LENGUAJES DE MARCAS Y SISTEMAS DE GESTIÓN DE LA INFORMACIÓN

Administración de sistemas informáticos en red

Desarrollo de aplicaciones multiplataforma

Desarrollo de aplicaciones web

ILERNA

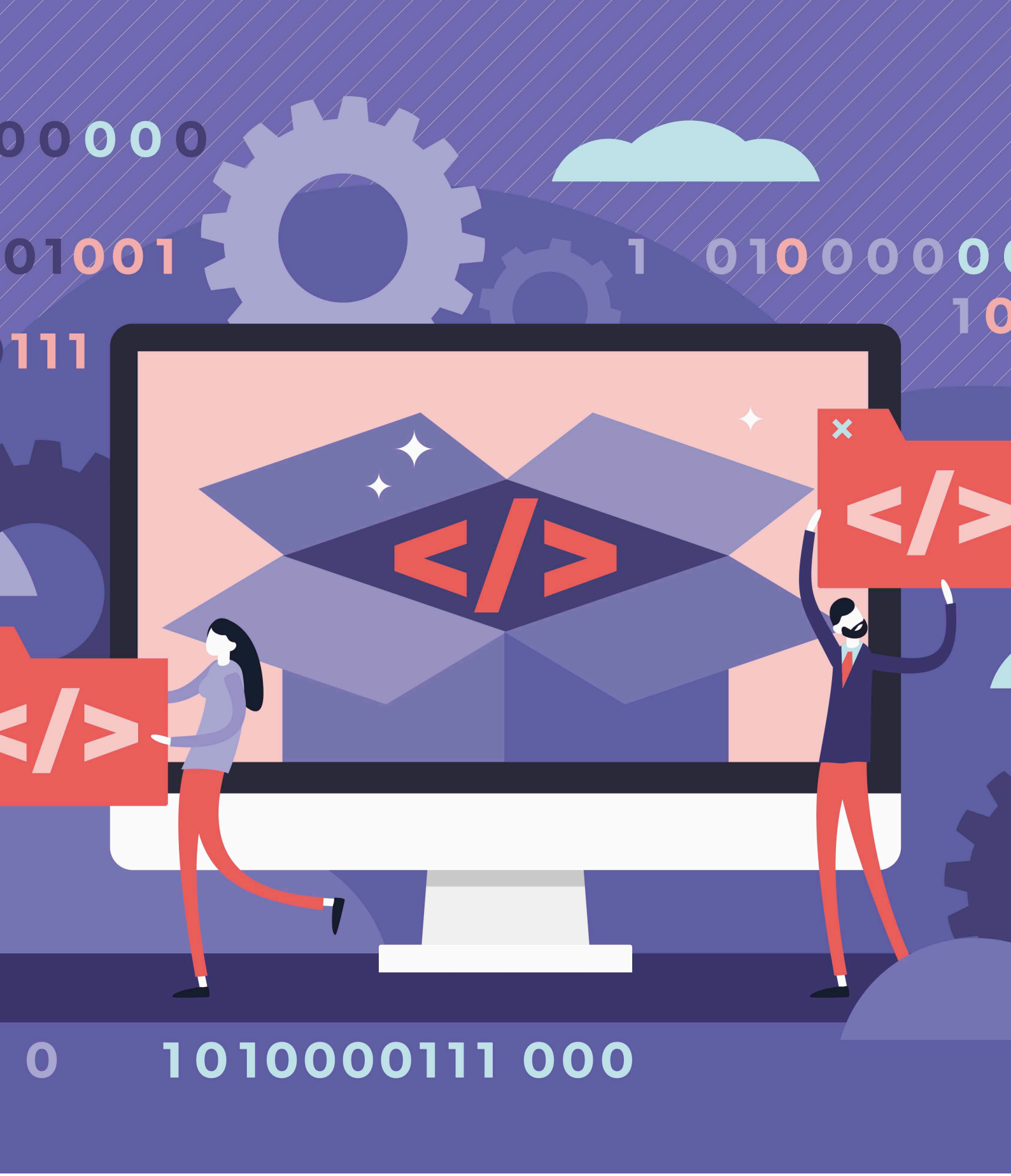
Maquetado e impreso por Ilerna Online S. L.

Impreso en España - Printed in Spain

2.ª edición: marzo 2023

ÍNDICE

Lenguajes de marcas y sistemas de gestión de la información



1 RECONOCIMIENTO DE LAS CARACTERÍSTICAS DE LOS LENGUAJES DE MARCAS

1.1. Conceptos generales

Concepto de lenguaje de marcas
youtu.be/vCVWZ3Cyzag

Los **lenguajes de marcas** o **lenguajes de marcado** son aquellos que combinan la información contenida en un documento a través de etiquetas, marcas o anotaciones relativas a la estructura del texto o su forma de representación.

Este lenguaje especifica qué etiquetas aparecerán, dónde se colocarán y el significado que tendrá cada una de ellas. Normalmente, estas etiquetas o marcas no se mostrarán al usuario final, ya que este estará interesado únicamente en el propio contenido del documento.

A continuación, se puede observar un ejemplo de cómo se muestra la información de una noticia mediante etiquetas o marcas:

```
<noticia>
    <fecha> 03/09/2020 </fecha>
    <lugar> Lleida </lugar>
    <descripcion> Inauguración de una tienda de zapatos </descripcion>
</noticia>
```

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, ni sonido, ni archivos comprimidos ni de 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 e ISO- 8859-15.

Entre sus principales ventajas podemos mencionar que los archivos de texto plano se pueden interpretar mediante un simple editor de textos. Sin embargo, los archivos binarios necesitan un software específico, como por ejemplo descompresores, visores multimedia o 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 número 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. Porque no es lo mismo emplear un móvil que un ordenador.

Especialización

Con la especialización nos referimos a que, con el paso de los años, los lenguajes 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 JavaScript. Además, existen unas etiquetas que ya vienen especificadas para tal fin, como son los *<script>*.

Clasificaciones de lenguaje de marcas

youtu.be/sdQLtfZblW4

1.2. Clasificación

Dependiendo del **tipo de marcas** que utilicemos, podemos dividir los lenguajes de marcas en tres tipos diferentes. Los detallamos a continuación.

De presentación

Son los lenguajes de marcas que permiten dar un estilo al texto sin necesidad de modificar su contenido. Por ejemplo, pueden aumentar el tamaño de la fuente, poner el texto en cursiva, establecer otra tipografía, etc.

Este tipo de lenguaje cumple con la filosofía WYSIWYG (*what you see is what you get* o "lo que ves es lo que obtienes"). En algunos programas que manejan este tipo de lenguajes de marcas, las etiquetas pueden estar ocultas para el usuario.

Las aplicaciones de edición y los procesadores de texto son algunos de los que utilizan este tipo de marcado. Algunos ejemplos de lenguajes de presentación son: Docbook (derivados de SGML) o RichText Format (RTF).

```
{ \rtf1\ansi{\fonttbl\f0\fswiss Free Sans;}\f0pard
Esto es un ejemplo de un documento en {\b RTF}-\par
}
```

Descriptivo, estructural o semántico

En estos tipos de lenguajes se utilizan las etiquetas para realizar una descripción del texto. Hacen referencia a las diferentes partes en las que se puede dividir un documento, pero no ofrecen detalles de cómo se representa el texto o en qué orden se muestra. Es decir, las etiquetas se usan para describir *lo que es* en vez de *cómo se debe presentar* un fragmento de texto de un documento.

