los elementos:

- **[atributo]** Elementos que tienen el mismo atributo, sin importar su valor.
- **[atributo = valor]** Elementos que tienen ese mismo atributo con el valor que se pasa.
- **[atributo ~= valor]** Elementos que tienen ese mismo atributo con, al menos, un valor igual al especificado.
- **[atributo |= valor]** Elementos que tienen ese mismo atributo y la palabra debe comenzar por el valor que se pasa.

- **De clase:**

Ofrece la posibilidad de seleccionar una determinada instancia de un elemento en concreto.

Veamos un ejemplo en el que existen dos instancias de un mismo elemento **<p>,** y para poder diferenciarlos, debemos hacer uso del atributo.

```
<h3 class = "grande">Texto1</h3>
<p class = "grande">Texto2</p>
<p class = "peque">Texto3</p>
```

Podemos construir de la siguiente forma:

```
p. grande {font-size: 50px}
```

Visualiza solo el Texto2 grande.

- **ID**

Nos ofrece la posibilidad de seleccionar una sola línea de una página completa.

```
#peque {font-size: 8px}
```

Visualiza el Texto3 en tamaño pequeño.

- **Pseudo-clases**

Permite seleccionar elementos según las indicaciones del usuario. Algunas de las pseudo-clases más frecuentes son las que se aplican a:

- **:link**. Los enlaces que nunca se visitan.
- **:visited**. Los enlaces que se han visitado una vez como mínimo.
- **:hover**. Los elementos indicados por el usuario, pero sin necesidad de ser activados.
- **:active**. Los elementos que se activan.
- **:focus**. Los elementos que tiene el foco.

Algunos ejemplos, pueden ser:

```
a:visited {color:black;}
a: active {color:red;}
```

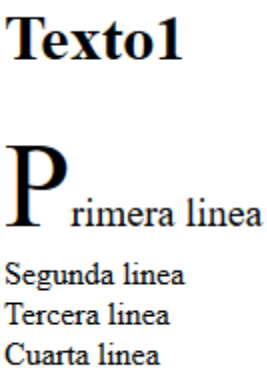
- **Pseudo-elementos**

Algunos de los pseudo-elementos principales son:

- **:first-line.** Para referirnos a la primera línea de un texto.
- **:first-letter.** Para referirnos a la letra de un texto.
- **:before.** Genera texto antes del contenido de un elemento.
- **:after.** Genera texto después del contenido de un elemento.

Veamos el siguiente ejemplo para comprobar cómo llevamos a la práctica estos elementos:

```
<html>
  <head>
    <style>
      h1:before {content: "Inicio --- "; }
       h1:after {content: " ---  Fin"; }
       p:first-line {font-size: 20px; }
       p:first-letter {font-size: 3em; }
    </style>
  </head>
  <body>
    <h1>Texto1</h1>
    <p>
      Primera linea <br>
      Segunda linea <br>
      Tercera linea <br>
      Cuarta linea <br>
    </p>
  </body>
</html>
```



Modelo de cajas:

- **Fundamentos**

El principal elemento del modelo de cajas de CSS, ya que confecciona en su totalidad la página web que estamos desarrollando. Los elementos que deseamos añadir al documento HTML se van a representar de forma automática a través de cajas rectangulares.

En un comienzo, son cajas invisibles a no ser que añadamos algún tipo de borde o color de fondo.

Las diferentes partes que podemos encontrar en una caja, junto con el orden en el que aparecen, son:

1. Contenido (content): texto, imágenes, vídeos, listas del elemento.

2. Relleno (padding): espacio que queda entre el contenido y el borde.

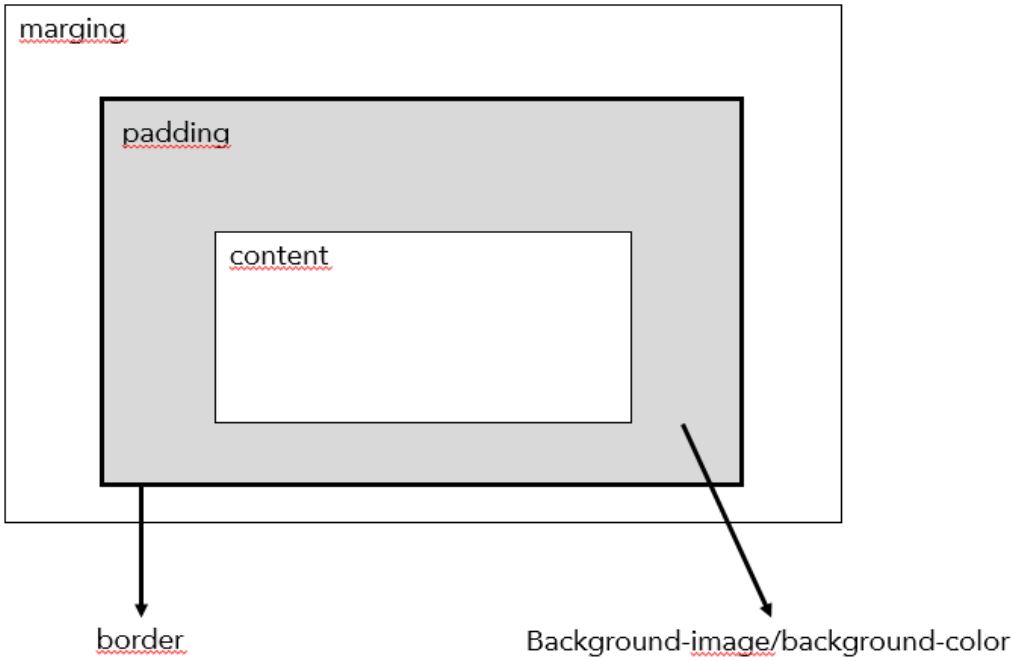
3. Bordes (border): la línea que agrupa el contenido junto con su relleno.

4. Imagen de fondo (background-image): es la imagen que se visualiza detrás del contenido y del espacio de relleno.

5. Color de fondo (background-color): el color que se visualiza detrás del contenido y del espacio de relleno.

6. Margen (margin): es el espacio que queda libre entre la caja y el resto de cajas adyacentes.

- *Padding* y *margin* son transparentes.
- En el espacio que ocupa *margin*, se visualiza el color o la imagen de fondo.
- El espacio que ocupa *margin* muestra el color o la imagen de fondo del elemento padre.
- Si ningún elemento padre tiene asignado un color o imagen de fondo, se va a mostrar el correspondiente a la página.
- Si en una caja definimos un color y una imagen de fondo, tiene más prioridad la imagen, ya que se va a mostrar antes.
- Si se da el caso en el que la imagen de fondo no cubra toda la caja del elemento por completo, se visualizará el color de fondo correspondiente de esas determinadas zonas.



- **Ancho, alto**

Podemos controlar el ancho y el alto de una caja mediante **width** y **heigth**.

Los valores que pueden tomar, podemos indicarlo mediante:

<numérico px>|<numérico %>|inherit|auto

Donde:

<numérico px> permite establecer los valores fijos de ancho y alto (en pixeles) de la caja.

<numérico %> permite establecer los valores fijos de ancho y alto (en porcentaje) del elemento padre.

<inherit> permite establecer los valores fijos de ancho y alto que se heredan del padre.

<auto> es el valor por defecto que permite determinar que el navegador sea quien calcule los valores de ancho y alto dependiendo del espacio que quede disponible en la página.

Por ejemplo:

#menu {width: 180px; heigth: 600px;}

* *

…

* *

Podemos aplicar el ancho y el alto a todos los elementos de bloque, a las diferentes imágenes y a las columnas de una tabla determinada.

- **Margin, Padding**

Margin es una propiedad "*shorthand*" que nos permite centrar de forma horizontal una caja.

Podemos distinguir entre las propiedades:

margin, margin-top, margin-rigth, margin-botton, margin-left

Que significan: margen, margen superior, derecho, inferior, izquierdo.

Y puede tomar los valores:

<numérico px, em>|<numérico %>|inherit|auto

- Se puede aplicar a todos los elementos menos a margin-top y margin-botton.

Padding es otra propiedad que utilizaremos para los diferentes tipos de relleno.

Podemos distinguir entre las propiedades:

padding, padding-top, padding-rigth, padding-botton, padding-left

Que significan: relleno, relleno superior, derecho, inferior, izquierdo.

Y puede tomar los valores:

<numérico px, em>|<numérico %>|inherit|auto

- Se puede aplicar a todos los elementos excepto cabeceras y pies de tablas.

- **Bordes**

Presentan tres tipos de propiedades para los bordes, como son: ancho, color y estilo.

- **Ancho**

Podemos distinguir entre las propiedades:

border-width, border-top-width, border-rigth-width, border-bottom-width, border-left-width.

Que significan: ancho de borde, ancho de borde superior, ancho de borde derecho, ancho de borde inferior, ancho de borde izquierdo.

Y puede tomar los valores:

<numérico px, em>|thin|médium|thick|inherit

- Se puede aplicar a todos los elementos.

- **Color**

Podemos distinguir entre las propiedades:

border-color, border-top-color, border-rigth-color, border-bottom-color, border-left-color.

Que significan: color de borde, color de borde superior, color de borde derecho, color de borde inferior, color de borde izquierdo.

Y puede tomar los valores:

<color>|transparent|inherit

- Se puede aplicar a todos los elementos.

- **Estilo**

Podemos distinguir entre las propiedades:

border-style, border-top-style, border-rigth-style, border-bottom-style, border-left-style.

Que significan: estilo de borde, estilo de borde superior, estilo de borde derecho, estilo de borde inferior, estilo de borde izquierdo.

Y puede tomar los valores:

none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset | |inherit

- Se puede aplicar a todos los elementos.
- Si el valor del estilo se corresponde con none o hidden, no va a aparecer borde.

- **Colores y Fondos**

  - **Color**

En CSS, podemos especificar los colores de dos formas diferentes:

— **Palabras reservadas**

Aliceblue, antiquewhite, aqua, aquamarine, azure, beige, bisque, black, blue, blueviolet, Brown, coral, cyan, darkblue, darkmagenta, …

Un ejemplo podría ser:

p {color: yellow}

— **Números hexadecimales en el modelo RGB**

Hace referencia al modelo de colores aditivo, con el que vamos a poder ir obteniendo colores utilizando la mezcla de los tres colores primarios: *red, green* y *blue* con sus cantidades correspondientes.

Un ejemplo sería:

p {color: #ff0}

p {color: #ffff00}

p {color: rgb (255, 255, 0)}

p {color: rgb (100%, 100%, 0%)}

Estas cuatro formas de representar colores son iguales y equivalen al color amarillo.

  - **Fondo**

Solo vamos a poder apreciar el color de fondo en las zonas que ocupe el contenido y en su relleno.

Para elegir un determinado color para una página entera, debemos utilizar el elemento <body> para aplicar el fondo.

Debemos considerar una serie de propiedades fundamentales cuando trabajamos con el fondo:

– **Color de fondo (background- color)**

  Puede tomar los valores: **<color> | transparent | inherit**

  - Se puede aplicar a todos los elementos.
  - Primer valor: transparent.

– **Imagen de fondo (background- image)**

  Puede tomar los valores: **<url ("ruta de la imagen")> |none | inherit**

  - Se puede aplicar a todos los elementos.
  - Primer valor: none.

- **Repetir imagen de fondo hasta completar elemento (background- repeat)**
  Puede tomar los valores: **repeat | repeat-x | repeat-y | no repeat | inherit**
  - Se puede aplicar a todos los elementos.
  - Primer valor: repeat.

- **Desplazamiento de la imagen desde la esquina superior izquierda al sitio especificado mediante desplazamiento horizontal y vertical (background-position)**
  Puede tomar los valores:
  (<num%> | <medida> | left | center | rigth | inherit) y
  (<num%> | <medida> | top | center | botton | inherit)
  - Se puede aplicar a todos los elementos.
  - Primer valor: 0% 0%. Si alguno se omite, asume el valor de 50% por defecto.

- **Controla el comportamiento de la imagen de fondo respecto a la scroll-bar (background-attachment)**
  Puede tomar los valores: **fixed | scroll | inherit**
  - Se puede aplicar a todos los elementos.
  - Primer valor: scroll.

- **Posicionamiento**

En CSS tenemos cinco formas diferentes para poder establecer la posición en pantalla de una caja determinada: estático, relativo, absoluto, fijo y flotante.

A la hora de establecer la posición, debemos tener en cuenta una serie de propiedades: position, top, botton, rigth, left y float.

Veamos de forma más detenida algunas de estas propiedades.

- **Position** (Establece la posición de la caja por pantalla).
  Puede tomar los valores: **static | relative | absolute | fixed | inherit**
  - Se puede aplicar a todos los elementos
  - Valor por defecto: static.

- **Top, botton, rigth, left** (establece el desplazamiento vertical u horizontal de la caja)
  Puede tomar los valores:
  <número px>|<numero %>|auto|inherit
  - Se puede aplicar a todos los elementos.
  - Valor por defecto: auto.

- **Float** (establece la posición flotante de la caja)
  Puede tomar los valores: **left | rigth | none | inherit**
  - Se puede aplicar a todos los elementos
  - Valor por defecto: none.

A continuación, vamos a ver los diferentes tipos de posicionamiento que nombramos al iniciar este apartado.

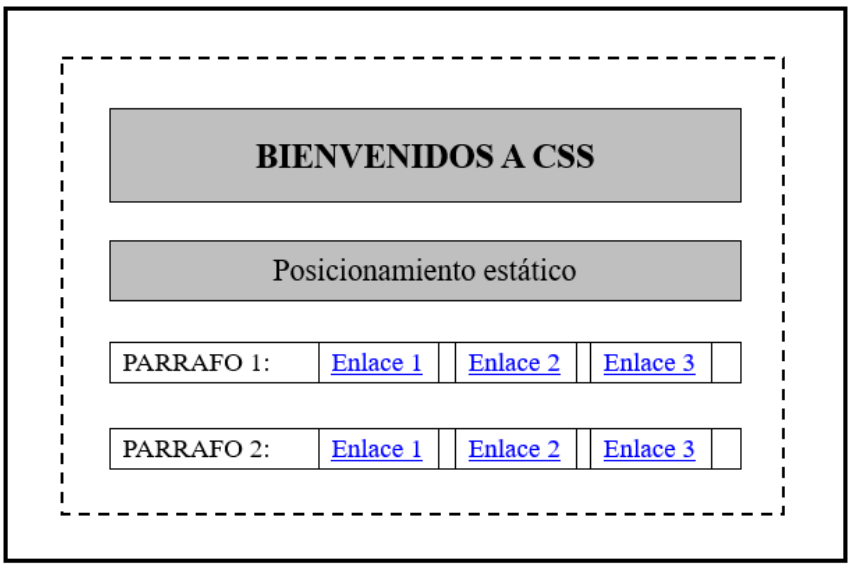
- **Posicionamiento estático** {position: static}

Es el posicionamiento que tienen todos los navegadores por defecto.

Si los elementos que debemos posicionar son de:

- **Bloque**. Como por ejemplo un párrafo, vamos a colocar las cajas en forma vertical una debajo de otra.
- **Línea**. Como un hipervínculo, vamos a colocar las cajas una al lado de otra, de forma horizontal, dentro del elemento bloque.
- No caben en una sola línea, entonces, deben ocupar las líneas siguientes.

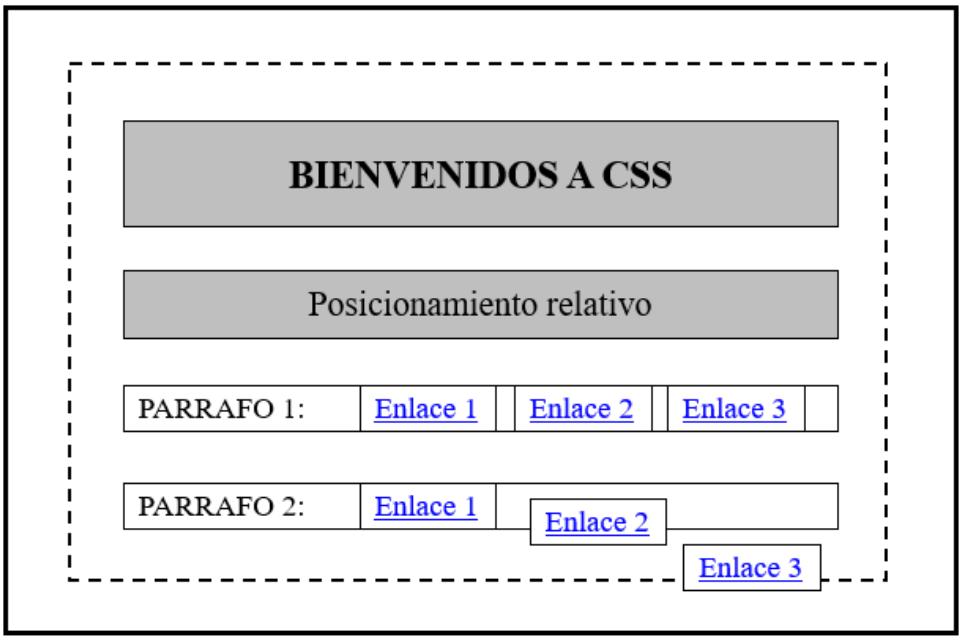
```
<style>
   ...
   a {background-color: yellow;}
</style>
<body>
   ...
   <p> PARRAFO 2:
      <a href="#"> Enlace 1 </a><a href="#"> Enlace 2 </a>
      <a href="#"> Enlace 3 </a>
   </p>
</body>
```



- **Posicionamiento relativo** {posición: relative}

Desplaza la caja un valor determinado con respecto al posicionamiento estático.

```
<style>
   ...
   a {background-color: yellow;}
   a.abajo {position: relative; top: 1em;}
   a.mas_abajo {position: relative; top: 2em;}
</style>
<body>
   ...
   <p> PARRAFO 2:
      <a href="#"> Enlace 1 </a>
      <a href="#" class= "abajo"> Enlace 2 </a >
      <a href="#"  class="mas_abajo"> Enlace 3 </a>
   </p>
</body>
```

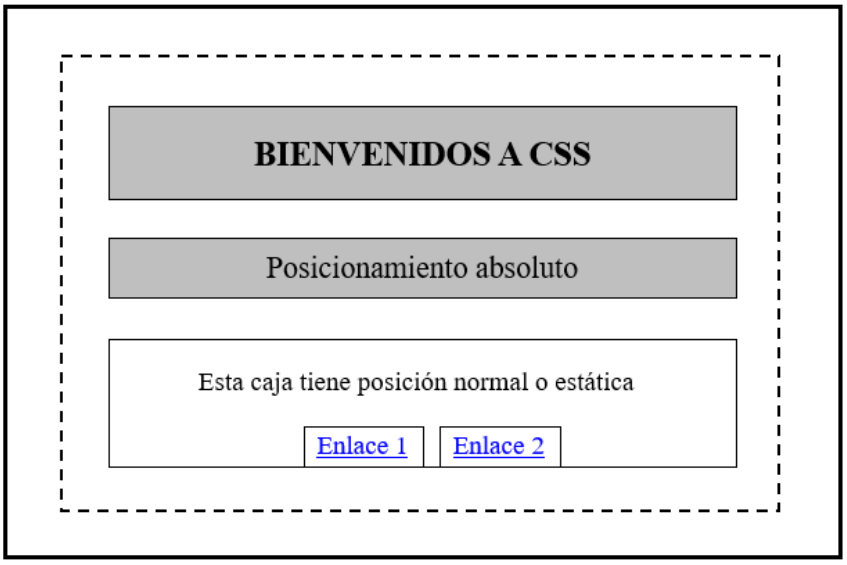


- **Posicionamiento absoluto** {position_absolute}

Desplaza la caja un valor determinado con respecto al elemento contenedor.

Para los desplazamientos, utilizamos top, botton, left rigth y podemos tomar como origen de coordenadas la esquina superior izquierda del primer contenedor que nos encontremos que no sea static.

```
<style>
   ...
   a {background-color: yellow; border: 2px solid;}
   a.absoluto {background-color: orange;
                position: absolute; top: 30px; left: 30px;}
</style>
<body>
   ...
   <div>
      <p>Esta caja tiene posición normal o estática</p>
      <a href="#" class="absoluto"> Enlace 1 </a>
      <a href="#"> Enlace 2 </a>
      <a href="#"> Enlace 3 </a>
   </div>
</body>
```



- **Posicionamiento fijo** {position fixed}

Es bastante parecido al posicionamiento absoluto, aunque en este caso, cuando el usuario se mueve por la página, las cajas permanecen fijas en el mismo sitio.

- **Posicionamiento flotante** {float: left | rigth | inherit | none}

Permite desplazar las cajas hacia la derecha o hacia la izquierda de la línea en la que se encuentran.

Las demás cajas siguen permaneciendo en el mismo sitio que deja la caja flotante. Con toda esta serie de cambios, no llega a producirse solapamiento entre las cajas gracias a la propiedad ***float***, que tiene en cuenta todas las cajas flotantes.

- **Visualización**

Vamos a definir de forma más específica las cuatro propiedades que nos podemos encontrar en CSS a la hora de visualizar las diferentes cajas.

- **Display**

Podemos distinguir entre los valores:

inline | block | none | list-item | run-in | inline-block | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | inherit

Que significan:

El valor **inline** se refiere a cualquier elemento html se muestre como un elemento en línea. Podemos utilizarlo cuando tengamos que mostrar listas de forma horizontal.

El valor **block** se refiere a que cualquier elemento html se pueda mostrar como elemento de bloque. Podemos utilizarlo en los diferentes enlaces que forman parte del menú de navegación.

El valor **none** permite ocultar de forma completa un elemento desplazándolo de la página. Los demás elementos se mueven para ocupar su posición.

- **Visibility**

Podemos distinguir entre los valores: *visible | hidden | collapse | inherit*

Que significan:

El valor **visible** es la opción seleccionada por defecto y muestra la caja.

El valor **hidden** oculta la caja sin modificar la posición de las demás.

El valor **collapse** lo podemos aplicar a las diferentes filas y columnas de una tabla y nos ofrece la posibilidad de poder visualizar algunos contenidos de una tabla en cuestión.

El valor **inherit** es heredado.

- **z- index**

Sirve para asignar la posición de la caja respecto al eje z. De esta forma, conseguiremos ver la profundidad con respecto al documento.

Podemos distinguir entre los valores: *auto | <nº entero> | inherit*

- **overflow**.

Vamos a utilizarlo para controlar que no se produzca desbordamiento del texto en una caja.

Podemos distinguir entre los valores: *visible | hidden | scroll | auto | inherit*

Que significan:

El valor **visible** nos permite observar el contenido que se desborda.

El valor **hidden** oculta el texto que se desborda.

Scroll deja la tabla con su mismo tamaño y, además, crea las barras de desplazamiento vertical y horizontal.

El valor **auto** también mantiene el tamaño de la caja, aunque en este caso, solo van a aparecer las barras si son necesarias.

Y el valor **inherit** es heredado.

- **Texto**

Dispone de una serie de propiedades que detallaremos a continuación.

Propiedad: valor	Significado
Color: <color> | inherit	Color del texto
Font-family: <fuente> | inherit	Tipo de fuente de letra
Font-size: <absoluto> | <relativo> | | <%>	Tamaño de letra
Font-weight: normal | bold | bolder |lighter | 100 | 200 | 300 | 400 | 500…	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-though || 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 líneas
Letter-spacing: normal | <medida> | inherit	Espaciado entre letras
White-space: normal | nowrap | pre | pre-line | pre-wrap | inherit	Tratamiento de los espacios en blanco
Word-spacing: normal | <medida> | inherit	Espaciado entre palabras

- **Listas**

Propiedad: valor	Significado
List-style-type: disc | circle | square | decimal | decimal-leading-zero | lower-roman | upper-roman | lower-greek | lower-latin | upper-latin | armenian | georgian | lower-alpha | upper-alpha | none | inherit	Viñeta usada para los elementos de la lista
List-style-position: inside | outside | inherit	Posición de la viñeta
List-style-image: <url> | none | inherit	Sustituye la viñeta por una imagen
List-style: (propiedad shorthand)	Combina las anteriores

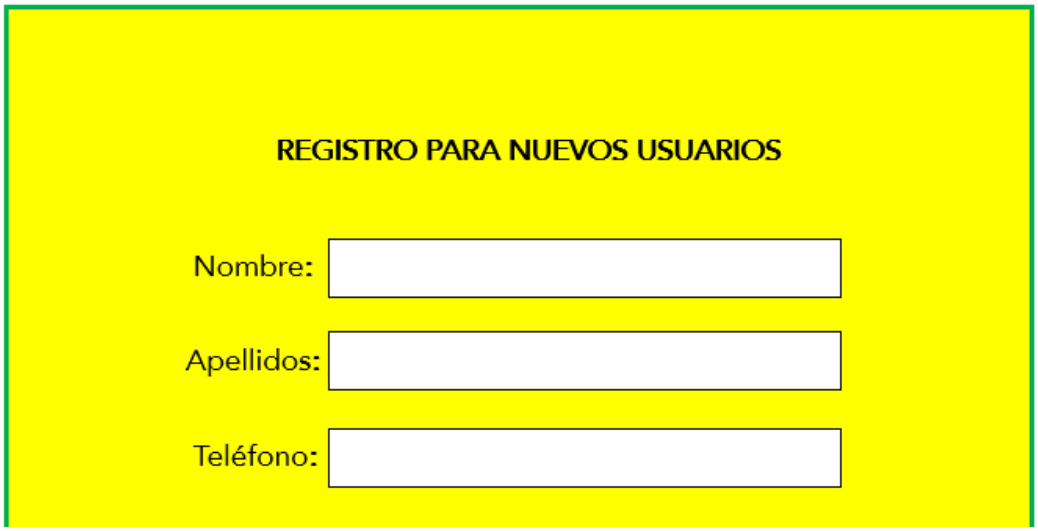
- **Tablas**

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

- **Formularios**

Todas las propiedades que hemos visto hasta el momento se pueden aplicar a un formulario para mejorar su aspecto y hacer más fácil la entrada de datos.

A modo de ejemplo, vamos a crear un formulario con una imagen de fondo. Supongamos un fondo amarillo con borde más ancho para el cuadro de texto activo.



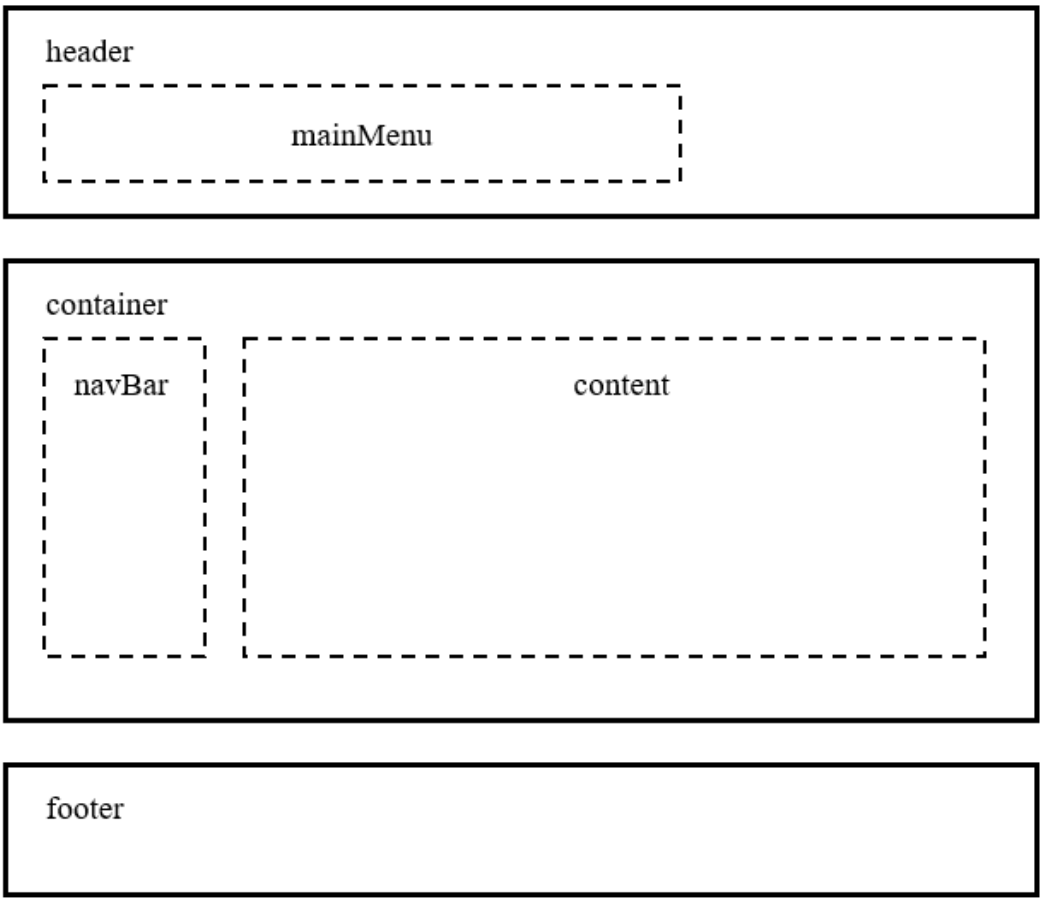

```
<style type="text/css">
   ...
   :focus {background-color: yellow; border: 2px solid;}
</style>
```

- **Layout**

Podemos representar el diseño general de una estructura de página a través de capas ** **. Las podemos clasificar según el número de columnas.

- **Layout líquido** es aquel que tiene un ancho de capas variable, haciendo uso de los porcentajes, para conseguir que se adapte a las medidas de que tenga la pantalla.

Supongamos un diseño de página líquido a dos columnas con la navegación hacia la izquierda.



Entre sus características más importantes destacamos:

- Las capas header, container y footer tienen las propiedades width: 100%, margin: 0, padding: 0 y overflow: auto.
- Las capas internas "navBAR" y "content" tienen propiedades float: left y margin 1%.
- El elemento **<body>** presenta la propiedad min-width: 600px para conseguir evitar que se descomponga la estructura.
- Tenemos la posibilidad de seguir añadiendo todas las capas internas que deseemos.

- **Prioridad**

A parte de las opciones de las hojas de estilos definidas por los usuarios, hay otras dos opciones disponibles que son la del navegador y la del usuario:

- **La hoja de estilo del navegador**

Va a ser la primera que apliquemos y la vamos a utilizar para definir el estilo inicial que vamos a aplicar a todos los elementos por defecto.

A la hora de comprobar sus diferentes valores, podemos hacerlos desde la herramienta "*Developer tools*" desde Chrome.

- **La hoja de estilo del usuario**

La podemos configurar mediante una opción avanzada del navegador. Es bastante útil para personas de edad más avanzada o para personas con dificultad visual, ya que permiten aumentar el tamaño del texto.

Para configurar los diferentes estilos, podemos hacerlo a través de Chrome, mediante la ruta: Opciones/ configuración avanzada/ contenido web.

El orden, según la prioridad, va a ser:

Primero, las reglas de los navegadores; a continuación, las reglas que definen los usuarios; finalmente, las reglas que define el diseñador que, al estar más arriba, serán las primeras en poder verse.

Miscelánea:

- **Estructuración del código CSS**

Cuando escribimos código CSS es conveniente distribuirlo en hojas independientes de estilos y, dentro de cada hoja, agruparemos las diferentes reglas que necesitemos para cada función.

Un ejemplo de organización podría ser el siguiente:

Estilos globales (contenido <html> y <body>)

Estilos de layout (ancho, alto, posición, etc. De las capas principales)

Estilos de cada capa

- **Aplicación web**

Hace referencia a las diferentes herramientas que tenemos disponibles para poder visualizar nuestro sitio en distintos navegadores sin que necesitemos instalarlos.

- **Sitios web orientados al diseño**

http://librosweb.es/libro/css/

https://desarrolloweb.com/css/

https://www.w3.org/Style/Examples/011/firstcss.es.html

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

3. Definición de esquemas y vocabularios de XML:

Como un avance en los lenguajes de marcas que venimos estudiando, vamos a definir en este apartado el lenguaje de marcado extensible (XML). Se trata de un lenguaje que almacena información a partir de unas etiquetas o marcas creadas por el usuario. Al ser un lenguaje de marcas, como los definidos anteriormente, una vez que están escritos, debemos comprobar si está bien formado. Adicionalmente, veremos aquellos mecanismos de los que disponemos para validar un documento, en especial, mediante dos técnicas: los ficheros DTD y los esquemas XML.

A continuación, veremos algunos lenguajes basados en XML, como son: SVG, WML o RSS.