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 almacenarán 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 semi estructuradas. Algunos ejemplos de este tipo pueden ser: ASN.1, YAML, EBML, RDF, XFML, OWL, XTM.

```yaml
- Estudiantes:
  - Estudiante:
      estudiante_id: 1
      nombre: Alejandro
      apellido: Claramonte
  - Estudiante:
      estudiante_id: 2
      nombre: Pedro
      apellido: Perez
```

```xml
<?xml version="1.0"?>
<Company>
  <Employee>
      <FirstName>Pedro</FirstName>
      <LastName>Perez</LastName>
      <ContactNo>66682782</ContactNo>
      <Email>pedroperez@gmail.com</Email>
      <Address>
          <City>Santa Cruz</City>
          <State>Spain</State>
          <Zip>38001</Zip>
      </Address>
  </Employee>
</Company>
```

Procedimental

Es un tipo de lenguaje de marcas que da formato al texto del documento, pero el usuario que edita el documento puede visualizar y manipular las propias etiquetas que dan estilo al texto. Estos lenguajes suelen usar macros y rutinas para lograr una mayor eficiencia. Ejemplos típicos de lenguajes de marcas procedimentales son Latex o troff.

```
\documentclass[beamer,onecolumn,12pt]{article}
\usepackage[spanish]{babel}
\usepackage[latin1]{inputenc}
\usepackage{listings}
\AtBeginSection[]
    {
   \begin{frame}
       \frametitle{Resumen}
       \tableofcontents[currentsection]
   \end{frame}
    }
\begin{document}
% Portada de la presentación
\titlepage
% Tabla de contenidos
\input{portada}
\titlepage
```

1.3. Herramientas de edición

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

BaseX

Es un software libre (de licencia BSD) realizado como proyecto por la comunidad GitHub, que consiste en una interfaz gráfica para, no solo editar documentos XML, sino gestionar toda una base de datos XML. También tiene soporte para XPath, XQuery, JSON, HTML, CSV, etc.. Está disponible para Windows, Mac y Linux.

https://basex.org/download/

XMLSpy de Altova

Es un software propietario que consiste en un editor e interfaz gráfica de desarrollo con el que podemos modelar, editar, transformar y depurar tanto documentos XML (y sus derivados XSLT, XPath, XQuery y SXD) como documentos JSON. Cuenta con un gran número de herramientas diferentes, entre las que podemos distinguir las de representación gráfica de diferentes documentos. Aunque se trate de un software de pago, dispone de una versión de prueba gratuita de 30 días. Está diseñado para Windows.

https://www.altova.com/xmlspy-xml-editor

<oXygen/> de SyncroSoft

Es un software propietario que consiste en un editor y sus herramientas de desarrollo para documentos XML y derivados como XSL, XSLT, XQuery y otros. También dispone de una versión gratuita de 30 días. Funciona como una aplicación Java, de modo que puede ejecutarse tanto en Windows como en Linux o Mac.

http://www.oxygenxml.com

XML Copy Editor

Es un software libre que consiste en un editor y validador con el que podemos trabajar XML Schema, DTD, XSLT o XPath. Posee una licencia GNU GPL. Está disponible para Linux, Windows y Mac.

http://www.xml-copy-editor.sourceforge.net

XMLPad Pro Edition de WMHelp

Herramienta bastante fácil de utilizar, cuya función principal es comprobar si un documento XML está bien formado.

http://www.wmhelp.com/

Otras aplicaciones

Existen otra serie de aplicaciones, aunque detallaremos solo algunas:

Stylus Studio y Liquid XML Editor

Ambos son programas de software privativos, aunque también tienen una versión gratuita y temporal de prueba.

Parser XML

Es un procesador que se encarga de leer documentos 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 que se han producido.

BUSCA EN LA WEB

Stylus Studio

www.stylusstudio.com/xml-download.html

Liquid XML Editor

www.liquid-technologies.com/xml-editor

ponte a prueba

¿Qué tipos de herramientas podemos utilizar en un documento XML?

a) <oXygen/>
b) XML Copy Editor
c) XMLSpy
d) Todas las opciones son correctas

¿De qué forma se organiza la información de un documento XML?

a) Jerárquica
b) Relacional
c) En red
d) Todas las opciones son correctas

¿Qué nombre falta en la etiqueta p1?

```
<profesor>
    <nombre> Roberto </nombre>
    <apellido> Nadal <p1>
</profesor>
```

a) /apellido
b) apellido
c) /profesor
d) nombre

1.4. XML: Estructura, sintaxis y etiquetas

Estructura

Un documento XML tiene una estructura formada por los siguientes ítems:

- **Elementos:** son la unidad básica de los documentos XML. Tienen como función principal almacenar información, de tal modo que en un elemento XML podemos albergar información (como cadena de texto), atributos (que caracterizarán al elemento) u otros elementos (que serán descendentes del original).

  La sintaxis de un elemento XML es la siguiente:

```
<nombre_del_elemento atributo1 atributo2>
Contenido de la Información
</nombre_del_elemento>
```

  El *nombre_del_elemento* es el nombre que se asigna al elemento XML. Normalmente tiene dos etiquetas: una de apertura y otra de cierre, que deben coincidir:

```
<ejemplo>…</ejemplo>
```

  Existen, además, dos tipos de elementos especiales:

  – **Elemento raíz:** es el elemento principal a partir del cual derivan los otros elementos en el documento XML. Si el documento está bien formado, debe existir un solo elemento raíz.

  – **Elementos sin contenido:** tenga atributos o no, debe abrirse y cerrarse con una sola instrucción. Su sintaxis es la siguiente:

```
<nombre_del_elemento />
```

  En el siguiente ejemplo, podemos observar que el elemento principal es <cliente>, el cual contiene tres elementos más: <nombre>, <apellido> y <teléfono>.

```
<cliente>
    <nombre> Juan </nombre>
    <apellido> Espinosa </apellido>
    <telefono> 961377420 </telefono>
</cliente>
```

- **Atributos:** son una propiedad de los elementos, que tienen un nombre y un valor contenidos dentro de la etiqueta de apertura del elemento o bien en un elemento sin contenido. Son de tipo texto, de tal modo que el valor debe aparecer entre comillas. Un ejemplo sería el siguiente:

```
<persona DNI="35427641M">…</persona>
```

  En este caso, *DNI* es el nombre del atributo y *35427641M* es el valor del atributo.

- **Raíz:** el nodo raíz es el primer elemento que encontramos en la estructura y lo designamos como "/".
- **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 representar caracteres especiales de marcado, aunque se interpreten por el procesador como si fueran texto.

Entidad	Carácter
	Espacio en blanco
&amp;	&
<	<
>	>
'	'
"	"

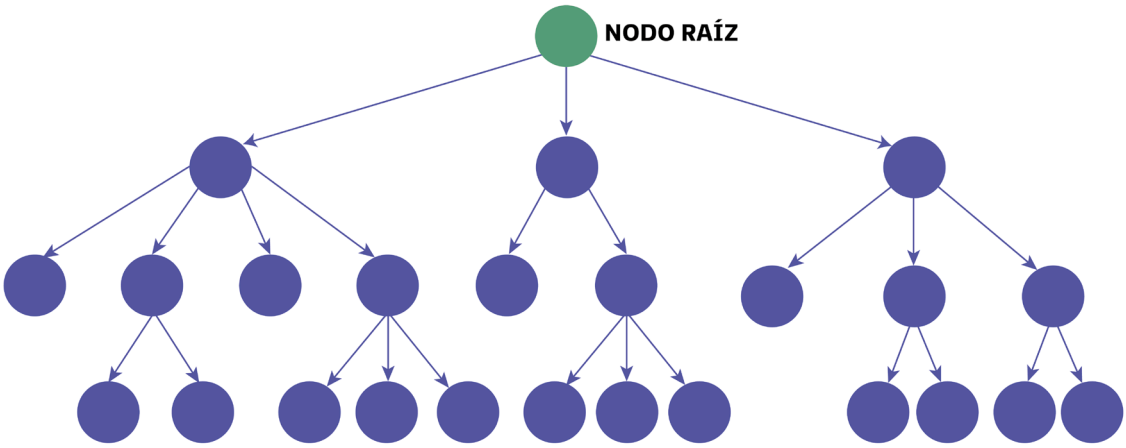
- **Secciones CDATA:** grupo de varios caracteres que no necesitan ser analizados por el procesador.

- **Definición Tipo de Documento (DTD):** se definen unas reglas que asignan una serie de restricciones sobre la estructura del documento XML.