3.1. Validación. Documentos XML válidos

En apartados anteriores hemos podido comprobar las características principales que debe tener un documento para estar bien formado. Ahora, nos centraremos en las reglas que debemos cumplir para que un documento sea válido.

Podemos decir que un documento XML es válido si cumple una serie de reglas de validación que van a especificar la estructura gramatical y la sintaxis que debe tener dicho documento.

Todos los documentos válidos, deben estar bien formados, aunque, NO todos los documentos bien formados son válidos.

3.1.1. Validación de documentos XML con DTD

Aunque hoy en día no es muy utilizada, esta técnica de validación consiste, mediante un DTD (*Document Type Definition*, Definición de Tipo de Documento), en especificar aquellos elementos que deben aparecer, junto con el orden que den seguir, los que son obligatorios y los que no, etc.

- **Generación automática de DTDs**

En muchos casos, disponemos de una serie de documentos XML con una gran cantidad de datos, por lo que aquellos DTD que los tienen que validar van a ser bastante extensos. Para facilitar este proceso, disponemos de una serie de herramientas que se pueden generar de forma automática. Entre los que diferenciamos:

- **Generadores online:**

http://xml.mherman.org/

- **Editores XML que pueden generar un DTD partiendo de un documento XML**

  Engloba todos los anteriores.

  XMLSpy de Altova

  XMLPad Pro Edition de WMHelp

- **Estructura y elementos de un DTD**

Los **DTD** podemos encontrarlos de diferentes formas: dentro del propio documento XML, en un archivo independiente y algunos, incluso, pueden mostrarse en una parte interna y otra externa.

Declaración:

Para declarar un DTD, lo haremos siempre al comenzar un documento XML:
<!DOCTYPE>
Y debemos seguir una serie de reglas que lo forman:

→ **Ubicación**. Donde se encuentran las reglas del DTD.

Interno. Cuando las reglas se encuentran en el mismo documento XML.

Externo. Cuando las reglas se encuentran en un archivo independiente.

Mixto. Combina los dos anteriores. Las reglas se encuentran en ambas partes.

→ **Carácter**. Indica si el DTD es de uso privado o público.

Uso privado. Identificado por la palabra SYSTEM.

Uso público. Identificado por la palabra PUBLIC. Debe ir con la palabra FPI (*Formal Public Identifier*, Identificador Público Formal), que identifica al DTD de forma universal.

Entre las diferentes combinaciones posibles, encontramos las siguientes:

Sintaxis	Tipo de DTD
<!DOCTYPE elemento_raíz [reglas]>	Interno (luego privado)
<!DOCTYPE elemento_raíz SYSTEM URL>	Externo y privado
<!DOCTYPE elemento_raíz SYSTEM URL [reglas]>	Mixto y privado
<!DOCTYPE elemento_raíz PUBLIC FPI URL>	Externo y público
<!DOCTYPE elemento_raíz PUBLIC FPI URL [reglas]>	Mixto y público

Entre los distintos atributos, distinguimos:

- Es el nombre del elemento raíz de un documento XML. Se encuentra a continuación de DOCTYPE.
- Es la declaración privada o pública. Se encuentra a continuación del elemento raíz y sirve para identificar si el DTD es de uso privado o público.
- Identificador, solo va a aparecer si el DTD es PUBLIC.

Estructura del FPI: formado por cuatro campos diferentes, que se refieren a:

- Primer campo: puede ser formal o no formal.
    - DTD no aprobado por una norma formal.
    - DTD aprobado por un organismo no oficial.
    - DTD aprobado por un organismo oficial.
- Segundo campo: es el nombre del organismo responsable.
- Tercer campo: hace referencia al tipo del documento descrito.
- Cuarto campo: indica el idioma del DTD.

- **Componentes del DTD**

Los principales componentes que tenemos en un DTD son: elemento, atributo, entidad y notación.

A continuación, detallamos más información sobre ellos:

- **<!ELEMENT>**

Es una declaración de un tipo de elemento que nos señala si existe un elemento a un determinado documento XML.

Sintaxis:

<!ELEMENT nombre_elemento modelo_contenido>

Donde *nombre_elemento* debe coincidir con el mismo que tenga el documento XML.

Y *nombre_contenido* indica:

- Una regla, que puede ser:

  ANY (cualquier cosa). La utilizamos en la construcción del DTD para validar la descripción de un elemento sin llevar a cabo ningún tipo de comprobación sintáctica.

  EMPTY (elemento vacío). Tiene posibilidad de describir un elemento que no tenga descendientes.

  Por ejemplo, la descripción de un elemento br.

  <!ELEMENT br EMPTY>

- **Datos**. Se refiere a aquellos caracteres, que pueden ser textuales, numéricos o de cualquier otro formato que no disponga de ninguna marca (etiqueta). Lo podemos describir como #PCDATA y debe aparecer entre paréntesis.
  Por ejemplo,
  <ELEMENT> titulo (#PCDATA)

- **Elementos descendientes**. Es conveniente que aparezcan entre paréntesis y se basa en una serie de reglas.
  - **Cardinalidad de los elementos**
    Indica la cantidad de veces que aparece un determinado elemento.

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

- **Secuencia de elementos**
  Secuencia que determina en qué orden aparece o debe aparecer un elemento.

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

- **Contenido mixto.** Consigue mezclar distintos textos de elementos descendientes.

o **<!ATTLIST>**

Declaración del tipo de atributo que cuenta con la posibilidad de poder indicar si existe un elemento determinado en un documento XML. **Sintaxis:**

```
<!ATTLIST nombre_elemento
        nombre_atributo tipo_atributo caracter
        nombre_atributo tipo_atributo caracter
        ...
>
```

Donde, ***nombre_atributo*** se debe corresponder con un nombre XML válido carácter del atributo hace referencia a:

- Un valor de texto que debe ir entre comillas.
- #IMPLIED es opcional y no lleva asignado ningún valor por defecto.
- #REQUIRED es un valor obligatorio, aunque no lleva asignado ningún valor por defecto.
- #FIXED, cuando el atributo es de carácter obligatorio y tiene asignado un valor por defecto que va a ser único para el atributo.

En cuanto a los tipos de atributos, podemos diferenciar:

- **CDATA**. Hace referencia a aquellos caracteres que no disponen de etiquetas.
- **ENTITY**. Asociado al nombre de una entidad.
- **Enumerado**. Lista de distintos valores entre los que se debe encontrar el valor uno.

- **ID**. Identificador único que permite identificar elementos.
- **IDREF**. Valor correspondiente a un atributo ID de un elemento diferente.
- **IDREFS**. Permite representar una gran cantidad de IDs correspondientes a otros elementos, que deben estar separados entre ellos por un espacio en blanco.
- **NMTOKEN**. Nombre que no tiene ningún espacio en blanco en su interior. Si existen espacios en blancos antes o después del nombre, se ignorarán.
- **NMTOKENS**. Igual que la anterior, pero en este caso, hace referencia a una lista de nombres que van separados entre ellos por un espacio.
- **NOTATION**. Nombre de notación que debe estar definido previamente en el DTD.

o **<!ENTITY>**

Elemento bastante avanzado en cuanto al diseño de DTDs. Su declaración hace referencia a un tipo determinado de entidad, entre los que podemos distinguir:

- **Referencia a entidades generales.**

Internas

Sintaxis:

<!ENTITY nombre_entidad definición_entidad>

Por ejemplo:

<!ENTITY rsa "República Sudafricana>

Y, a continuación, podemos anteponer el nombre de la entidad correspondiente para utilizar el XML, anteponiendo el carácter &.

```
<pais>
   <nombre>&rsa;</nombre>
   ...
</pais>
```

Externas

Sintaxis:

<!ENTITY nombre_entidad tipo_uso url_archivo>

Pudiendo seleccionar SYSTEM si es para uso privado o PUBLIC si es público.

Por ejemplo, si tenemos un archivo de texto que contenga los nombres de los alumnos "Ana López y Marcos Ortiz". Se debe crear un documento XML que va a hacer referencia a ese archivo como entidad externa. Cuando se visualice el documento, haremos referencia a la entidad general externa para sustituirla por el texto que contenga el archivo.

```
<!DOCTYPE estudiantes [
   <!ELEMENT estudiantes (#PCDATA)>
   <!ENTITY alumnos SYSTEM "alumnos.txt">
]>
<estudiantes>&autores;</estudiantes>
```

- **Referencias a entidades parámetro.**

Internas

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

Externas.

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

- **Entidades que no están procesadas (unparsed).**

<!ENTITY % nombre_entidad tipo_uso fpi valor_entidad NDATA tipo>

- **<!NOTATION>**

Lo podemos utilizar para señalar el tipo de atributo al que se le permite utilizar algún valor que se haya declarado previamente en el DTD como notación. Mediante la notación podemos determinar un formato de datos distinto a XML como pueden ser: MIME, image/gif, image/jpg, etc.

Sintaxis**:**

<!NOTATION nombre_notacion SYSTEM "identificador_externo">

3.1.2. Validación de documentos XML con esquemas XML

Es un mecanismo que podemos utilizar para hacer una comprobación sobre la validez de un determinado documento XML. Presenta ciertas ventajas entre las que distinguimos:

- Al ser un documento XML podemos comprobar que está bien formado.
- Dispone de un amplio catálogo de tipos de datos predefinidos para los diferentes elementos y atributos.
- Permiten conocer el número de veces que aparece un determinado elemento en un documento XML.
- Ofrece la posibilidad de utilizar varios espacios de nombres.

Aunque presente estas ventajas, no podemos pasar por alto el inconveniente más importante que presentan, que es saber que los esquemas XML son bastante más difíciles de interpretar por las personas que los DTD.

Entre las diferentes herramientas de las que disponemos a la hora de valorar los esquemas, diferenciamos entre las siguientes:

- **Herramientas para validar esquemas**. Las herramientas que ya hemos detallado en apartados anteriores que permiten comprobar si un documento está bien formado, son las mismas que nos pueden determinar si permiten validar un documento que cuente con un esquema XML. Existen una serie de herramientas que permiten llevar a cabo esta función, a continuación, citamos una de ellas.

  http://www.corefiling.com/opensource/schemaValidate.html

- **Herramientas para generar esquemas**. En este caso, como el generador no está capacitado para conocer las distintas reglas de negocio existentes, lo que hace es crear una especie de coraza a partir de un documento XML, que consta de una serie de herramientas como pueden ser:

  Herramientas online:

  http://www.freeformatter.com/xsd-generator.html

  Editores XML que cuentan con esta funcionalidad:

  XMLSPY de Altova

- **Componentes y estructura de esquema XML.**

El esquema XML corresponde a un documento XML que debe llevar a cabo una serie de reglas como:

- El nombre correspondiente al elemento raíz se denomina <schema>.
- Dispone de un espacio de nombres:
  - http://www.w3.org/2001/XMLSchema (linkque sirve únicamente de ejemplo).
  - Aunque es posible no definir ningún prefijo o, por el contrario, utilizar uno de los más comunes, que son xs o xsd.

  Por ejemplo:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

- Como un documento bien formado, dispone de un elemento raíz, como mínimo, cuando hablamos de esquemas XML, vamos a necesitar, al menos, la declaración de ese elemento raíz correspondiente al documento XML.

  Veamos un ejemplo de un XML bastante sencillo:

```
<simple> Esto es un documento sencillo</simple>
```

  Y, a continuación, vamos a escribir un ejemplo de un esquema que lo puede validar, como:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="simple"/>
</xs:schema>
```

- También podemos tener dos elementos raíz declarados: <simple> y <complejo>, por ejemplo:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="simple"/>
        <xs:element name="complejo"/>
</xs:schema>
```

- El orden de los componentes, no afecta al funcionamiento del esquema correspondiente.

- El documento XML debe contener la vinculación al documento XML. Mediante la utilización de xmlns:xsi y xsi:noNamespaceSchemaLocation. Como vemos en el siguiente ejemplo:

```
<simple xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="simple.xsd">
...
</simple>
```

- **Componentes básicos de un esquema**

Para construir un esquema XML haremos uso de una serie de componentes. A continuación, nombramos algunos de ellos que son imprescindibles en este proceso, como:

xs: schema

Referencia URI que utilizamos a la hora de declarar un esquema y su elemento raíz.

- Atributos optativos principales:
  xmlns: permite indicar la cantidad de espacios de nombres que podemos utilizar en un determinado esquema.

- Otros atributos optativos:
  id: utilizado para identificar de forma única un elemento.
  targetNamespace: se refiere al espacio de nombres del esquema.
  elementFormDefault: para referirnos al formato de nombres del esquema.
  attributeFormDefault: para referirnos al formato de los atributos del esquema.
  versión: versión que tiene asignada un esquema determinado.

xs: element

Componente que utilizamos para declarar un elemento y comprobar si existe en el documento XML.

- Atributos optativos principales:
  name: asociado al nombre del elemento. Se considera obligatorio siempre que su padre sea <xs: schema>.
  ref: utilizado para señalar la descripción del elemento que se encuentre en un lugar diferente al esquema. Podemos utilizarlo si su elemento padre es <xs: schema>.

type: tipo del elemento.
default: valor que toma un elemento por defecto siempre que su contenido sea de tipo texto.
fixed: único valor que puede tomar un elemento, lo usaremos siempre que sea un elemento con contenido textual.
minOccurs: mínimo número de apariciones de un elemento en un documento XML.
maxOccurs: máximo número de apariciones de un elemento en un documento XML.

- Otros atributos optativos:
  id: utilizado para identificar de forma única un elemento.
  form: formato que tiene asignado el nombre del atributo.
  substitutionGroup: nombre de aquel elemento que puede sustituir este elemento principal.
  nillable: determina si es posible que aparezca el atributo de instancia.
  xsi: nil asociado a otro elemento del documento de instancia XML.
  abstract: determina si podemos utilizar un documento XML instancia.
  final: determina si el elemento puede derivar de algún otro.

xs: attribute

Componente que utilizamos para declarar un atributo y representar si existe algún atributo de un determinado elemento en un documento XML.

- Atributos optativos principales:
  name: asociado al nombre del elemento.
  ref: utilizado para señalar la descripción del elemento que se encuentre en un lugar diferente al esquema. Podemos utilizarlo si su elemento padre es <xs: schema>.
  type: tipo del elemento.
  use: indica si es obligatorio, opcional, o prohibida la existencia de un atributo.
  default: valor que toma un atributo cuando es procesado.
  fixed: único valor que puede tomar un atributo en un documento XML.
- Otros atributos optativos:
  id: utilizado para identificar de forma única un atributo.
  form: formato que tiene asignado el nombre del atributo.

- **Tipos de datos**

Tanto los diferentes elementos como los atributos de un documento XML pueden ir tomando diferentes valores según los tipos de datos que vayamos asignando. Podemos diferenciar entre los tipos de datos predefinidos frente a los construidos que, a continuación, detallamos:

Tipos de datos predefinidos**:** están organizados de forma jerárquica y podemos diferenciar sobre unos 45 tipos de datos predefinidos diferentes. Cada tipo debe ser de la misma forma que sea el tipo de su padre.

- **Definición del tipo predefinido especial:**

xs: anyType

Lo encontramos en la jerarquía de tipos y, a partir de él, derivan todos los demás.

- **Definición de otro tipo predefinido especial:**

xs: anySimpleType

Puede representar a cualquier tipo simple sin llegar a particularizar.

Los tipos predefinidos se dividen a su vez en:

- **Primitivos.** Los que derivan de xs: anySimpleType
- **No primitivos**. Los que derivan de algún tipo primitivo

Estos tipos predefinidos se agrupan por distintas categorías, como son:

Numéricos:

Tipo de datos	Descripción
Float	Número en punto flotante de precisión simple (32 bits).
Double	Número en punto flotante de precisión doble (64 bits).
Decimal	Números reales que pueden ser representados como $i \cdot 10^{-n}$
Integer	Números enteros, de – infinito a infinito.
nonPositiveInteger	Números enteros negativos más el 0.
negativeInteger	Números enteros menores que 0.
nonNegativeInteger	Números enteros positivos más el 0.

positiveInteger	Números enteros mayores que 0.
unsignedLong	Números enteros positivos desde 0 hasta 18.446.744.073.709.551.615 (64 bits de representación: 2^{64} combinaciones.
unsignedInt	Números enteros positivos desde 0 hasta 4.294.967.295 (32 bits de representación: 2^{32} combinaciones).
unsignedShort	Números enteros positivos desde 65.535 (16 bits de representación: 2^{16} combinaciones).
unsignedByte	Números enteros positivos desde 0 hasta 255 (8 bits de representación: 2^{8} combinaciones).
Long	Números enteros representados con 64 bits: 2^{64} combinaciones, desde -9.223.372.036.854.775.808 hasta 9.223.372.036.854.775.807.
Int	Números enteros representados con 32 bits: 2^{32} combinaciones, desde -2.147.483.648 hasta 2.147.483.647.
short	Números enteros representados con 16 bits: 2^{16} combinaciones, desde -32.768 hasta 32.767.
byte	Números enteros representados con 8 bits: 2^{8} combinaciones, desde -128 hasta 127.