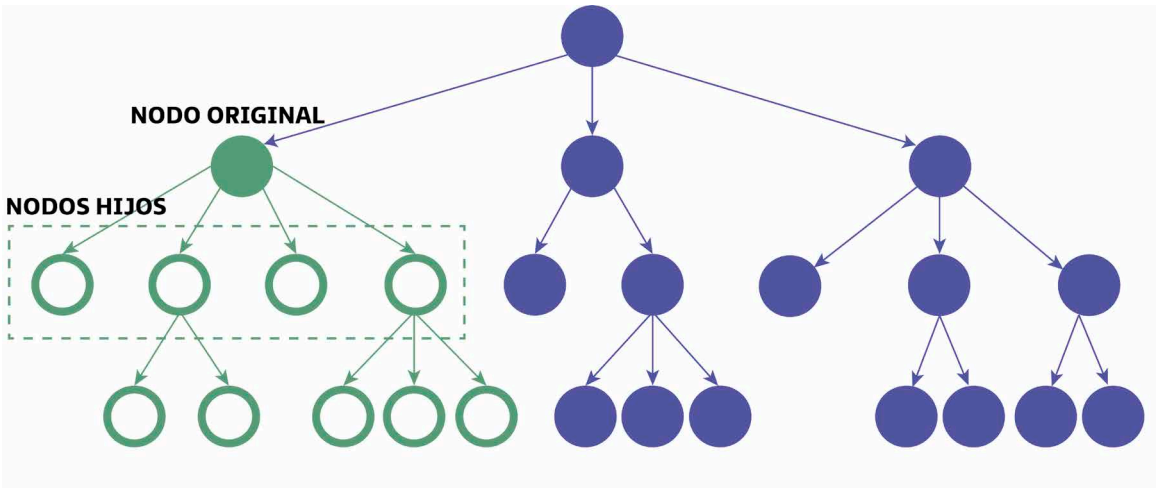
La información de un documento XML se organiza de forma jerárquica, de tal forma que los diferentes elementos del documento se van a relacionar entre sí mediante relaciones de padres, hijos, hermanos, ascendentes, descendentes, etc.

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

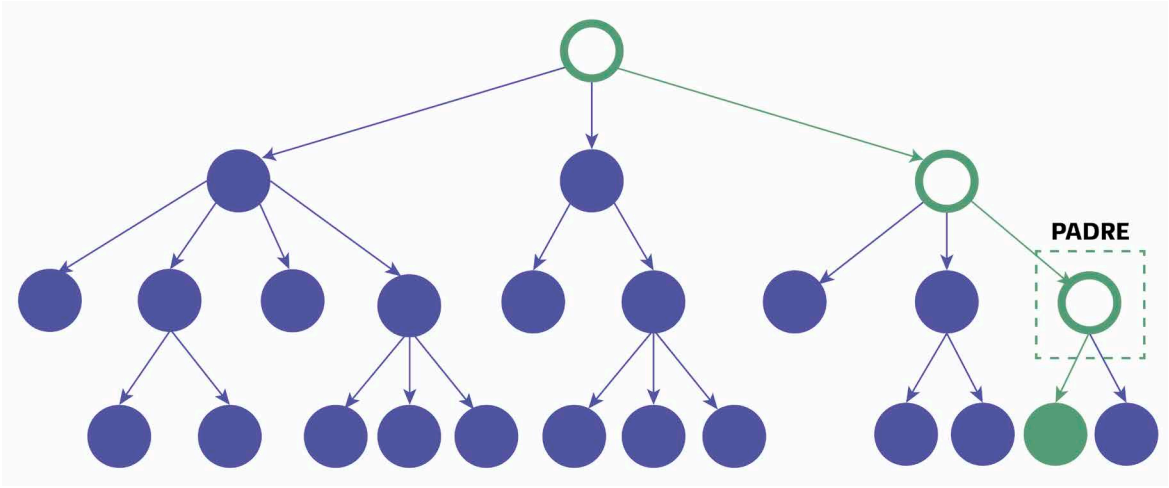
- **Nodo raíz: nodo a partir del cual descienden el resto de nodos.**



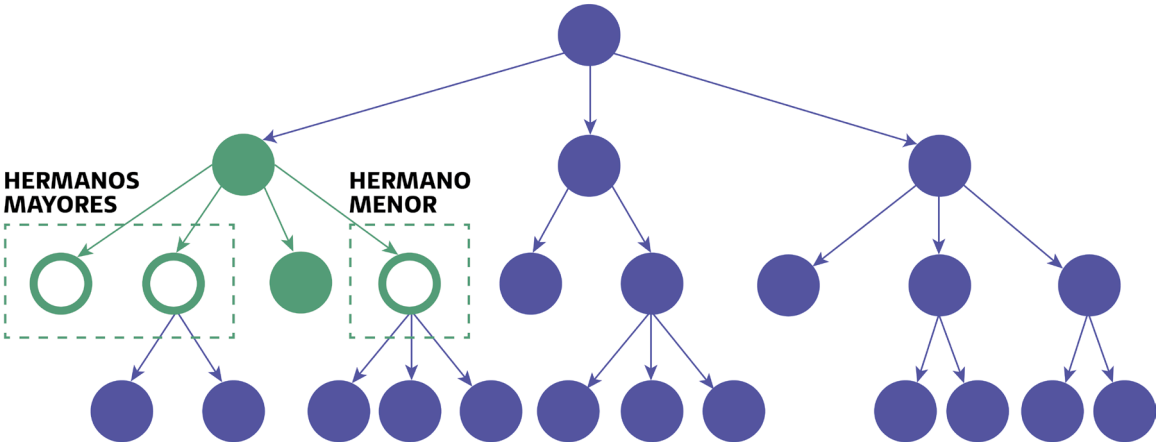
- **Nodos descendentes:** a partir de un nodo original pueden derivan otros nodos que llamamos nodos hijos. Y a partir de estos, pueden derivar más nodos. Los nodos descendentes respecto a un nodo original son todos aquellos que deriven directa o indirectamente del nodo original.



- **Nodos ascendentes:** son todos aquellos nodos antecesores a un nodo concreto.



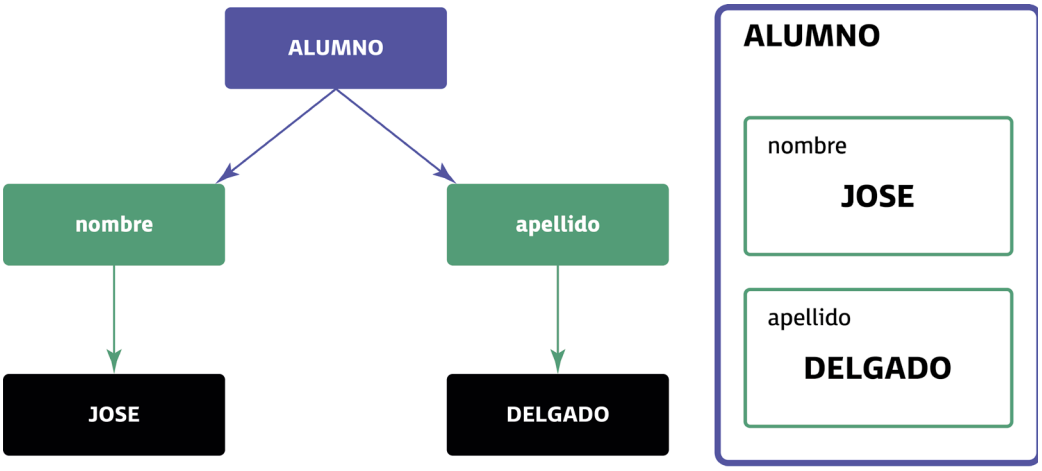
- **Nodos hermanos:** son todos aquellos nodos que comparten el mismo nodo padre .



Veamos ahora el ejemplo del elemento *<alumno>*, que es *padre* de los elementos *<nombre>* y *<apellido>*, que son *hermanos* entre sí.

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

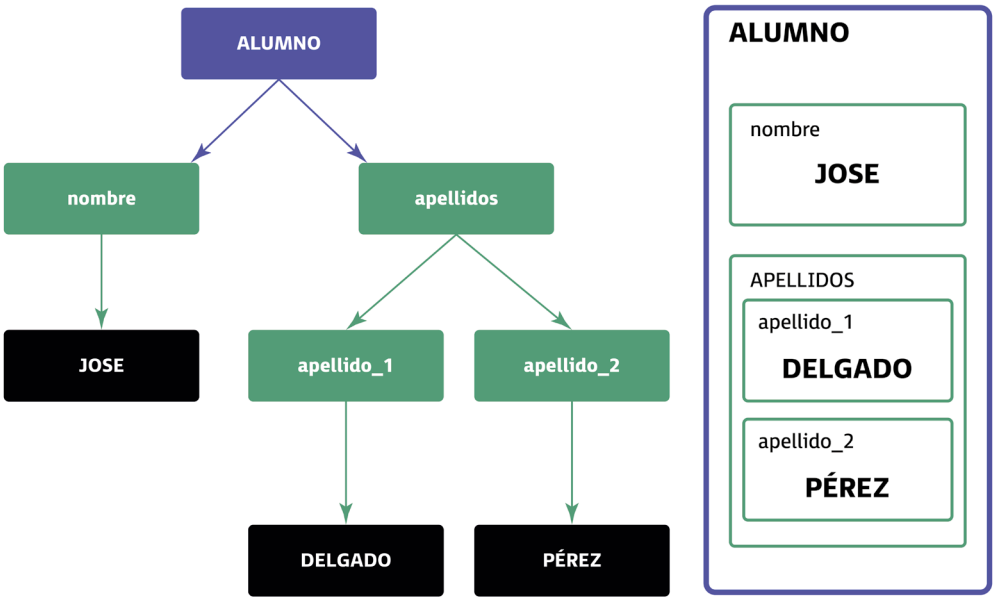
Si lo representamos, nos queda de la siguiente forma:

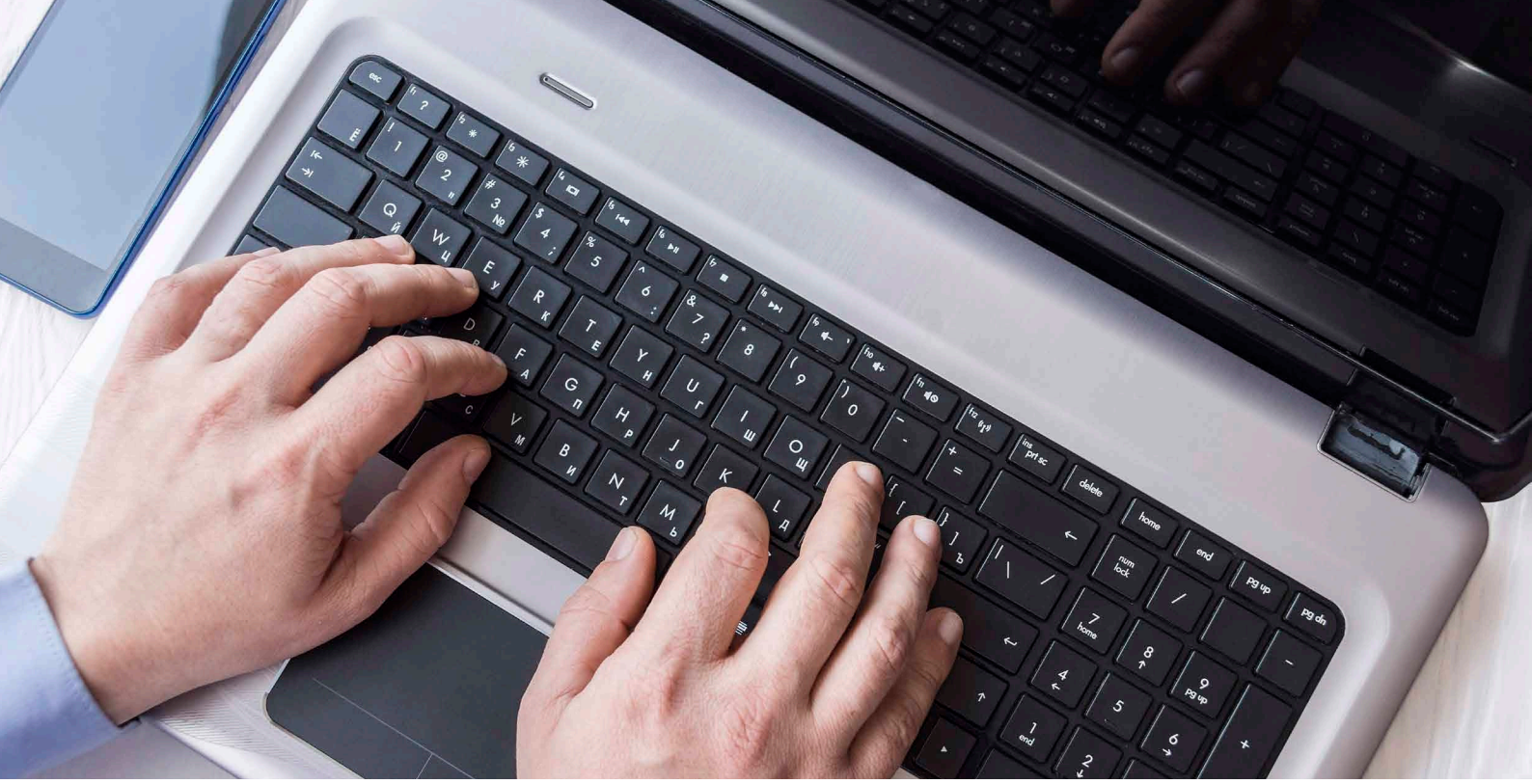


Supongamos que también queremos recoger los dos apellidos de cada alumno. En tal caso tendríamos una nueva ramificación en el árbol:

```
<alumno>
    <nombre>Jose</nombre>
    <apellidos>
          <apellido_1> Delgado </apellido_1>
          <apellido_2>Pérez </apellido_2>
</apellidos>
</alumno>
```

Si lo representamos, nos queda de la siguiente forma:





Sintaxis y etiquetas

Las etiquetas XML son elementos básicos que conforman los nodos de la estructura de un documento XML. Para la elaboración de dichas etiquetas existen unas reglas sintácticas dictaminadas por el W3C (World Wide Web Consortium) que debemos tener presente a la hora de elaborar un documento XML y que describimos a continuación:

- Hay sensibilidad hacia mayúsculas y minúsculas. Por ejemplo: la etiqueta *<persona>* es diferente a la etiqueta *<Persona>*.
- Todo documento XML debe tener una etiqueta raíz.
- La *declaración* (también llamada *prólogo*) XML es opcional. Y, en caso de incluirla, esta debe estar situada al principio del documento (suele ser útil para especificar el tipo de codificación usada en el documento).
- Todos los elementos XML deben tener una etiqueta de cierre (la declaración no se considera como un elemento propio del documento y, por ello, queda exenta de esta norma).
- El valor de los atributos que tengan las etiquetas debe estar siempre entrecomillado.
- Las etiquetas deben estar bien anidadas. La etiqueta de cierre de un elemento *padre* no podrá aparecer antes de que lo hayan hechos todas las etiquetas de cierre de sus elementos hijos.
- Debe evitarse el uso explícito de los caracteres reservados en las etiquetas. Un ejemplo de carácter reservado es el mayor que: > .

- Los nombres de los elementos deben iniciarse con una letra o con el símbolo de guion bajo _.
- La sintaxis de un comentario XML es: *<!-- comentario -->*.

1.5. Elaboración de documentos XML bien formados

Para que un documento XML esté bien formado debe cumplir con las reglas sintácticas que se han listado anteriormente.

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.

Respecto a la nomenclatura de nombres de etiquetas debemos tener en cuenta lo siguiente:

- Tienen permitido empezar con una letra (puede tener tilde o no), que pertenezca 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 reservados.
- No deben contener espacios en blanco.

Estas normas se aplican también a los atributos.

No todos los lenguajes de marcas son tan restrictivos como XML. Por ejemplo, los navegadores de páginas HTML permiten abrir y visualizar un documento HTML con algunos errores, como puede ser alguna etiqueta sin cerrar. En cambio, en XML esto nunca se permitirá.

Por otro lado, debemos tener bien presente la diferenciación de la utilidad de los elementos y los atributos.

Elementos

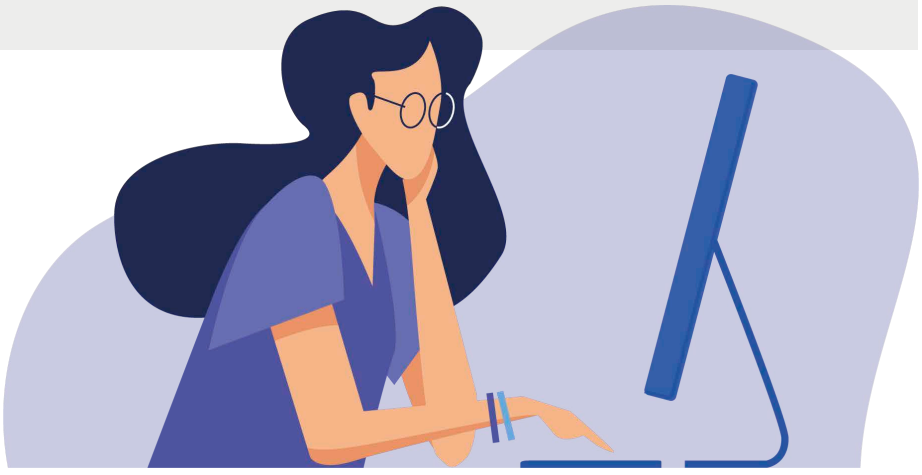
- Se pueden utilizar para representaciones jerárquicas.
- Pueden tener atributos.
- Puede que un elemento tenga distintas ocurrencias.
- Aparecen en orden representativo.
- Pueden extender algunos elementos en su interior.