De fecha y hora

Tipo de datos	Descripción
duration	Duración en años + meses + días + horas + minutos + segundos
dateTime	Fecha y hora, en formato aaaa-mm-dd T hh:mm:ss
date	Solamente fecha en formato aaaa-mm-dd
time	Solamente la hora, en formato hh:mm:ss
gDay	Solo el día, en formato –dd (la g viene de calendario Gregoriano).
gMonth	Solo el mes.
gYear	Solo el año, en formato yyyy.
gYearMonth	Solo el año y el mes, en formato yyyy-mm.
gMonthDay	Solo el mes y el día, en formato –mm-dd.

De texto

Tipo de datos	Descripción
string	Cadenas de texto.
normalizedString	Cadenas de texto en las que se convierten los caracteres tabulador, nueva línea, y retorno de carro en espacios simples.
token	Cadenas de texto sin los caracteres tabulador, ni nueva línea, ni retorno de carro, sin espacios por delante o por detrás, y con espacios simples en su interior.
language	Valores válidos para xml:lang (según la especificación XML).
NMTOKEN	Tipo de datos para atributo según XML 1.0, compatible con DTD.
NMTOKENS	Lista separada por espacios de NMTOKEN, compatible con DTD.
Name	Tipo de nombre según XML 1.0.
QName	Nombre cualificado de espacio de nombres XML.
NCName	QName sin el prefijo ni los dos puntos.
anyURI	Cualquier URI.
ID	Tipo de datos para atributo según XML 1.0, compatible con DTD. Tienen que ser valores únicos en el documento XML.
IDREF	Tipo de datos para atributo según XML 1.0, compatible con DTD.
IDREFS	Lista separada por espacios de IDREF, compatible con DTD.
ENTITY	Tipo de datos para atributo XML 1.0, compatible con DTD.
ENTITIES	Lista separada por espacios de ENTITY, compatible con DTD.
NOTATION	Tipo de datos para atributos en XML 1.0, compatible con DTD.

Binarios

Tipo de datos	Descripción
hexBinary	Secuencia de dígitos hexadecimales (0…9, A, B, C, D, E, F).
Base64Binary	Secuencia de dígitos en base 64.

Booleanos

Tipo de datos	Descripción
boolean	Puede tener 4 valores: 0, 1, true y false.

Los distintos tipos de datos, también podemos dividirlos en simples o complejos.

- **Tipos de datos simples**
  - Son simples y tienen valores atómicos todos los tipos de datos predefinidos
  - Se construyen a partir de un tipo base predefinido.
  - Se asignan a aquellos elementos que solo tienen carácter textual.

Definición

<xs: simpleType>

Permiten limitar el rango de valores mediante <xs: restriction>.

xs: simpleType

Define los tipos simples.

- Atributos optativos principales:
  name: corresponde al nombre del tipo. Es de carácter obligatorio siempre que el componente sea hijo de <xs:schema> Si no es así, no está permitido.
  id: único identificador para el componente.

xs: restriction

Permite definir aquellas restricciones correspondientes a los valores de un tipo determinado.

- Atributos obligatorios.
  base: nombre asociado al tipo a partir del que se va a empezar a construir un nuevo tipo (predefinido o construido).
- Atributos optativos principales.
  id: único identificador para el componente.

Faceta para restringir el rango de valores:

Faceta	Uso	Tipos de datos para donde se usa
xs:minInclusive	Especifica el límite inferior del rango de valores aceptable. El propio valor está incluido.	Numéricos y de fecha/horas.
xs:maxInclusive	Especifica el límite superior del rango de valores aceptable. El propio valor está incluido.	Numéricos y de fecha/horas.
xs:minExclusive	Especifica el límite inferior del rango de valores aceptable. El propio vaolr no está incluido.	Numéricos y de fecha/horas.
xs:maxExclusive	Especifica el límite superior del rango de valores aceptable. El propio valor no está incluido.	Numéricos y de fecha/horas.
xs:enumeration	Especifica una lista de valores aceptable.	Todos
xs:pattern	Especifica un patrón o expresión regular que deben cumplir los valores válidos.	Texto
xs:whiteSpace	Especifica cómo se tratan los espacios en blanco, entendiéndose como tales los saltos de línea, tabuladores y los propios espacios. Sus posibles valores son preserve, replace y collapse.	Texto
xs:length	Especifica el número exacto de caracteres (o elementos de una lista) permitidos. Tiene que ser mayor o igual que o.	Texto
xs:minLength	Especifica el mínimo número de caracteres (o elementos de una lista) permitidos. Tiene que ser mayor o igual que o.	Texto
xs:maxLength	Especifica el máximo de caracteres (o elementos de una lista) permitidos. Tiene que ser mayor o igual que o.	Texto
xs:fractionDigits	Especifica el número máximo de posiciones decimales permitidas en números reales. Tiene que ser mayor o igual que o.	Números con parte decimal
xs:totalDigits	Especifica el número exacto de dígitos permitidos en números. Tiene que ser mayor que o.	Numéricos

- **Tipos de datos complejos**
  - Se asignan a aquellos elementos que poseen elementos o atributos descendientes.
  - Puede contener contenido complejo por lo que tiene elementos descendientes.
  - Puede estar vacío, es decir, que no tiene contenido, aunque sí atributos.
  - Puede tener contenido mixto, que significa, mezcla de contenido textual con elementos descendientes.

Definición:

<xs: complexType>

Permiten asignar elementos.

El contenido que puede tener asignado un elemento va entre las etiquetas de apertura y cierre, y diferencia entre:

- **Contenido simple**. El elemento solo contiene elementos que posean contenido textual. Sin elementos descendientes.

  (xs: simpleContent).

- **Contenido complejo.** El elemento puede poseer contenido textual o no. Tiene elementos descendientes.

  (xs: complexContent)

- **Distintos modelos de contenido**

Podemos diferenciar entre los siguientes modelos de contenido

- **Secuencia**. Los elementos se presentan en filas, uno detrás de otro siguiendo un orden determinado. Se declara:
  <xs: sequence>
- **Alternativa**. Los elementos aparecen como una alternativa los unos de los otros. Solo se puede elegir uno. Se declara:
  <xs: choice>
- **Todos**. Los elementos pueden aparecer en un orden cualquiera. Se declara:
  <xs: all>

- **xs: complexType**

Componente utilizado a la hora de definir los tipos de datos complejos.

- Atributos optativos principales:
  name: corresponde al nombre del tipo. Es de carácter obligatorio siempre que el componente sea hijo de <xs: schema> Si no es así, no está permitido.
  id: único identificador para el componente.
  mixed: mezcla contenida de texto con otros elementos descendientes.
  abstract: señala si es posible utilizar de forma directa un tipo de elemento.
  final: señale si el tipo es derivable según la restricción, extensión o por ambas.

3.2. Lenguajes basados en XML

Hay una serie de lenguajes en XML que cuentan con un propósito específico con un mecanismo de validación correspondiente, como pueden ser:

- **SVG** (Scalable Vector Graphics, Gráficos Vectoriales Escalables). Es un lenguaje utilizado para representar una serie de gráficos vectoriales bidimensionales.
- **WML** (Wireless Mark-up Language, Lenguaje de Marcado Inalámbrico). Es un lenguaje mediante el cual podemos representar la información en dispositivos móviles.
- **RSS** (Really Simple Syndication, Sindicación Realmente Simple). Se refiere a la familia de los diferentes formatos web que podemos utilizar para publicar información que va a sufrir constantes actualizaciones.
- **Atom** (Atom Syndication Format, Formato Atómico de Sindicación). Lenguaje que se utiliza como semillas web.
- **DocBool**. Lenguaje utilizado para definir documentos.
- **XBRL** (*Extensible Business Reporting Language*, Lenguaje Extensible de Informas Financieros). Lenguaje estándar abierto que permite representar la información junto con la expresión semántica necesaria para los distintos informes financieros.

UF2. Ámbitos de aplicación de XML

1. Aplicación de los lenguajes de marcas a la sindicación de contenidos

RSS (*Really Simple Syndication*, Sindicación Realmente Simple), son las siglas correspondientes a una tecnología que tiene como función principal, compartir o distribuir la información que se encuentre en la web.

Es conveniente que, al principio, se haga uso de un software específico que esté diseñado para leer contenidos RSS (*feed reader*).

La información que se encuentra disponible en un sitio web, se puede compartir de varias formas diferentes:

- Una bastante sencilla es suscribiendo la fuente con un agregador de noticias.
- Otra, si compartimos aquella información insertada en otros lugares web, así, la persona encargada de recibir las noticias, actúa también como emisor. De esta forma, existe un proceso al que denominamos redifusión web.

Por tanto, RSS hace referencia a una técnica que podemos utilizar para conseguir extender aquellos contenidos que se encuentran en un sitio web. Es un lenguaje derivado de XML mediante el cual podemos construir los diferentes archivos que almacenan el contenido que se puede difundir.

Es necesario, que señalemos también que existe otro programa que realiza las mismas funciones, como Atom, que deriva también de XML, aunque añade una serie de ventajas. Estos dos lenguajes se utilizan para guardar diferentes contenidos que se deseen distribuir. Son distintos, ya que el primero (RSS) se refiere a la redifusión web mientras que el segundo, hace referencia a un archivo determinado que posee la información que se desea difundir.

Entre sus ventajas principales, destacamos las siguientes:

- Aquellos usuarios que utilicen este programa no necesitan hacer una comprobación para saber si la información está actualizada en los lugares en los que se encuentran los contenidos de interés.
- Sigue un formato para los datos de tipo texto plano. Presenta bastante rapidez a la hora de transmitir los datos por lo que es bastante recomendable para los dispositivos móviles.
- Ofrece la posibilidad de poder filtrar información de diferentes sitios sin perder tiempo utilizando la agrupación.

1.1. Ámbitos de aplicación

La sindicación o redifusión web no está relacionado con los web blogs. Es un repositorio de contenidos compartiéndolo con otros usuarios o todo tipo de información XML. De esta forma podemos ofrecer contenidos propios para que sean mostrados en otras páginas web de forma integrada. Esto hace que la página origen también se vea afectada positivamente. Todo tipo de datos se puede realizar la redifusión de contenidos web. Para permitir la visualización de los contenidos debe suscribirte en el enlace web correspondiente. Desde el punto de vista de los suscriptores, la redifusión de contenidos permite, entre otras cosas, la actualización profesional.

1.2. Estructura de los canales de contenidos

Siempre comienza con el siguiente prólogo:

<?xml versión="1.0" encoding="utf-8"?>

<rss versión="2.0">

La primera línea hace referencia a la declaración XML, que se encarga de definir la versión seleccionada para trabajar con XML junto con la codificación de caracteres utilizados. En este ejemplo, la versión seleccionada es la 1.0 de XML que utiliza los caracteres Unicode.

La segunda línea es la declaración de tipo de documento, que nos permite identificar el lenguaje que deriva de XML, en el ejemplo, RSS versión 2.0.

El contenido del código debe encontrarse entre las etiquetas: *<rss>contenido</rss>*

Y, a continuación, viene seguido del prólogo del documento en cuestión, que vendrá identificado entre:

<channel>prólogo</channel>

Este elemento <channel> puede contener uno o más elementos <item>.

```
<?xml versión="1.0" encoding="utf-8"?>
<rss versión="2.0">
   <channel>
      <tittle>Hola mundo</tittle>
      <link>...</link>
      <description>...</description>
      <item>
      ...
      </item>
   </channel>
</rss>
```

1.3. Elementos principales de un RSS

1.3.1. <channel>

Su función principal es describir la fuente RSS mediante tres elementos obligatorios, que son:

- **<tittle>**. Hace referencia al nombre del canal.
- **<link>**. Define el hipervínculo al canal.
- **<description>**. Se encarga de describir el contenido correspondiente al canal.

Este elemento **<channel>** tiene posibilidad de contener un elemento o varios **<item>.**

Esta lista que contiene aquellos elementos opcionales en <channel>:

Elemento	Descripción
<category>	Define una o más cateogrías para el canal.
<cloud>	Permite información inmediata 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 cuando los agregadores muestren un canal.
<Language>	Especifica el idioma en que está escrito en el canal.
<lastBuildDate>	Define la fecha de la última modificación del contenido.
<link>	Define el hipervínculo del 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.

1.3.2. <item>

Cada elemento <item> cuenta con la posibilidad de poder definir un determinado artículo en el canal RSS. En este caso, también debe contar con tres elementos fundamentales, como:

- **<tittle>**. Hace referencia al título del artículo.
- **<link>.** Define el hipervínculo al artículo.
- **<description>.** Se encarga de describir el canal.

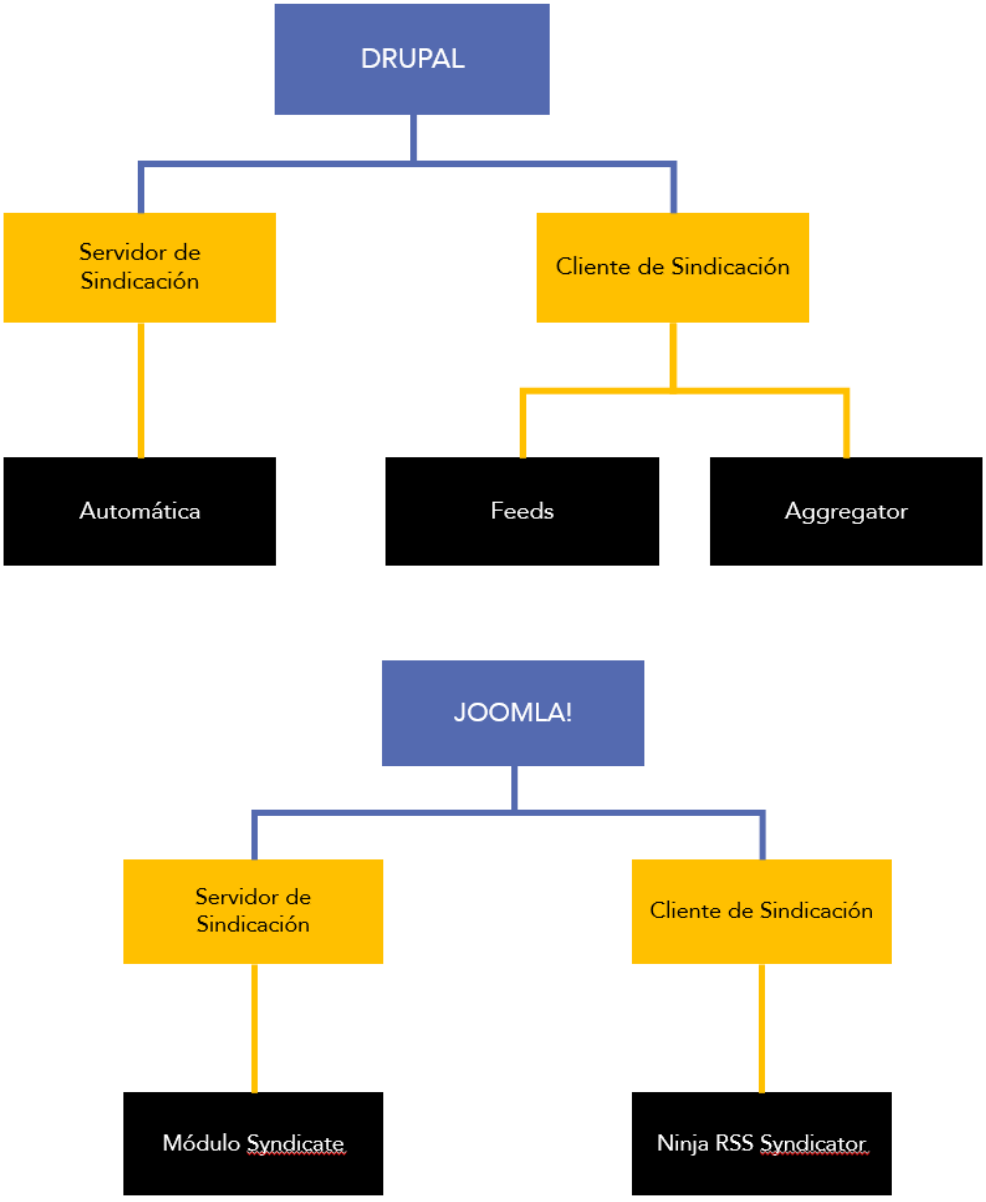
Los elementos opcionales más importantes de <item> son los siguientes:

Elemento	Descripción
<author>	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 artículo mediante un link.

1.4. Tecnologías de creación de canales de contenidos

Para la creación de un canal de contenido, lo único que hace falta es un editor de XML y página web. Para facilitar la creación de páginas de sindicación podemos utilizar las herramientas de Joomla o DRupal. Estos programas que permiten crear una estructura de soporte para la creación y administración de contenidos.

Presentan la ventaja de ser de código abierto y con implementación mayoritariamente en php, utiliza también servidores web Apache y gestores de bases de datos Mysql.



1.5. Validación y publicación

- **Validación del archivo RSS**

Después de haber creado un archivo RSS, ya podemos realizar la validación de dicho archivo mediante la herramienta de validación del **W3C**.

En la siguiente dirección podemos validar los distintos archivos desarrollados en RSS o Atom:

https://validator.w3.org/feed/

También contamos con la posibilidad de seleccionar entre las opciones:

- Validación por URI, si hemos conseguido subir el archivo al servidor.
- Validación pegando el archivo directamente.

- **Publicación del archivo RSS**

Una vez que ya hemos escrito el código y hemos realizado su validación correspondiente, debemos subirlo a un servidor web. Para conseguirlo, debemos seguir una serie de pasos, que definimos a continuación:

- Primero insertaremos el enlace en la página de inicio y debemos conseguir que apunte al archivo RSS. De esta forma, podremos consultarlo mediante el navegador.
- Seguidamente, debemos insertar un elemento <link> para que los lectores RSS encuentren el archivo RSS para leerlo. Se suscriben a nuestro feed.
- A continuación, enviamos la dirección URI correspondiente desde el archivo RSS hasta el sitio web determinado. Se catalogan y almacenan feeds para que los diferentes buscadores los puedan visitar.

1.6. Directorios de canales de contenidos

Existen una serie de canales de contenidos de donde podemos coger la información que deseemos. Estos canales están catalogados por temática. Por tanto, podemos suscribirnos al canal deseado, de esta forma se actualizará el contenido integrado en mi portal. Todo esto es gracias al lenguaje XML.

El geoportal de Infraestructuras de Datos Espaciales de España, es un listado de canales de Rss para suscribirnos a los que nos interesen.

http://www.idee.es/suscripcion-canales-rss

1.7. Agregación

Es necesario que el usuario disponga de un agregador. Podemos diferenciar entre una gran variedad de lectores RSS, que vamos a clasificar en tres categorías distintas, que son:

- **Agregadores de escritorio.** Aquellas que se pueden instalar en el ordenador de un usuario.
- **Agregadores en línea.** No es necesario que el usuario las instale. Solo es necesario darse de alta en el sitio para utilizarlo.
- **Agregadores como plugins.** Existen algunos navegadores, como FireFOX, Nestcape, Opera y algunos más, que lo incluyen entre sus programas para ofrecer un servicio al usuario.

Cuando el usuario ya ha seleccionado el agregador adecuado, tiene que seleccionar, a continuación, qué feeds o archivos RSS va a necesitar para poder llevar a cabo la correspondiente sindicación de contenidos.

- **Cómo utilizar RSS para ofrecer información**

Para poder ofrecer una determinada información desde nuestro sitio web, es necesario un creador de contenidos que cuente con conocimientos sobre XML.

El código necesario a la hora de crear un feed (documento RSS) debe contener la información necesaria referente al sitio web. Esta información no varía, aunque, sus datos, se irán actualizando cada cierto período de tiempo.

Por tanto, estos datos, deben estar ordenados mediante las etiquetas de inicio y fin según el lenguaje XML. De esta forma, iremos creando nuestro feed propio que puede tener distintos ítems.

Cuando terminemos de crear el archivo RSS, debemos validarlo y comprobar que funciona correctamente. De esta forma, iremos registrando diferentes agregadores y así podremos comprobar cuántos usuarios están interesados por la información que ofrecemos a través del feed.

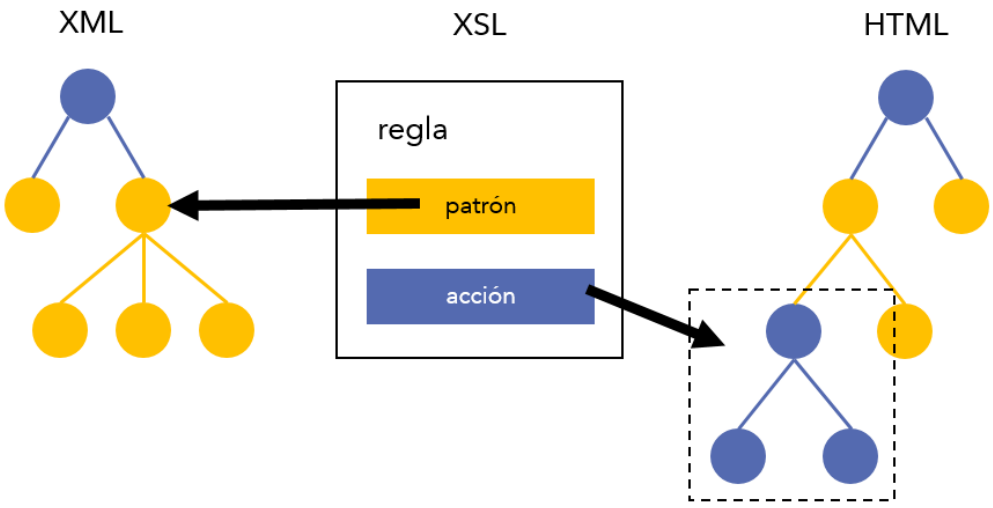
2. Conversión y adaptación de documentos XML

Existen una serie de lenguajes, basados en XML, como es XSL (*eXtensible Stylesheet Language*) que consta de las siguientes partes:

- **XSLT**. Este lenguaje, puede transformar aquellos documentos XML a diferentes formatos.
- **XPtah**. Es un tipo de lenguaje funcional que puede tener acceso a determinadas secciones de un documento XML.
- **XSL- FO**. Hace referencia al vocabulario XML para poder determinar las distintas semánticas de formateo.

- **XSLT (*eXtensible Stylesheet Language for transformations*)**

Es un lenguaje estándar de W3C que se recomienda por primera vez en 1999. Basado en XML, por lo que, podemos decir que, cada hoja de transformaciones es un documento XML bien formado.



Se trata de un lenguaje de programación (declarativo) que es capaz de generar documentos en diferentes formatos, como pueden ser: XML, HTML, texto, PDF, etc. partiendo de un documento XML.

Lo definimos como un lenguaje declarativo porque está basado en definir una serie de reglas o plantillas que deben ser aplicadas a un documento XML para conseguir transformarlo en la salida seleccionada.

Estas reglas, que suelen tener extensión .xls, unidas al documento .xml, se van a pasar como parámetros a un procesador XSLT que será el encargado de generar (como salida) el nuevo documento ya transformado.

2.1. Técnicas de transformación de documentos XML

Hojas de estilos: CSS vs XSLT:

Existen dos familias diferentes de normas definidas por W3C: CSS y XSLT.

- **CSS**

Ofrece la posibilidad de definir las propiedades asociadas a los diferentes elementos de marcado (letra negrita, de un tamaño determinado, con o sin bordes, etc.), aunque presenta una serie de inconvenientes como:

- No puede cambiar orden de los elementos pertenecientes a un documento HTML.
- No tiene permitido llevar a cabo distintas operaciones con los elementos.
- Y los elementos no se pueden combinar entre ellos.

- **XSLT**

Supera los inconvenientes anteriores ya que puede determinar el contenido que tiene un documento XML, y elegir el orden en el que se va a mostrar. También cuenta con la posibilidad de llevar a cabo las distintas operaciones que sean necesarias.

Es posible decantarse por una de las dos o, incluso tenemos la opción de utilizar CSS y complementarlo con XSLT

- **Procesador XSLT**

Formado por una serie de aplicaciones que permiten procesar aquellos documentos XML mediante las hojas de transformaciones XSLT.

- Primero, el procesador XSLT lee un documento XML
- A continuación, puede representar el documento mediante un árbol de nodos. Estos nodos pueden ser: elementos, atributos, texto, comentarios, instrucciones o espacios de nombres. Los nodos actúan de la misma forma que un árbol (padres, hijos, ascendientes, etc.).
  - El procesador XSLT recorre y procesa nodo a nodo. El nodo que se está tratando en un instante determinado, se denomina nodo de contexto

- **Proceso de transformación**

Desde XML podemos llevar a cabo las transformaciones a:

- Texto
- XML
- HTML

Para llevar a cabo este proceso de transformación, debemos disponer del XML de partida junto con la hoja de estilos XLST.

Podemos llevarlo a cabo desde tres sitios diferentes:

- **En el servidor web**. Utiliza una serie de lenguajes de servidor como pueden ser: PHP, JSP, etc. para aplicar la XSLT al XML, de tal forma que permite enviar de vuelta el HTML.
- **En el cliente web**. En este caso, es el cliente el encargado de realizar las transformaciones necesarias mediante determinados programas, como puede ser JavaScript que cuenta con una serie de funcionalidades como pueden ser: tratamiento de documentos XML, transformaciones XSLT, etc.
- **En una aplicación diferente**. Permite utilizar distintos programas independientes para llevar a cabo este proceso de transformación.

- **Pasos para la transformación**
  - Debe analizar la hoja de instrucciones pasando a convertirse en una estructura árbol.
  - Debe procesar el documento XML y convertirlo en una estructura árbol.
  - El procesador XSLT debe posicionarse en la raíz del documento XML
  - Aquellos elementos que no formen parte del espacio de nombres (con prefijo xsl), se pasan a la cadena de salida sin que haya existido modificación alguna.
  - El procesador XSLT solo puede llevar a cabo una única regla a cada nodo.

2.2. Descripción y utilización de estructura, sintaxis y plantillas

- **Estructura básica de una hoja de transformaciones**

Una hoja de transformaciones debe ir compuesta por lo siguiente:

- Declaración del documento XML.

  <?xml versión="1.0?">
- Elemento raíz.

  <xsl:stylesheet versión="1.0"
  xmlns= "http://www.w3.org/1999/XSL/Transform"
  …
  /xsl:stylesheet>
- Espacio de nombres.

  http://www.w3.org/1999/XSL/Transform, donde xsl corresponde al prefijo.
- Otra serie de elementos que denominamos de nivel superior que, es conveniente que aparezcan en la hoja de transformaciones, siempre como hijos de <xsl:stylesheet>.

- **Algunas transformaciones especiales**

A continuación, vamos a ver una serie de casos en los que es conveniente hacer uso de diferentes trasformaciones especiales, como:

- Si tenemos una hoja de transformación sin plantillas (template), el contenido de texto del documento XML se va a enviar directamente a la salida, sin tener en cuenta los distintos valores de los atributos, ni los comentarios ni las instrucciones específicas de procesamiento.
- Si la hoja de transformación solo tiene una plantilla, la asocia al nodo raíz del documento XML sin contenido alguno, es decir, no va a enviar nada a la salida.
- Si tenemos una hoja de transformaciones, pero esta no tiene asociada ninguna plantilla al nodo raíz, aunque, sí cuenta con otras que pueden ir asociadas a otros elementos, se va a ir realizando un recorrido por el árbol del documento XML. De esta forma, puede enviar contenido textual a la salida y aplicar las reglas que se hayan definido en cuanto se localice el elemento con la plantilla asociada.
- Si tenemos una hoja de transformaciones con alguna plantilla, se va a ir realizando un recorrido por el árbol del documento XML en el mismo orden en el que está escrito y, cuando encuentre una plantilla que coincida con el elemento que nos encontramos, se aplicará a la plantilla.

- **Elementos básicos de una hoja de transformaciones**

Existen un grupo de instrucciones que nos van a permitir poder generar un gran número de salidas, entre los que destacamos:

xsl:stylesheet y xsl:transform

Son unos elementos que podemos utilizar para poder definir el elemento raíz en la hoja de transformaciones.

- Atributos obligatorios
  versión: determina la versión seleccionada XSLT del documento.
  xmlns: hace referencia al espacio de nombres del documento XML.
- Atributos optativos principales
  exclude-result-prefixes: lista los espacios de nombres separados por un espacio. Estos no se deben enviar a la salida.

xsl:output

Es el elemento encargado de definir el formato del documento de salida. Este elemento corresponde a un nivel superior y tiene que aparecer como "hijo de".

<xsl:stylesheet> o <xsl: transform>

- Atributos optativos principales
  method: tiene como valor por defecto, XML y va a definir el formato de salida.
  versión: determina la versión del formato de salida.
  encoding: apunta al juego de caracteres de salida.
  indent: determina si la salida está preparada conforme a la estructura jerárquica.
  omit-xml-declaration: determina si la instrucción declarada para procesar el documento se puede omitir en la salida.
  standalone: determina si es necesario añadir el atributo standalone a la instrucción que declara el tipo. Su valor asignado por defecto es, no.

xsl:template

Es una de las instrucciones más importantes de las hojas de transformaciones, junto con <xsl:apply-templates>.

Puede representar una plantilla que dispone de una serie de acciones encargadas de realizar el patrón de una plantilla.

Si aplicamos una plantilla a un nodo determinado, se va a aplicar solo a ese nodo, pero lo que hace es sustituir al nodo y a sus descendientes por el resultado que hemos obtenido de la aplicación de la plantilla. Este hecho nos va a eliminar toda la información de los descendientes.

- Atributos optativos principales
  name: asigna un nombre a una plantilla.
  match: señala el patrónXPath al que se le va a aplicar la plantilla.
  priority: valor que oscila entre -9 y 9 (de menos a más prioridad) que le asignamos a las diferentes reglas de resolución de conflictos.
  mode: diferencia entre dos plantillas con el mismo atributo match.

xsl:apply-templates

Es otra de las funciones principales en el ámbito del lenguaje de las transformaciones XSLT. Si comprendemos su funcionamiento sin ningún tipo de problema, vamos a conseguir sacar el máximo rendimiento a la hora de trabajar con las distintas transformaciones.

- Atributos optativos principales
  select: selecciona los nodos que se van a procesar.
  mode: diferencia entre distintas plantillas que tengan el mismo atributo match.

- **Reglas de resolución de conflictos en plantillas**

Existen unas reglas que se deben aplicar a los elementos siguiendo un orden. Cada una de estas reglas, va a tener una prioridad asociada (-9,9) que, cuando las aplicamos de forma automática, va a tener los valores:

Patrón	Prioridad
"*", "@*", "text ()", y otros patrones para los que no se indica un nombre concreto.	-0.5
"Prefijo:*", "@prefijo:*", y otros patrones para los que se indica el espacio de nombres, pero no un nombre concreto.	-0.25
"producto", "@unidades", y otros patrones donde se indiquen nombres de elementos y atributos concretos.	0
"proveedor/representante", "productos/producto/precio", "precio/@unidades", y otros patrones para los que se indique una jerarquía (además del nombre del elemento y/o atributo.	0.5

En caso de existir diferentes plantillas con la misma prioridad, puede que se produzca un error o puede que se aplique a la última plantilla.

- **Modos de las plantillas**

Es bastante frecuente que una hoja de estilo acceda distintas veces a un mismo nodo, devolviendo valores diferentes a la salida cada vez. Para estas ocasiones, debemos utilizar el atributo de las plantillas mode, que nos permite etiquetar cada plantilla para que se puedan llamar por el texto de la etiqueta.

xsl:call-template. Podemos utilizarla para llamar a una plantilla por su nombre. Cuando la aplicamos, no cambia el nodo de contexto, por tanto, en la plantilla que invoquemos tenemos que utilizar el mismo direccionamiento de aquellos elementos que existían en la que realizaba la llamada.

- Atributos obligatorios
  name: nombre de la plantilla a la que se va a llamar.

xsl:value-of. Nos permite visualizar el contenido que se indica en el atributo select junto con todos sus descendientes.