Atributos

- Deben asociar a los diferentes elementos.
- Permiten alterar la información.
- Aparecen en orden no representativo.
- Permiten realizar el registro de metadatos.
- No pueden extender otros elementos en su interior.
- Un atributo no dispone de múltiples ocurrencias.

A continuación, mostramos un ejemplo de documento XML bien formado:

```
<?xml version="1.0" encoding="UTF-8"?>
<biblioteca>
    <libro>
          <titulo>100 años de soledad</titulo>
          <autor>Gabriel García Márquez</autor>
          <ISBN>84-9759-220-4</ISBN>
          <editorial>Aspasa</editorial>
    </libro>
    <libro>
          <titulo> Cartago </titulo>
          <autor>Ross Leckie</autor>
          <ISBN>978-84-350-6197-1</ISBN>
          <editorial>edjasa</editorial>
    </libro>
</biblioteca>
```



ponte a prueba

En el siguiente ejemplo XML, ¿cuál es el elemento raíz?

```
<alumno>
  <nombre>Roberto</nombre>
  <segundo_nombre>Mesa</segundo_nombre>
  <edad>19</edad>
</alumno>
```

a) alumno

b) nombre

c) edad

d) profesor

¿Cuántos "hermanos" tiene la etiqueta <nombre> en el anterior ejemplo XML?

a) 1

b) 2

c) 3

d) No tiene

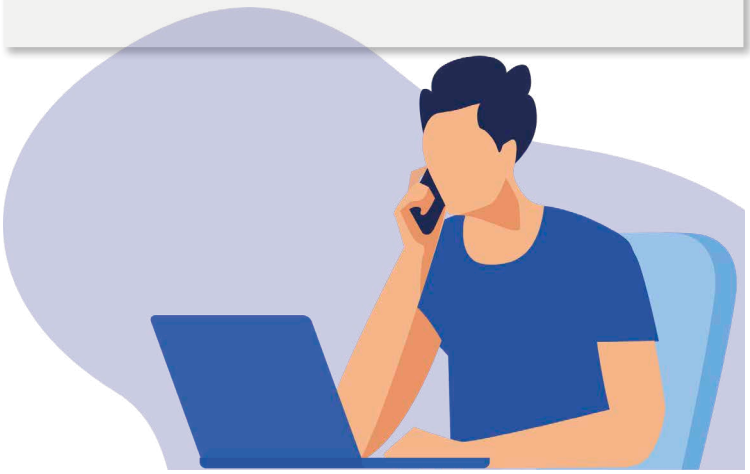
Se definen una serie de reglas que asignan unas restricciones sobre la estructura del documento XML.

a) Definición tipo de documento (DTD)

b) Atributos

c) Texto

d) Espacio de nombres



1.6. Utilización de espacio de nombres en XML

Espacio de nombres

Es posible que al crear un documento XML varios elementos coincidan en el nombre. Por ejemplo, podría haber dos elementos llamados *<id>…</id>*, uno que haga referencia al *id del alumno* y otro que haga referencia al *id de la asignatura*.

Esto puede ocurrir en varias circunstancias. Por ejemplo, al combinar en un mismo documento dos vocabularios XML distintos. Imaginemos el caso de un documento XHTML que posee una parte en *musicXML*. Podría ocurrir que *XHTML* y *musicXML* tuviesen alguna etiqueta con el mismo nombre.

Por tanto, un espacio de nombres es una colección de etiquetas XML con un ámbito semántico en común, contenidas y definidas en una misma URI (identificador de recursos uniforme). Dentro de una URI no pueden coincidir dos etiquetas, pero sí sería posible que dos etiquetas de diferentes URI coincidieran, como en el caso de *<id>*.

La sintaxis para declarar un espacio de nombres en un documento XML es la siguiente:

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

Un ejemplo podría ser el siguiente:

```
<alu:alumno xmlns:alu='http://web.ejemplo.com/espacio_nombres_XML/
alumno'>
    <alu:codigo> 828392 </alu:codigo>
    <alu:nombre> Pedro Albors </alu:nombre>
<asig:asignatura xmlns:asig='http:// web.exam.com/asignatura_namespa-
ce>
          <asig:codigo> A244D </asig:código>
          <asig:nombre>Programacion B </asig:nombre>
    </asig:asignatura>
</alu:alumno>
```

Podemos observar dos espacios de nombres, con sus respectivos prefijos (*alu* y *asig*) y sus respectivas URI (en este caso inventadas).En el documento aparecen dos etiquetas coincidentes, *<código>*, pero para diferenciarlas se ha establecido el espacio de nombres de cada una.

El URI es un nombre lógico del espacio de nombres. Debe ser único, aunque no exista ninguna conexión que lo compruebe. El espacio de nombres afecta al elemento que lo declara, así como a todos los elementos descendientes

que deriven de este. En caso de declararse en un elemento descendiente otro espacio de nombres, para los elementos descendientes prevalecerá el último declarado, es decir, el más próximo en la estructura de árbol.

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 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 refiramos 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

Los lenguajes de marcas se empezaron a utilizar a finales de los años 60, permitiendo introducir anotaciones en documentos electrónicos. Esta posibilidad de añadir anotaciones o marcas es de donde recibe su nombre.

En esta década se estandariza el lenguaje **SGML** (Standard Generalized Markup Language), dando lugar a la posibilidad de compartir información a través de sistemas informáticos. Lamentablemente, este estándar no consiguió asentarse debido a su complejidad.

Cuando hablamos del origen del HTML nos remontamos al año 1980, cuando el físico Tim Berners-Lee, que trabajaba para CERN (Organización Europea para la Investigación Nuclear), propone un nuevo sistema de hipertexto para compartir información utilizando redes de computadoras y a través de Internet.

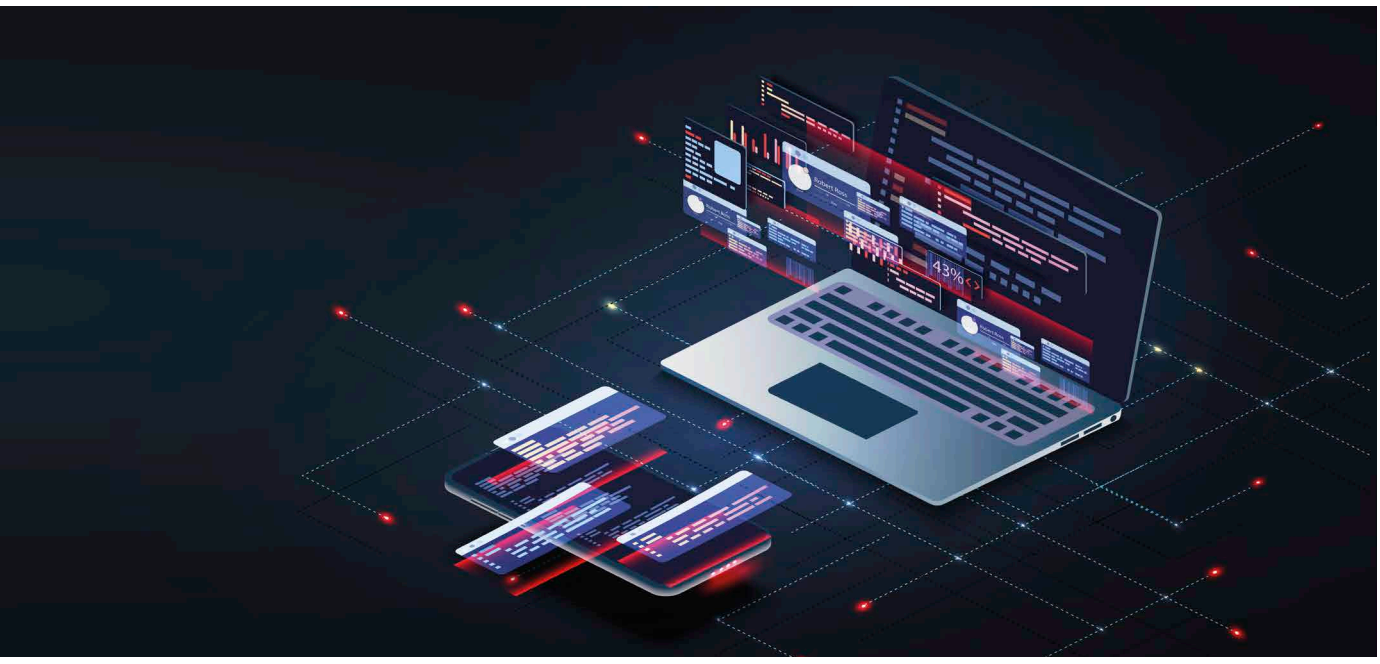
Estos sistemas de hipertexto ya se habían desarrollado con anterioridad, aunque en el ámbito de la informática, cuando hablamos de hipertexto nos referimos a que los usuarios puedan acceder a la información que esté relacionada con aquellos documentos electrónicos que están visibles. Así, los hipertextos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

Años después, Tim Bereners-Lee se une al ingeniero de sistemas Robert Cailliau, con el que gana la propuesta World Wide Web (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 al definirse las etiquetas de imágenes, las tablas y los formularios, no se llegó a conseguir que este fuera el estándar oficial.
- Ya en **1995**, es el organismo IETF el que 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 de **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. Apareció la posibilidad de añadir pequeños programas (*scripts*) en las páginas web, con lo que se consiguió mejorar la accesibilidad de las páginas que ya estaban diseñadas, trabajar mediante la utilización de tablas más complejas y mejorar los formularios.

- La publicación de HTML 4.01 en **1999** se basó, sobre todo, en revisar publicaciones anteriores, pero no añadió avances significativos. Se 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 Apple, Mozilla y Opera empezaron a mostrar su preocupación por la falta de interés denle el W3C de HTML. Fue entonces cuando empezaron a organizar una nueva asociación denominada WHATWG (Web Hypertext Application Technology Working Group).
- De forma paralela, se seguía avanzando en 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.
- Las versiones XHTML 1.1 y XHTML 2.0 se publicaron en forma de borrador con la intención de poder modularizar XHTML.
- En marzo de **2007** se publicaron distintos borradores de HTML 5.0.
- El 22 de enero de **2008** se publicó el primer borrador oficial del estándar HTML5.



AÑO	EVENTO
1991	Nacimiento. "HTML tags"
1993	HTML 1.2
1994	Se funda W3C
1995	HTML 2.0
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 de este lenguaje de marcas a lo largo de los años, nos detendremos un poco en conocer más a fondo este lenguaje.

2.2. Identificación de etiquetas y atributos de HTML

HTML (Hypertext Markup Language) es un lenguaje de marcas que nos permite desarrollar diferentes páginas web. Para poder desarrollar una página web necesitamos:

- Un editor de textos mediante el que vamos a 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* y *.htm*.

Existen diferentes editores de textos para el desarrollo de una página web. Más adelante veremos con más detalle los editores y otras herramientas de desarrollo web.

Cuando hayamos desarrollado los documentos HTML y tengamos la página web lista para su posterior publicación en internet, vamos a necesitar un servidor de páginas web en el 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 para todos los usuarios que pertenezcan a la misma red.

Existen proveedores de servicios de Internet que ofrecen a sus clientes sitios web gratuitos, para que puedan publicar sus páginas web personales o corporativas y, de esta forma, evitar la instalación de un servidor web propio.

Al igual que en XML, un documento HTML es un conjunto de nodos enlazados de manera jerárquica en forma de árbol. Los *elementos HTML* son sus componentes principales. Es decir, un elemento HTML es un nodo del árbol.

Un elemento HTML está formado por sus etiquetas, un contenido y unos atributos opcionales. Los atributos son las especificaciones o caracterizaciones de los elementos. Es decir, los atributos otorgan una serie de propiedades a los elementos.

El contenido es la información en formato texto que está entre las etiquetas de apertura y cierre.

2.2.1. Etiquetas y atributos

Los elementos HTML poseen una sintaxis y unas normas determinadas. Para que un documento HTML sea válido deben respetarse dichas normas.

No todos los elementos tienen la misma estructura, pero de manera general, la composición de un elemento HTML es la siguiente:

```
< etiqueta   atributo="valor">   contenido   </ etiqueta>
```

Como podemos observar, las etiquetas delimitan el elemento, estando el atributo dentro de la misma etiqueta y el contenido entre la etiqueta de apertura y la de cierre.

Etiquetas

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

- **Etiquetas cerradas:** constan 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:

```
<p>   El hombre llegó a la luna el 24 de julio de 1969    </p>
```

  La etiqueta * * es la de apertura y la etiqueta * * es la de cierre.

- **Etiquetas abiertas:** cuentan con una única palabra reservada para indicar el inicio y fin de la etiqueta a la vez.

  Por ejemplo:

```
<hr>
<br>
<img>
```

Atributos

Los atributos modifican el significado de un elemento HTML o incluso pueden proveerle de funcionalidad. Las etiquetas pueden contener atributos si necesitan realizar alguna configuración sobre una característica determinada. Existen atributos obligatorios y atributos opcionales, lo cual dependerá de la etiqueta HTML en la que se sitúen.

Respecto a la sintaxis, 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 son sensibles a mayúsculas o minúsculas. Por ello, es recomendable utilizar siempre minúsculas para etiquetas y atributos. De esta forma, evitaremos confusiones.

Los atributos se colocan en el interior de la etiqueta de apertura, después del nombre de la etiqueta y separados por un espacio en blanco. Por tanto, el atributo se coloca antes del signo de cierre de la etiqueta de apertura.

Los atributos se componen por un conjunto de palabras *nombre-valor*: esto es, el nombre que define qué atributo se va a usar y su valor asignado correspondiente. Ambas palabras se separan mediante el signo =. Por ejemplo:

```
<p align = "left">
```

En este caso, tenemos la etiqueta * * correspondiente a *párrafo* junto al atributo *align = left,* que indica que la alineación del texto de dicho párrafo estará a la izquierda. Es recomendable poner el valor entre comillas dobles para que sea más legible.

Cada etiqueta cuenta con una serie de atributos con sus correspondientes valores, pero hay atributos que pueden usarse en diferentes etiquetas. Una etiqueta HTML puede tener varios atributos a la vez. Por ejemplo:

```
<a href="http://www.myweb.com"  target="_blank"> Esto es un link </a>
```

ponte a prueba

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

a) Verdadero

b) Falso

Definen una serie de elementos que forman el léxico del lenguaje HTML. Se encuentran entre los signos de menor que (<) y mayor que (>). ¿De qué estamos hablando?