- Atributos obligatorios
  select: expresión XPath que determina el elemento o atributo del que debemos extraer el contenido.
- Atributos optativos principales
  Disable-output-escaping: indica si algunos caracteres especiales (<,&) van a ser enviados a la salida tal cual o, por el contrario, en su forma de entidad (&lt, &amp).

- **Versión simplificada para mostrar valores de atributos**

Solo es posible utilizar esta versión simplificada en determinados casos, como cuando se genera un nuevo elemento (etiqueta) en el documento de salida, y deseamos asignar algún valor a alguno de sus atributos.

xsl:text. Permite la posibilidad de poder escribir un texto de forma literal en la salida.

- Atributos optativos principales:
  Disable-output-escaping: indica si algunos caracteres especiales (<,&) van a ser enviados a la salida tal cual o, por el contrario, en su forma de entidad (&lt, &amp).

- **Instrucciones de control**

Disponemos de un conjunto de etiquetas que ofrecen la posibilidad de reproducir el funcionamiento de aquellas instrucciones de control de flujo correspondientes a los lenguajes imperativos.

xsl:for-each. Permite repetir una lista de elementos para poder realizar diferentes transformaciones sobre ellos.

- Atributos obligatorios
  Select: expresión XPath que determina la lista de elementos que se van a repetir.

xsl:sort. Devuelve los datos de un documento XML ordenados por algún método.

- Atributos optativos principales
  select: expresión XPath que determina el nodo o grupo de nodos que van a constituir el método de ordenación.
  lang: permite seleccionar el idioma que vamos a llevar a cabo al ordenar.
  data-type: tipo de aquellos datos que se van a ordenar.
  order: indica si los elementos se van a ordenar de forma ascendente o descendente (*ascending, descending*)
  case-order: determina el orden de letras mayúsculas (*upper-first*) o minúsculas (*lower-first*).

xsl:if. Utilizado cuando existe comportamiento condicional. Realiza una pregunta por una condición y, si esa condición es cierta va a realizar una cosa. En caso de que sea falso, realizará otra.

- Atributos obligatorios
  test: expresión XPath que, en caso de ser cierta, se puede aplicar a una plantilla.

xsl:choose,xsl:when, xsl:otherwise. Ofrecen la posibilidad de llevar a cabo un comportamiento condicional con distintas opciones, además de otra por defecto.

2.3. Elaboración de documentación.

- **Elaboración de documentos XML con hojas de estilo**

Las hojas de estilos CSS son los archivos destinados a dar formato a las páginas web. Normalmente se enlazan a los archivos HTML, aunque también pude hacer referencia a un documento XML y visualizarse en el navegador.