a) Etiquetas

b) Atributos

c) Documentos

d) Imágenes

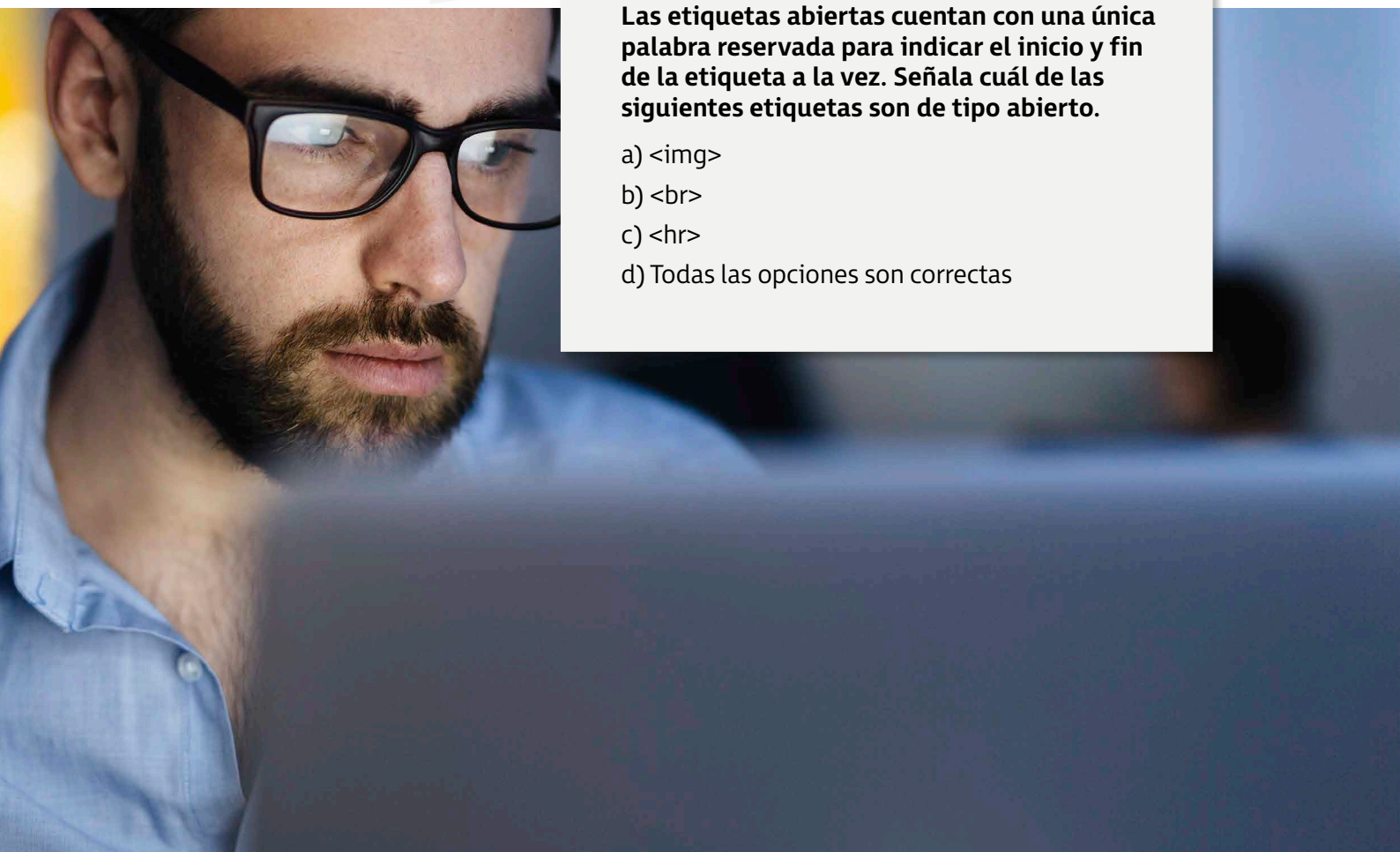
Las etiquetas abiertas cuentan con una única palabra reservada para indicar el inicio y fin de la etiqueta a la vez. Señala cuál de las siguientes etiquetas son de tipo abierto.

a) <img>

b)

c) <hr>

d) Todas las opciones son correctas



2.3. Estructura básica

Estructura básica

youtu.be/HvrgYn0cmvI

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 contenido que se encuentre delimitado entre la apertura y cierre de la etiqueta correspondiente. El contenido puede ser una cadena de texto con información, otras etiquetas o incluso un código funcional.

La estructura básica de un documento HTML es la siguiente:

```
<html>
    <head>
        <title>
            título del documento
        </title>
    </head>
    <body>
        (aquí se colocaría todo el contenido)
    </body>
</html>
```

Como podemos observar, los elementos están anidados y cada etiqueta de apertura está correspondida con otra de cierre. Veamos cuál es el significado de cada etiqueta:

- **<html>:** el elemento raíz del documento. Esta etiqueta engloba toda la página HTML.
- **<head>:** la cabecera del documento. Contiene información que no es directamente el contenido de la página. Por ejemplo, título de la página web, metadatos, estilos, código JavaScript, etc.
- **<title>:** el nombre que le damos al título que se asignará al documento.
- **<body>:** el cuerpo del documento. Aquí se codifica todo aquello que deseamos que se visualice en el navegador. Por ejemplo, los párrafos, las imágenes, formularios, etc.

HTML ha ido evolucionando a lo largo del tiempo, de modo que existen diferentes versiones de HTML que han ido aportando mejoras. HTML 5 es la más reciente.

Y es precisamente en HTML5 que mediante la aportación de unas etiquetas nuevas, se consigue añadir 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 con un nombre asignado que se corresponde con la tarea a realizar.

Algunas de estas etiquetas son las que detallamos a continuación:

- **<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 se utilizan para agrupar los diferentes elementos hijos haciendo uso de atributos como: *class, id* o *title*. 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 a ir realizando en cada momento, aunque no son interpretados por el navegador. Es decir, los comentarios que indiquemos en el código HTML no se visualizan en el navegador.

A lo largo del código fuente del desarrollador es conveniente utilizar una serie de comentarios para explicar con palabras específicas lo que se va realizando en cada momento. De esta forma, se quiere 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.

La sintaxis de los comentarios es la siguiente:

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

Por ejemplo, en la línea 6 de este documento HTML hemos insertado un comentario. El software editor ha detectado el comentario y por eso le ha otorgado otro color, que en este caso es grisáceo.

```
1  <html>
2      <head>
3          <title>
4              título del documento
5          </title>
6      </head>  <!-- aquí finaliza la etiqueta de la cabecera -
7      <body>
8          (aquí se colocaría todo el contenido)
9      </body>
10 </html>
```

ponte a prueba

Es la etiqueta que se utiliza para el inicio del cuerpo.

a) <body>

b) <title>

c) <head>

d) Ninguna opción es correcta

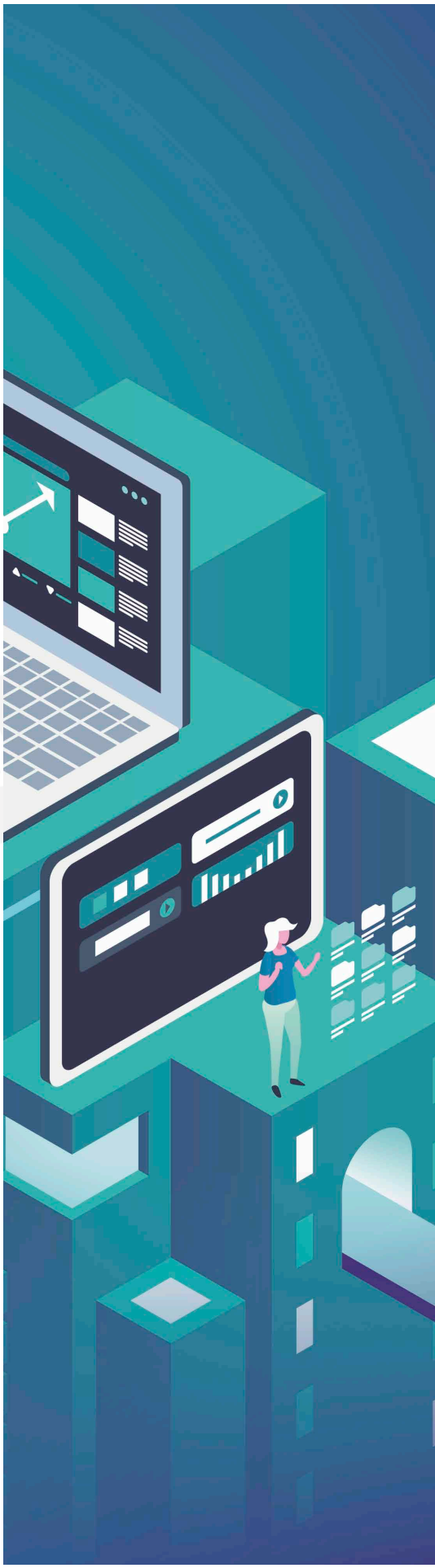
¿Cómo se representan los comentarios en lenguaje HTML?

a) <!- -Esto es un comentario- ->

b) //Esto es un comentario

c) (Esto es un comentario)

d) Todas las opciones son válidas



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.

Cabecera

Un documento HTML se puede dividir entre la cabecera y el cuerpo, es decir, entre *<head>* y *<body>*.

La cabecera contiene información general acerca del documento, pero normalmente la información que figura dentro de la cabecera no es la destinada a visualizarse en el navegador. Salvo el título, que sí es mostrado por el navegador, normalmente en la pestaña.

El elemento de la cabecera se encuentra justo después del elemento raíz. Es decir, la etiqueta *<head>* es el primer descendiente de la etiqueta raíz *<html>*. La cabecera tiene una etiqueta de apertura y otra de cierre y su sintaxis es la siguiente:

```
<head>
…
</head>
```

Dentro del elemento de la cabecera pueden existir otros elementos. Es decir, dentro de *<head>* pueden incluirse otras etiquetas hijas. El único elemento obligatorio que debe estar en la cabecera es el que especifica el título, *<title>* (salvo que esté indicado en otro lugar). El resto de elementos son opcionales.