El W3C aprobó en junio de 1999 la recomendación que definen cómo vincular documentos XML con hojas de estilo CSS. De acuerdo con esta recomendación mediante la instrucción <?xml-stylesheet ?> de forma similar a como se hace en una página web XHTML con la etiqueta <link />. En ambos casos el atributo href incluye el camino absoluto o relativo a la hoja de estilo CSS. va al principio del documento, después de la declaración XML, como muestra el siguiente ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="ejemplo.css"?>
<persona>
  <nombre>Juan</nombre>
  <localidad>Lleida</localidad>
</persona>
```

- **Generación de nuevos elementos y atributos**

Cuando la salida que se genera está en formato XML, podemos elegir la opción de enviarlos a la salida elementos/atributos que ya existen en el documento XML de entrada, o podemos también tener la posibilidad de volver a crearlos.

xsl:element. Permiten la posibilidad de poder generar un elemento nuevo al documento de salida.

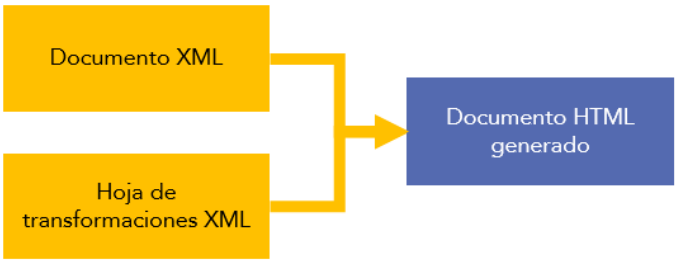
- Atributos obligatorios
  name: determina el nombre del elemento.
- Atributos optativos principales
  namespace: permite definir el URI correspondiente al espacio de nombres del atributo.
  use-attribute-sets: muestra una lista, separada por espacios de conjuntos de atributos, que van a contener aquellos atributos que pueden incluir al elemento.

xs:attribute. Puede producir un atributo nuevo de un determinado elemento.

- Atributos obligatorios
  name: determina el nombre del atributo.
- Atributos optativos principales
  namespace: permite definir el URI correspondiente al espacio de nombres del atributo.

xsl:comment. Crea un comentario en el documento de salida, que puede ser de texto literal o, un dato que se ha extraído del documento XML inicial.

xsl:processing-instruction. Ofrece la posibilidad de crear distintas instrucciones de procesamiento sobre el documento de salida.



xsl:copy-of. Permite realizar una copia exacta de un elemento a la salida. Tanto atributos como aquellos elementos descendientes, también se copiarán.

xsl:copy. Permite realizar la copia de un determinado nodo sin hacerla de sus descendientes.

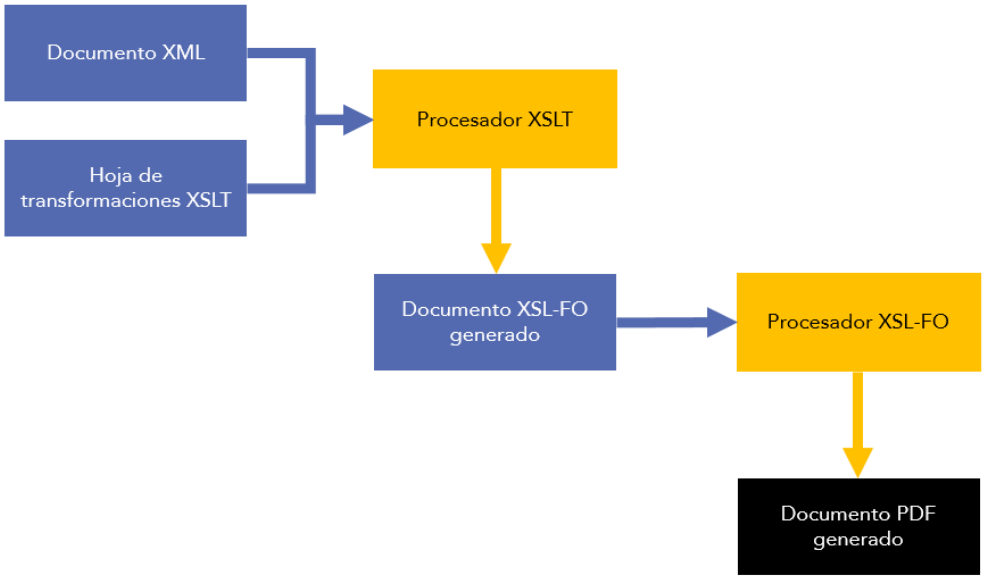
- Atributos optativos principales
  use-attribute-sets: muestra una lista con los diferentes conjuntos de atributos, separados por un espacio, que se pueden aplicar al nodo de salida.

2.4. XSL- FO

Como ya indicamos al comienzo de esta unidad formativa, **XSL-FO** (*XSL- Formatting Objects*, Objetos de formateo XSL) es el lenguaje que se encarga de determinar el aspecto que van a tener los distintos datos a la hora de mostrarlos en un formato determinado de salida, que suele ser: PDF, RTF, PostScript, etc.

- **Procesadores XSL- FO**

Hacen referencia a una serie de programas que recibe un documento XSL- FO de entrada para generar otro de salida, con diferente formato.



- **FOP (*Formatting Objects Procesor*- Procesador de Objetos de Formateo)**

Este procesador es una aplicación (Java) dentro del código de Apache que, entre sus tareas principales, se encarga de coger un documento XSL-FO de entrada para poder generar una salida en un formato diferente, por ejemplo PDF.

Los pasos de sintaxis son:

– Inicialmente, vamos a utilizar un documento FO de entrada para generar otro en formato PDF con la siguiente sintaxis:

Fop -fo <doc_inicial.fo> -pdf <doc_final.pdf>

- A continuación, podemos generar un documento de salida con formato PDF. Recordemos que el documento inicial está en formato FO y este se construye a partir de un documento XML con una hoja de transformacions XSLT. En este caso, seguiremos la sintaxis:

  fop -xml <doc_inicial.xml> -xls <doc_inicial.xsl>
  -pdf <doc_final.pdf>

También existe la posibilidad de realizar la salida mediante los dos pasos que indicamos a continuación:

1. Generamos el documento con formato FO a partir de XML y una XSLT.

fop -xml <doc_inicial.xml> -xsl <doc_inicial.xsl>

-foout <doc_final.fo>

2. A continuación, generamos el documento que se va a poder visualizar (en formato pdf) a partir del documento FO.

fop -fo <doc_final_paso_1.fo> -pdf <doc_final.pdf>

- **Objetos de formateo**

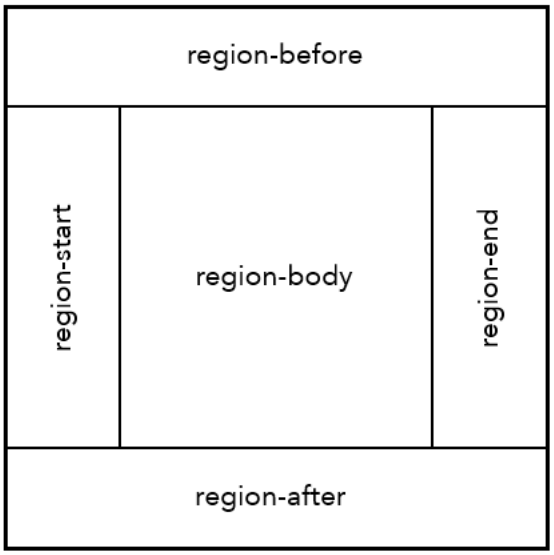
Las etiquetas que se utilizan en este lenguaje están siempre relacionadas con distintos elementos de maquetado como páginas, párrafos, bloques y tablas.

En este caso, XSL-FO utiliza distintas áreas rectangulares para visualizar la salida.

Entre sus áreas más importantes, destacamos:

- **Páginas**. Cuando la salida de una trasformación se organiza mediante páginas que, a su vez, están compuestas de regiones.
- **Regiones**. Contiene una serie de regiones como:

  region-body (cuerpo de página)
  region-before (cabecera de la página)
  region-after (pie de página)
  region-start (lateral izquierdo)
  region-end (lateral derecho)



Las diferentes regiones pueden contener áreas de bloque.

- **Áreas de bloque**. Permiten definir elementos de bloques no muy grandes, como párrafos, líneas o incluso tablas. La mayoría de las áreas de bloque contienen áreas de línea.
- **Áreas de línea**. Permiten definir, dentro de las áreas de bloque, un texto. Las áreas de línea pueden contener áreas secuenciales.
- **Áreas secuenciales**. Permiten definir algún texto dentro de las áreas de líneas.

- **Estructura de una página XSL- FO**

- **<fo:root>** corresponde al elemento raíz.
- Sus descendientes directos son:
    - **<fo:layout-master-set>** patrones de las páginas del documento
    - **<fo:page-sequence>** páginas del documento
- Este elemento **<fo:layout-master-set>** contiene el patrón de la página o grupo de páginas **<fo:simple-master-page>** que dispone de patrones, márgenes, cabecera.

- **Objetos de formateo para la estructura de documentos**

Estos objetos permiten definir el diseño de las páginas, las diferentes regiones, cabeceras, etc.

- **fo:root**

Corresponde al elemento raíz de un documento determinado XSL-FO, cuyo contenido se corresponde a:

<fo:layout-master-set> <fo:declarations>?

<fo:page-sequence> +

- **fo:layout-master-set**

Corresponde a un objeto que contiene todos los modelos de páginas que se pueden utilizar en un documento.

- **fo:simple-page-master**

Ofrece la posibilidad de definir un formato (*layout*) de una página o grupo de páginas con sus correspondientes dimensiones.

Cada objeto de este tipo puede disponer de hasta cinco regiones que son: el cuerpo (*region-body*), cabecera (*region-before*), pie (*regi-after*), margen izquierdo (*region-start*) y margen derecho (*region-end*). De todas estas regiones, la única que es imprescindible es el cuerpo.

Entre sus principales propiedades, destacamos:

- **master- name**: utilizado para indicar el nombre del modelo en cuestión.
- Aquellas que hacen referencia a los estilos de CSS como margin, page-height, page-width, etc.

- **fo:declarations**

Su función principal consiste en unificar las distintas declaraciones de la hoja de estilos. Actúa como contenedor de aquellos objetos de formateo que se van a utilizar a la hora de generar la salida.

- **fo:page-sequence-master**

Indica el orden en el que van a parecer los objetos <fo:simple-page-master>

Entre sus principales prioridades, destacamos:

- **master-name**: para indicar el nombre correspondiente al modelo (*master*)

- **fo:single-page-master-reference**

Hace referencia a <fo:simple-page-master> que se va a insertar solo una vez.

Entre sus principales prioridades, destacamos:

- **master-reference**: para indicar el nombre de <fo:simple-page-master>

- **fo:repeatable-page-master-reference**

Hace referencia a <fo:simple-page-master> que se va a insertar varias veces

Entre sus principales prioridades, destacamos:

- **master-reference**: para indicar el nombre de <fo:simple-page-master>

- **fo:repeatable-page-master-alternatives**

Indica la repetición que realizan un grupo de determinados objetos <fo:simple-page-master>.

Contenido::=<fo:conditional-page-master-reference> +

Entre sus principales prioridades, destacamos:

- **Maximum-repeats**: cuenta el máximo número de repeticiones que pueden aparecer.

- **fo:conditionel-page-master-reference**

Modelo condicional de una página que debe cumplir la serie de condiciones que se indiquen.

Entre sus principales prioridades, destacamos:

- **master-reference**: se refiere al nombre del <fo:simple-page-master>
- **page-position**: señala el valor ordinal de la página a la que se le va a aplicar el modelo.
- **odd-or-even**: puede hacer referencia a las páginas que se van aplicar, en el caso de que sean pares (*even*) o impares (*odd*).

Objetos de formateo para el contenido de páginas:

Utilizados para asignar un contenido determinado a cada página que forme parte del documento.

- **fo:page-sequence**

Actúa como un contenedor de aquellos objetos página de salida. Cada objeto

<fo:page-sequence> hace referencia a otro objeto <fo:page-sequence-master> o <fo:simple-page-master>, cuyo contenido es:

Contenido::=<fo:title>?<fo:static-content><fo:flow>*

Entre sus principales prioridades, destacamos:

- **master-reference**: para indicar el nombre correspondiente al objeto <fo:page-sequence-master> o <fo:simple-page-master> al que está asociado.

- **fo:title**

Objeto que determina el título para una <fo:page-sequence>.

- **fo:static-content**

Dispone de una serie de elementos (estáticos) que actúan como cabeceras o pies de página y tienen posibilidad de poder repetirse en diferentes páginas.

Entre sus principales prioridades, destacamos:

- **flow-name**: para indicar donde se va a ubicar el contenido de <fo:static-content>

- **fo:Flow**

Dispone de una serie de elementos que se van a mostrar en una determinada página.

Entre sus principales prioridades, destacamos:

- **flow-name**: para indicar el lugar en el que se van a posicionar los elementos de flujo.

- **fo:block**

Actúa como si fuera un contenedor a nivel de bloque, como de HTML.

- **fo:inline**

Actúa como un contenedor que ofrece la posibilidad de almacenar diferentes objetos para que se puedan distribuir de forma secuencial sin que se produzca ningún salto de línea.

Objetos de formateo para generar listas:

Se refiere a una serie de objetos de formateo que se pueden utilizar para construir listas. Se utilizan de forma conjunta, como si fuera un bloque.

- **fo:list-block**

Objeto que podemos utilizar para declarar una lista.

Contenido::=<fo:list-item>

- **fo:list-item**

Objeto que podemos utilizar para declarar cada elemento perteneciente a una lista.

Contenido::=<fo:list-item-label><fo:list-item-body>

- **fo:list-item-label**

Objeto que tiene asignada las distintas etiquetas utilizadas en forma de marcadores. Estas etiquetas deben formar parte de un objeto <fo:block>.

Contenido::=(<fo:block> | <fo:block-container> | <fo:table-and-caption>

| <fo:table> | <fo:list-block> | <fo:list-item> | <fo:list-item>)+

- **fo:list-item-body**

Objeto que contiene el cuerpo principal o el contenido de un determinado elemento de la lista correspondiente.

Contenido::=(<fo:block> | <fo:block-container> | <fo:table-and-caption> |

<fo:table> | <fo:list-block> | <fo:list-item> | <fo:list-item>)+

Objetos de formateo para generar tablas

- **fo:table-and-caption**

Actúa como un objeto contenedor para otros objetos relativos en la construcción de tablas, como: <fo:table>, <fo:table-and-caption>, <fo:table-column>, <fo:table-header>, <fo:table-footer>, <fo:table-body>, <fo:table-row>, <fo:table-cell>

Contenido::= <fo:table-caption>?<fo:table-caption>

- **fo:table-caption**

Objeto que cuenta con el título de la tabla, y es descendiente de <fo:table-and-caption>

- **fo:table**

Lleva a cabo la definición de una tabla.

Contenido::=<fo:table-column> <fo:table-header>? <fo:table-body>*

<fo:table-footer>?

- **fo:table-column**

Ofrece la posibilidad de seleccionar el formato de las columnas de la tabla.

Entre sus principales prioridades, destacamos:

- **column-width**: permite indicar el ancho asignado a la columna.

- **fo:table-header**

Hace referencia a la cabecera de la tabla.

Contenido::=(<fo:table-row>+ | <fo:table-cell>+)

- **fo:table-footer**

Hace referencia al pie de la tabla.

Contenido::=(<fo:table-row>+ | <fo:table-cell>+)

- **fo:table-body**

Objeto que va a actuar como contenedor de aquellas filas (<fo:table-row>) y celdas (<fo:table-cell>) pertenecientes a la tabla.

- **fo:table-row**

Objeto encargado de definir una fila perteneciente a una tabla.

Contenido::=<fo:table-cell>+

- **fo:table-cell**

Objeto que ofrece la posibilidad de poder representar una celda correspondiente a una tabla.

Contenido::=(<fo:block> | <fo:block-container> | <fo:table-and-caption> | <fo:table> | <fo:list-block>)+

Objetos de formateo para generar enlaces, imágenes

- **fo:basic-link**

Hace referencia a un enlace determinado del documento.

Entre sus principales prioridades, destacamos:

- **internal-destination**: identificador a donde apunta el enlace al documento.
- **external-destination**: URI de la página a la que lleva el enlace.

- **fo:external-graphic**

Ofrece la posibilidad de poder insertar una imagen.

- **fo:leader**

Ofrece la posibilidad de realizar una línea horizontal.

Entre sus principales prioridades, destacamos:

- **leder-length**: hace referencia a la longitud de la línea.
- **leder-patten**: hace referencia al patrón de la línea.

3. Almacenamiento de información

En esta unidad formativa vamos a definir las bases de datos XML nativas como alternativa a las relacionales, de tal forma que podremos sustituir las tablas con todos sus campos por una serie de documentos HTML, con sus correspondientes elementos y atributos.

La herramienta que vamos a utilizar va a ser BaseX para llevar a cabo dos tipos diferentes de lenguajes de consulta:

- **XPath**. Lenguaje bastante sencillo que dispone de una gran cantidad de expresiones que nos van a permitir la posibilidad de poder acceder a las diferentes partes de un documento HTML.
- **XQuery**. Es otro lenguaje que utiliza XPath para facilitar la manipulación de documentos XML.

Vamos a conocer brevemente una serie de clases que ofrece el lenguaje de alto nivel Java para el tratamiento de documentos XML.

3.1. Sistemas de almacenamiento de información XML nativo.

Tradicionalmente, las bases de datos se han tratado con sistemas gestores de bases de datos (SGBD), siendo los más conocidos y utilizados en anteriores módulos de este ciclo MySQL y ORACLE.

Una alternativa de estos SGBD pueden ser los sistemas de bases de datos nativos XML que, mientras que los primeros estaban basados en tablas con su información correspondiente relacionada entre sí, estos segundos lo que almacenan son documentos XML. De esta forma, podemos comprobar que los SGBD tradicionales son adecuados para almacenar datos, y los gestores de bases de datos XML son adecuados para almacenar documentos.

Entre los ejemplos más conocidos de estos gestores de base de datos nativos, pueden ser:

BaseX

http://basex.org

eXist (programa de pago)

http://exist-db.org

- **BaseX**

Motor de base de datos nativo XML que incluye XPath y XQuery. Presenta las características de ser una aplicación ligera, de alto rendimiento en las distintas operaciones que realiza y fácilmente escalable. Como ya hemos comentado anteriormente, crea las bases de datos a partir de uno o varios documentos XML. Utilizaremos los lenguajes XPtah o XQuery para poder realizar las pertinentes consultas y, de esta forma, poder sacar información de la base de datos.

La información resultante de una consulta, se puede presentar de diferentes formas:

- Modo texto.
- Estructura de árbol.
- Estructura de carpetas.
- Modo tabla.
- Diagrama de dispersión.

En la interfaz gráfica de esta aplicación podemos visualizar fácilmente los distintos apartados para realizar las operaciones, como pueden ser la barra de herramientas, el editor de consulta, las visualizaciones de los datos y la línea de comando. En esta última, ejecutaremos las cláusulas para poder trabajar con bases de datos nativas XML.

En las bases de datos nativas podemos ejecutar tres tipos de sentencia:

- **Opción *Command***: podemos indicar en este apartado los comandos propios de la aplicación para trabajar con las bases de datos, como, crear una base de datos, tratamiento de usuario, abrir o modificar una base de datos, etc.
- **Opción *Search*:** en este apartado indicaremos expresiones XPtah, que detallaremos más adelante.
- **Opción *XQuery***: indicaremos los diferentes comandos para realizar las consultas, utilizaremos el lenguaje XQuery.

3.2. XPath

Lenguaje diseñado para acceder, transformar y dar formato de salida a los documentos XML. Como ya sabemos, un documento XML posee una estructura árbol que tiene su origen en un elemento raíz, y que se va a ir derivando en sus respectivos hijos que, a su vez, pueden ser raíz de otras estructuras árbol. Esta estructura es bastante parecida la estructura jerárquica de directorio de cualquier sistema operativo de un equipo informático.

De la misma forma que podemos recorrer el árbol de directorios mediante unos comandos preestablecidos (cd, dir, ls, etc.), también podemos hacerlo en un documento XML con comandos XPath.

En este apartado veremos la sintaxis y funcionalidades básicas de este nuevo lenguaje de expresiones.

- **Direccionamiento**

Como en todo contenido web, el direccionamiento o la ruta que podemos utilizar en este lenguaje pueden ser de **dos tipos diferentes: absolutos o relativos**. El primero se indica comenzando el direccionamiento a partir del directorio raíz (/) mientras que el segundo, empieza desde el directorio en el que nos encontremos, sabiendo que podemos hacer uso del símbolo "." para representar el directorio actual y el ".." para acceder al directorio padre.

XPath trata al documento XML como un árbol de nodos. El lenguaje DOM, también trabaja de la misma forma, como hemos visto en capítulos anteriores, podemos diferenciar varios **tipos de nodos XPath**:

- **Nodo raíz del documento**: nodo principal, ya que, a partir de los demás, comienzan a derivar los demás. Cada documento, tiene un único nodo raíz y se caracteriza por ser un nodo que no tiene padre y tener, al menos, un hijo.
- **Elemento**: todo elemento de un XML es un elemento XPath.
- **Atributo**: se parece al nodo atributo de un XML.
- **Texto**: en este nodo, podemos almacenar toda la información que deseemos.
- **Comentarios**: conjunto de texto que no es procesado por el compilador, por tanto, el desarrollador del programa explica los pasos que está realizando, para que, en un futuro, cuando retorne el código fuente por alguna avería o incidencia se conozcan los detalles.
- Instrucciones de procesamiento.
- Espacio de nombre (namespace).

Los tipos de datos primitivo más básicos son *String* para las cadenas de caracteres, *number* para representar la información numérica, *boolean* representa el tipo de datos SI/No y, por último, como estamos ante un leguaje similar a DOM, es decir, que existe una jerarquía de nodos, disponemos del *node-set* para representar el conjunto de nodos.

Hemos comentado con anterioridad que este lenguaje XPath es un código de expresiones. A continuación, veamos las más comunes:

elem	Elemento de nombre elem.
/elem	Elemento de nombre elem situado en el directorio raíz (*/).*
e1/e2	Elemento e2 es hijo directo de su padre e1.

e1//e2	Elemento e2 desciende de e1 pero desconocemos su grado. Puede ser hijo, nieto, bisnieto….
//elem	Elemento de nombre elem depende del directorio / pero no sabemos a que nivel .
@atrib	Atributo que se identifica con el nombre de atrib.
*****	Carácter comodín. Cualquier elemento. También puede ser visto como todos los elementos.
@*	En este caso es el comodín de los atributos. Cualquier atributo (Todos los atributos).
.	Como en el sistema operativo Linux, pero llevado a lenguaje de marcas. Representa el nodo actual en el que nos encontramos.
..	Representa el nodo padre.
espNom:*	Hace mención a todos los elementos del espacio de nombre espNom.
@espNom:*	Se referencia a todos los atributos del espacio de nombre espNom

Una vez vista las expresiones más básicas de este lenguaje, el siguiente paso es la ejecución de las mismas. Lo podemos realizar en dos maneras diferentes, o bien mediante un analizador de expresiones de XPath online como por ejemplo en la web de Whitebean:

www.whitebeam.org/library/guide/TechNotes/xpathtestbed.rhtm

O bien la forma alternativa es la descarga de un entorno de desarrollo como por ejemplo BaseX. Siendo la última opción la más recomendable.

- **Acceso a los elementos / atributos**

Este lenguaje se utiliza como analizador de expresión a partir de un documento XML compuesto por nodos con sus identificadores atributos y valores. Utilizamos estas expresiones cuando deseamos tratar un conjunto de nodos en especial. A la hora de escribir bajo el lenguaje de expresiones XPath, lo único que deseamos realizar es trabajar con los nodos de una determinada expresión. Podemos poner el ejemplo */coche/nombre*, lo que nos devolvería de esta expresión, sería el conjunto de nodos nombre que cuelgan de coche, dentro del directorio raíz (/), en caso de no existir ningún nombre en esa ruta, devolverá el conjunto vacío. Existen analizadores que devuelven el contenido textual del conjunto de nodos; para que cualquier analizador devuelva el texto de los nodos, se lo vamos a indicar en la expresión con la cláusula *text()*, por tanto la expresión anterior quedaría */coches/nombre/text()*.

Seguimos en el apartado de acceso, pero vamos a detallar como sería el acceso a los atributos de un nodo.

En este caso, escribiremos */coches/@matrícula*. Si queremos acceder al valor de la matrícula del coche, en este caso, debemos escribir */coches/@matrícula/data()* o bien, */coches/@matrícula/string()* ya que la primera forma, en algunos analizadores, no se puede ejecutar.

- **Acceso a elementos con rutas simples**

Son una serie de rutas a las que podemos acceder mediante un directorio absoluto, es decir, empezando desde el nodo raíz del sistema, devolviendo el conjunto de nodos que se encuentre en dicha ruta.

- **Elementos del lenguaje**

Avanzando en la sintaxis del lenguaje XPath vamos a generar expresiones más complejas a partir de los elementos básicos ya visto. Es posible gracias a los operadores y funciones.

Operadores:

- **Operadores booleanos**

Operador	Codificación alternativa
and	
or	
not	
=	
¡=	
>	>
<	<
>=	>=
<=	<=

- **Operadores aritméticos**

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
div	División
mod	Resto (módulo)

- **Otros Operadores**

Operador	Significado
|	Unión de resultados

Funciones:

- **Funciones numéricas**

Función	Devuelve	Ejemplo
round()	Redondeo	round(3.14) = 3
abs()	Valor absoluto	abs(-7) = 7
floor()	Suelo	floor(7.3) = 7
ceiling()	Techo	floor(7.3) = 8

- **Funciones de cadena**

Función	Devuelve	Ejemplo
substring()	Subcadena	Substring('Beatles', 1, 4) = Beat
starts-with()	Cadena comienza por	starts-with('XML','X') = true
contains()	Cadena contiene	contains('XML','XM') = true
normalize-space()	Espacios normalizados	Normalize-space('The end') = 'The end'
translate()	Cambia caracteres en una cadena	translate('The end', 'The', 'An') = 'An end'
string-length()	Longitud de una cadena	string-length('Beatles') = 7

Funciones que devuelven elementos por su posición

Función	Devuelve
position() = n	Elemento que se encuentra en la n-ésima posición
elemento[n]	Elemento que se encuentra en la n-ésima posición
last()	El último elemento de un conjunto
last() - i	El último elemento –i (ej. Si i=1 -> el penúltimo)

Funciones que devuelven nodos

Función	Devuelve
name()	Nombre del nodo actual
root()	El elemento raíz
node()	Nodo descendientes del actual
comment()	Comentarios
processing-instruction()	Instrucciones de procesamiento

Funciones de agregado

Función	Devuelve
count()	Conteo de elementos
avg()	Media de valores
max()	Valor máximo
min()	Valor mínimo
sum()	Suma de valores

- **Acceso a elementos con filtros de valores**

Podemos acceder a elementos mediante filtros con dos tipos de valores:

Literales: que van entre dobles comillas para acceder a ese valor concreto y recuperados, que permiten el acceso mediante otras expresiones.

Ejemplo de valores literales:

Acceder a un coche cuya matrícula es 0011

/coche/matrícula[@valor="0011"]

Ejemplo de otras expresiones:

Acceder a aquellas personas que tengan un coche de la marca Opel:

/coche/marca[@valor="Opel"]/persona/@dni

En los valores recuperados, deben existir relaciones entre nodos. En este ejemplo anterior, estamos suponiendo que las personas están relacionadas con el nodo coche.

- **Acceso a elementos mediante ejes**

Estas expresiones permiten acceder a partes concretas del árbol XML siempre que exista parentesco entre los distintos nodos, ya que es una condición indispensable.

Función	Uso
shelf::*	Devuelve el propio nodo de contexto. Equivalente a.
child::	Devuelve los nodos "hijo" del nodo de contexto.
parent:: *	Devuelve el nodo padre del nodo contexto. Equivale a.
ancestor::*	Devuelve los "antepasados" (padre, abuelo, hasta el nodo raíz) del nodo de contexto.
ancestor-or-self::*	Devuelve los nodos "antepasados" del nodo de contexto, además del propio nodo de contexto
descendant::*	Devuelve los nodos "descendientes" (hijo, nieto...) del nodo de contexto.
descendant-or-self::*	Devuelve los nodos "descendientes" (hijo, nieto...) del nodo de contexto, además del propio nodo de contexto. Equivalente a //
following::*	Devuelve los nodos que aparezcan después del nodo de contexto en el documento, excluyendo a los nodos descendientes, los atributos y los nodos de espacio de nombres.

preceding::*	Devuelve los nodos que aparezcan antes del nodo de contexto en el documento, excluyendo a los nodos ascendientes, los atributos y los nodos de espacio de nombres.
preceding-sibling::*	Devuelve los "hermanos mayores" del nodo de contexto.
following-sibling::*	Devuelve los "hermanos menores" del nodo de contexto.
attribute::*	Atributos del nodo contexto. Equivalente a @.
namespace::*	Espacio de nombres del nodo de contexto.

3.3. Lenguajes de consulta y manipulación

En este apartado veremos el lenguaje **XQuery,** que permite realizar consultas para extraer y procesar la información contenida en el documento XML.

Podemos definir que XQuery diferencia entre dos partes: una que se asemeja al lenguaje SQL, ya que es un lenguaje orientado a bases de datos donde comparte las mismas cláusulas que este emplea, como *where*, *order by*, etc. ; por otra parte, también podemos compararlo al lenguaje descrito anteriormente XPath, ya que ambos se refieren a documentos de marca como XML. Con este lenguaje comparte los modelos de datos y soporta las mismas funciones y operadores descritos en el apartado anterior.

XQuery, al ser un lenguaje de expresiones, se compone también de los mismos tipos de nodos que el lenguaje anterior: nodo raíz, elementos, atributos, textos, comentarios, expresiones, etc. Y, además, de los mismos tipos de datos primitivos que XPath: numéricos, boleanos, cadenas de texto, fecha hora, etc.

Lo que sí es específico de este lenguaje son las distintas funciones o cláusulas que indicamos a continuación:

- **Función *doc()***: permite leer un documento XML que indiquemos por parámetro, devolviendo el nodo raíz.

- **Función FLWOR**: recibe este nombre por ser un conjunto de funciones que se refieren a *for, let, where, order by, return*. Las iniciales de estas funciones dan nombre al conjunto que la identifica. Estas funciones hacen posible construir funciones en XQuery.

for → indica, mediante un rango de valores, los elementos para tratar.
let → permite declarar variables para que, a continuación, podamos asignarle valores.
where → acompaña a la sentencia *for*, permitiendo introducir una condición, para que la estructura iterativa se repita mientras que se cumpla dicha condición.
order by → parámetro por el que vamos a ordenar la visualización del resultado de una determinada consulta.
return → comando que nos permite devolver un resultado de una expresión dada.

- **Otras cláusulas**:
  declare function → permite declarar funciones en XQuery.
  if … else → simula una situación condicional, ejecutamos una parte de la estructura u otra mediante el cumplimiento de dicha condición.

- **Funciones predefinidas**: como es habitual en todos los lenguajes, existen una serie de funciones predefinidas por el compilador. A continuación, veremos aquellas definidas por XQuery.
  Podemos encontrar funciones referentes a los textos como *uppercase* (para pasar a mayúscula), *substring* (devuelve subcadenas de texto) o *contains* (para comprobar si un carácter pertenece a una cadena de texto). También existen funciones predefinidas referidas a números, como *max* (devuelve el valor máximo de un conjunto de números), *sum* (devuelve la suma) o *avg* (devuelve la media aritmética).
  A continuación, podemos mencionar las funciones predefinidas correspondientes al tipo fecha, como pueden ser, *current-date* o *current-time* (para calcular la fecha o la hora actual).
  Por último, podemos mencionar funciones predefinidas a nivel de nodos XML, como la función *root*, que devuelve el nodo raíz de un árbol.

3.4. Otras tecnologías complementarias

Hasta ahora, hemos podido comprobar a lo largo de esta unidad formativa, los lenguajes más utilizados en los estándares W3C relacionados con la tecnología web, señalando entre los más importantes XPath y XML.

A continuación, vamos a definir dos tipos de lenguajes que no han tenido demasiada repercusión por lo que su uso se ha quedado muy restringido: XLink y XPointer.

- **XLink**. Es un lenguaje que permite vincular documentos XML mediante distintos tipos de enlaces.
- **XPointer**. Es un lenguaje en el que se pueden enlazar fragmentos de ficheros XML. En este caso, puede hacer uso de la identificación de fragmentos del lenguaje XPath.

3.5. Bases de datos relacionales con XML

En puntos anteriores, hemos podido comprobar la existencia de bases de datos nativas, incluyendo los gestores de bases de datos para implementar estos tipos de bases de datos.

En este apartado, vamos a ver las distintas bases de datos relacionales con implementación de XML. Al tratarse de estos tipos de bases de datos utilizaremos sistemas gestores adaptados a ellas como pueden ser: SQL Server u ORACLE.

A continuación, veremos los distintos elementos que nos podemos crear en etas bases de datos.

Empezaremos por el tipo de campo XML (XMLTYPE). Podemos crear una tabla, mediante la sentencia **CREATE TABLE**, visto en módulos anteriores, donde uno de los campos sea de contenido XMLTYPE. Por tanto, a la hora de crearnos un registro, esta columna debe contener un fragmento de código XML.

Una vez creada la tabla con el campo tipo XML, ya podemos realizar todas las operaciones de inserción (*INSERT*) o consulta (*SELECT*). Otra forma de ejecutar una consulta en este lenguaje es con la función **XMLQuery()**, en la cual recibirá como parámetro la consulta, además del campo XML. Además de existe la cláusula **XMLTable()** mediante la cual podemos tratar el resultado de una consulta con XQuery.

Con referencia a las tablas de tipo de XML, igual que las bases de objetos-relacional, podemos crearlas a partir de las sentencias en XQuery

CREATE TABLE nombre_tabla OF XMLTYPE

De manera ya disponemos de una tabla de objetos y podemos utilizar objetos de este tipo de tabla mediante el comando select.

SELECT objeto FROM nombre_tabla

De la misma forma, haremos la modificación y borrado de elementos con UpdateXML() y deleteXML() respectivamente

3.6. Tratamiento de XML desde JAVA

Una alternativa al tratamiento de los documentos XML es hacer lo mediante código JAVA en vez de con tratamiento en Bases de Datos. Una de las colecciones de datos (API) más conocida para realizar esta tarea es JAXP y en su interior se encuentra SAX y DOM.

En este caso definiremos las colecciones de datos, ya que no es objetivo de este módulo el tratamiento de XML con JAVA y lo profundizaremos en siguientes módulos del ciclo.

- **SAX**. Fue la primera colección de funciones utilizada para tratar ficheros XML desde JAVA. Está basada en eventos y, por esta razón. difiere un poco en el tratamiento del documento mediante árbol. Esta API a medida que va leyendo el documento va enviando notificaciones en tiempo real. Es recomendable para documentos grandes, ya que su manera de procesar los datos no los guarda en memoria.
- **DOM**. En este caso la API se utiliza para acceder y tratar documentos XML como estructuras árbol. Es independiente al lenguaje que se esté utilizando, en este caso, Java, ya que lo que realiza son modelos de objetos que se pueden tratar en cualquier lenguaje orientado a las marcas. La jerarquía de objetos DOM, almacena las relaciones entre los distintos nodos de un documento XML para facilitar su tratamiento. La diferencia con la API anterior es que, en este caso, sí que se almacena en memoria todo el documento. Esta característica hace que el espacio de memoria tenga una capacidad considerable y los archivos más recomendado para este modo de trabajar sea los documentos de menor tamaño.

Todo lo anterior está incluido en la API JAXP que se encuentra en la versión 6, de esta forma podemos procesar, tratar, validar y consultar los documentos XML.

Podemos encontrar toda la información de esta API en la siguiente dirección

https://docs.oracle.com/javase/tutorial/jaxp/

UF3. Sistemas de gestión de información empresarial

1. Sistema de gestión empresarial:

En esta Unidad Formativa vamos a conocer las distintas herramientas de **BI** (*Business Intelligence*, Inteligencia del Negocio), que nos permiten almacenar aquella información sobre los diferentes aspectos de una actividad determinada. El principal factor de las aplicaciones de inteligencia del negocio son los **ERP**, sistemas modulares integrados de gestión empresarial.

El componente más importante de los ERP son los **CRM**, que son un tipo de aplicación destinada a la gestión de clientes.

1.1. Inteligencia del negocio

Debido a la aparición de los equipos informáticos, se ha ido modificando la forma de trabajar de las distintas organizaciones que, si en un principio realizaban todo de forma artesanal, ahora disponen de estos ordenadores para poder registrar en ellos toda la información que se considere necesaria. Partiendo desde los archivos de texto hasta unos sistemas gestores de bases de datos bastante sofisticados, sistemas expertos, sistemas CBR (*Case Base Reasoning*, Razonamiento Basado en Casos), etc.

Estos factores son los que han ido evolucionando los sistemas de información hasta conseguir automatizar bastantes tareas sacando unas conclusiones muy significativas.

Desde el momento en el que hemos podido contar con una gran cantidad de datos, que podíamos procesar y recuperar, hemos tenido la oportunidad de poder estudiar los distintos procesos de una actividad, junto con los casos de éxito y de fracaso. Todos estos factores son los que nos han brindado la posibilidad de poder aprender en profundidad el posicionamiento de estos procesos, optimizarlos, transformarlos e incluso eliminarlos. De esta forma, se ha ido generando la denominada inteligencia de negocio.

Por tanto, podemos definir las aplicaciones de inteligencia de negocio como aquellas que consiguen, entre otras funciones, automatizar y agilizar procesos, registrar resultados y favorecer la extracción de conclusiones.

A continuación, vamos a citar algunas familias de aplicaciones son sus particularidades, como son:

- **ERP** (*Enterprise Resource Planning*, Planificación de Recursos Empresariales). Aplicaciones integrales y modulares.
- **CRM** (*Customer Relationships Management*, Gestión de Relaciones con Clientes). Aplicaciones con un amplio rango de funcionalidades.
- **CM** (*Change Management*, Gestión del Cambio). Distintas herramientas que tienen como fin facilitar a los empleados los cambios que tengan que modificar.
- **BA** (*Business Analytics*, Analítica de negocio). Las distintas herramientas que vamos a emplear para analizar datos y resultados hasta que podamos sacar las conclusiones oportunas.
- **ETL** (*Extract Transform and Load data*, Extraer Transformar y cargar datos). Estas aplicaciones tienen como función fusionar aquellos datos de diferentes orígenes.
- **SFA** (*Sales Force Automation system*, sistemas de Automatización de Ventas). Aplicaciones que forman parte de un CRM y permiten registrar las fases de un proceso de ventas.
- **BPM** (*Business Process Management*, Gestión de Procesos de Negocio).
- **MRP** (*Material Resource Planning*, Planificación de los Recursos Materiales).
- **SRM** (*Supplier Relationship Manager*, Gestión de Relaciones con proveedores).
- **KM** (*Knowledge Mangement*, Gestión del conocimiento).

1.2. ERP

La función principal de los ERP consiste en facilitar el flujo de información que hay entre los elementos de una empresa. Sin este sistema, se podrían producir duplicidades de los datos.

Los **ERP** son muy versátiles, ya que pueden funcionar en diferentes sistemas operativos; sus características principales son:

- **Integral**. Abarca todas las necesidades a la hora de gestionar una empresa.
- **Modular**. Organiza las aplicaciones por módulos independientes, aunque con posibilidad de poderlos combinar.
- **Parametrizable**. Consiguen adaptarse a las diferentes necesidades de una empresa.
- **Escalable**. Ofrecen capacidad de crecimiento.

Algunos ejemplos de ERP, podrían ser:

SAP

http://www.sap.com/spain

Odoo:

https://www.odoo.com/es_ES/

A continuación, vamos a centrarnos con más detalle en algunos de los más importantes:

- **SAP (*System Anwendungen and Producte*, Sistemas Aplicaciones y Productos).**

SAP es el nombre de la empresa alemana encargada de desarrollar ERP comercial. Esta empresa cuenta con el desarrollador de software más potente en Europa y es considerada una de las mayores del mundo.

Cuenta con un gran número de módulos que tienen la función de cubrir aquellos procesos de negocio que pueden necesitar las empresas. Algunos de los módulos más importantes son:

FI (*Finalcial*) Finanzas.
CO (*Controlling*) Control y costos.
LO (*Logistics*) 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 *(Custormer Relationship Management)* Gestión de clientes.
QM (*Quality Management*) Gestión de calidad.

Los ERP, mediante la codificación de programas encargados de realizar una función determinada, se adaptan a las diferentes necesidades de la empresa. SAP incorpora un lenguaje de programación de cuarta generación propio, ABAP, que ofrece, entre otros servicios, la posibilidad de trabajar con diferentes datos y acceder a distintas bases de datos. Además, puede realizar llamadas a procesos remotos para establecer una comunicación entre sistemas.

La última versión de SAP añade la codificación de módulos en Java.

- **OpenBravo**.

Es la alternativa a SAP, pero para software libre. Un cliente web es el encargado de ponerlo en marcha, haciendo uso de la interfaz RIA (Rich Internet Application, Aplicación Rica de Internet).

También cuenta con una gran cantidad de módulos que pueden combinarse entre sí, somo:

Gestión de datos maestros
Gestión de aprovisionamientos
Gestión de almacenes
Gestión de proyectos y servicios
Gestión de la producción
Gestión comercial y CRM
Finanzas y contabilidad
Inteligencia de negocio
Retail POS (*Point Of Sale*, Punto de venta)

o **Especificaciones técnicas.** Antes de proceder a la instalación de OpenBravo, debemos tener en cuenta una serie de requisitos, como los que indicamos, a continuación:

- Servidor:
  Sistema Operativo necesario: Windows Server y Linux entre otros que soporten el lenguaje Java.
  Bases de datos PostgreSQL u Oracle.
  Lenguaje de programación Java 2SE
  Fameworg: Hibernate y Weld
  Servidor web Apache
  Generación de informes: JasperReports de Jaspersoft.
- Cliente:
  Web sin necesidad de instalaciones
  JavaScript: librería SmartClient
  Ajax

1.3. CRM

CRM (*Customer Relationship Management*, Gestión de Relaciones con Clientes). Aplicación informática, que a veces se encuentra integrada en un ERP, y está orientada al registro de información de clientes, ventas y marketing. Su función principal es conseguir clientes y mantenerlos.

Las tareas principales que realizan los CRM pueden ser:

- Almacenar datos.
- Lanzar ofertas.
- Realizar informes.
- Llevar a cabo campañas publicitarias.

Algunos de los CRM pueden ser:

- **Salesforce**.
- **CiviCRM**.
- **HiperGate**.
- **SugarCRM**.

A continuación, vamos a ver con más detenimiento el CRM SugarCRM.

- **SugarCRM**

Utiliza software libre

http://www.sugarcrm.com

De la misma forma que los ERPs, cuenta con una serie de módulos, como son:

- Ventas.
- Marketing.
- Servicio al Cliente.

Y destacamos sus características principales, como:

- En sus inicios, se desarrolló para LAMP (Linux, Apache, MySQL y PHP).
- Implementación de la lógica del negocio en PHP.
- En la actualidad puede almacenar diferentes sistemas operativos como: Windows, Linux y Mac OS.

Podemos ejecutar esta aplicación en los diferentes sitios:

- En la nube, de forma local, desde teléfonos inteligentes y tabletas.

Entre sus principales funciones, destacamos:

- Mantener cuentas de empresa, contactos de personas y las diferentes oportunidades de registro.
- Crear informes: tabular, sumarizar, matricial.
- Mantener documentos y casos.
- Registrar llamadas.
- Enviar correos.
- Planificar reuniones.
- Mantener tareas.
- Mantener notas.
- Creación de casos.
- Creación de campañas publicitarias tanto mediante correo electrónico como a través del correo ordinario.
- Crear los diferentes artículos para la base de conocimientos.
- Mantener empleados.

- **Interfaz**

Si queremos realizar alguna prueba de la versión de evaluación, debemos registrarnos (podemos hacerlo de forma gratuita) y, una vez registrado, tenemos la oportunidad de elegir un perfil entre los cinco que tenemos disponibles, y ya podemos tener acceso a gestión de ventas, director de ventas, director de márketing, representante de clientes o administrador de SugarCRM.

Además, también podemos seleccionar el idioma en el que queremos realizar la demostración.

1.4. Instalación, configuración e integración de módulos

Para instalar este gestor de clientes comerciales debemos de disponer de un servidor web, ya que lo vamos a llevar a cabo en nuestro servidor.

Antes de nada, debemos de comprobar los requisitos mínimos. Comprobaremos que nuestro servidor tiene implementado PHP, tenemos disponible una base de datos para el software y un servidor web Apache.

Tanto los pasos de instalación, como su posterior configuración y otros puntos para tener en cuenta en el programa podemos consultarlos en el siguiente enlace oficial de la empresa distribuidora Sugarcrm, que está disponible para cualquier usuario por ser de código libre.

https://www.sugarcrm.com

Al ser una aplicación modular, realizaremos la instalación del formato básico; se pueden instalar los módulos cliente, pedidos y compras. Si necesitáramos nuevos módulos, como el de facturación, que es bastante común, necesitaríamos acceder al panel de control del usuario administrador en el apartado instalador de módulos.

A la hora de elaborar informes, podemos personalizarlos en el apartado correspondiente, dependiendo de las características del programa. Sugarcrm, también cuenta con la posibilidad de exportar los datos de la base de datos mediante archivos .csv, que es el formato más utilizado en esta operación.

En el enlace web anterior, podemos ampliar toda esta información y profundizar, de forma más detallada, en cada punto.

Bibliografía

Juan Manuel Castro Ramos, Jose Ramón Rodríguez Sánchez, "Lenguajes de marcas y sistemas de gestión de información". Garceta, 2015.

José R. García-Bermejo, "JAVA SE6 & Swing. El lenguaje más versátil". PEARSON Prentice Hall, 2007.

Webgrafía

https://docs.oracle.com

ILERNA Online