Los elementos que podemos incluir dentro de la cabecera son los siguientes:

Cabecera <head>	
Elementos hijos	**Significado**
<title>	Título del documento HTML que va a aparecer en el navegador web, en la barra superior
<style>	Usado para determinar el estilo de la información del documento, es decir, información CSS que nos permite insertar una hoja de estilo en la cabecera. Mediante el atributo *type* indicamos el lenguaje del estilo
<link>	Relaciona el documento HTML con otro documento externo, como por ejemplo un archivo CSS
<meta>	Utilizado para especificar información sobre el propio documento, como el conjunto de caracteres, la descripción de la página, las palabras clave, el autor del documento… Suele llevar dos atributos: *name* y *content*, que hacen referencia al nombre de la página y a sus principales contenidos
<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 caso de que sea externo
<base>	Se usa para especificar la URL base a partir de la que partirán el resto de URL relativas del documento. Puede indicar la base de elementos como sonidos, gráficos, etc., siempre que hagan referencia a una determinada página web

Otro uso diferente de la etiqueta *<meta>* es el refresco automático, que, transcurrido un tiempo estimado, pasa a actualizar la misma página. Es 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. Un ejemplo de documento HTML con la cabecera completa es el siguiente:

```
<html>
    <head>
        <title> Ejemplo de cabecera </title>
        <meta name = “description” content = “Página principal ILERNA con alumnos y profesores”>
        <meta name = “keywords” content = “Nombre y apellidos de alumnos matriculados”>
        <script type= “text/javascript” src=”/js/archive.js”> </script>
        <link rel="stylesheet" type="text/css" href="hoja_de_estilos.css">
        <base href="https://www.ilernaonlineextra.com/" target="_blank">
        <style type =”text/css”>
            body{
                margin-left:40px;
            }
        </style>
    </head>
    <body>
        (aquí se colocaría todo el contenido)
    </body>
</html>
```

En este ejemplo hemos incluido URL externas no reales, además de una simulación de enlace externo a un archivo JavaScript, que se encontraría en una carpeta llamada *js*.

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

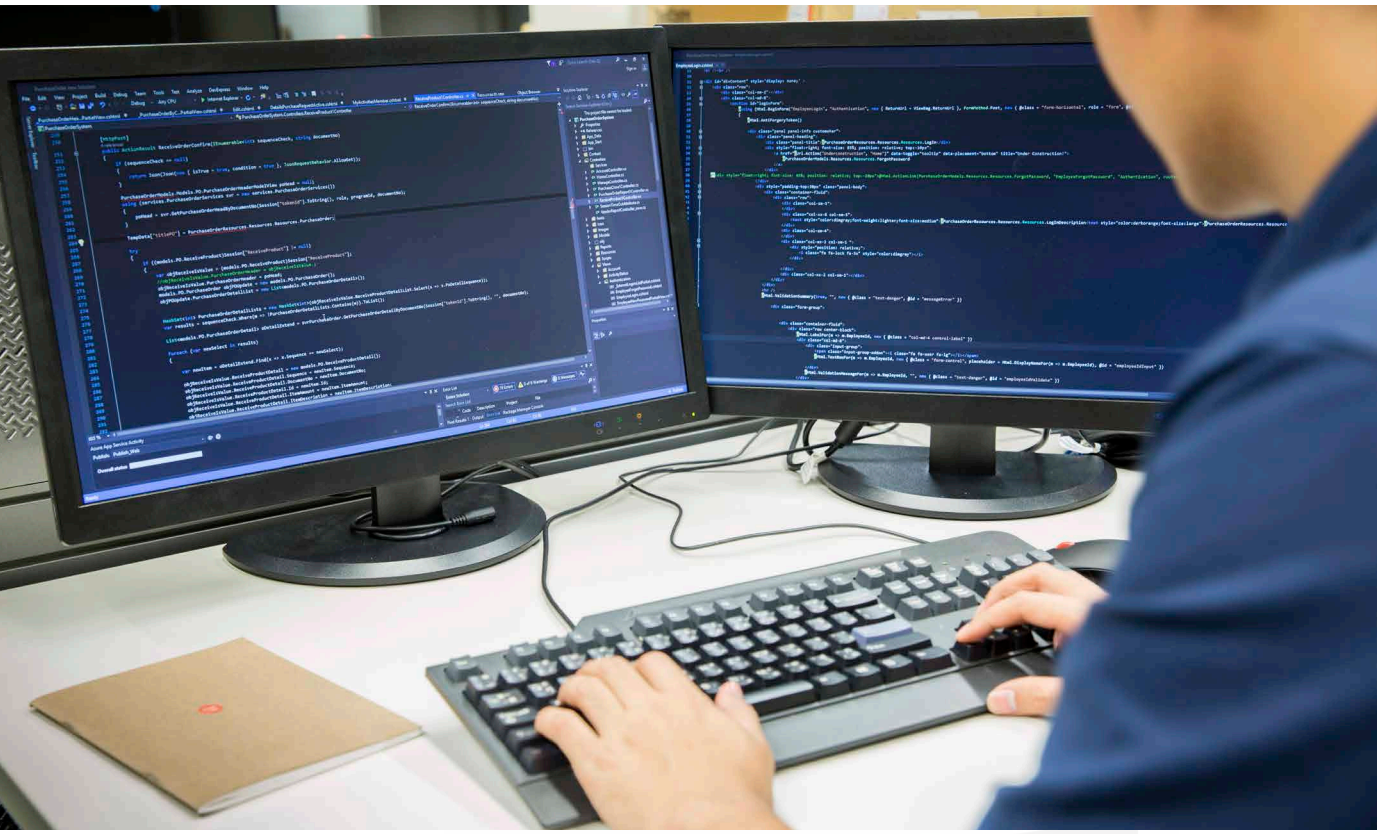
Cuerpo del documento

El cuerpo del programa se encuentra entre:

```
<body>
…
</body>
```

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

Existen un conjunto de atributos opcionales de *<body>* que permiten realizar diferentes configuraciones sobre la apariencia de un documento. Sin embargo, hoy están obsoletos, ya que HTML5 no los soporta. Algunos de ellos son: *bgcolor, text, link* y *vlink*.



Luego hay a algunos atributos que se recomienda utilizar en la marca *<body>*. Son los siguientes:

- **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 que lo puedan visualizar los navegadores.
- **style:** aplicar un estilo a este elemento.

A continuación, veremos todas las marcas que podemos utilizar en esta parte del documento para poder aplicar distintos tipos de comandos y así confeccionar la página web correspondiente.

Podemos clasificar los elementos del cuerpo *(<body>)* en dos tipos: elementos de bloque y elementos de línea.

Los **elementos de bloque** poseen las siguientes características:

- Comienzan en una nueva línea
- Ocupan todo el ancho de una línea disponible
- Por tanto, impiden que otros elementos se sitúen a su lado (horizontalmente)
- Pueden albergar en su interior otros elementos de bloque u otros elementos de línea

- Algunos elementos de bloque típicos son los siguientes: * *, *<form>*, *<h1>*, *<p>*, *<section>*, *<table>*...

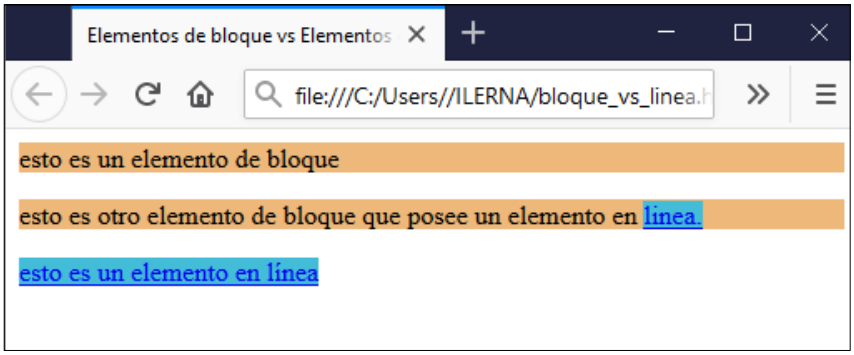
Por su parte, los **elementos de línea** tienen las siguientes características:

- Solo pueden contener en su interior datos y otros elementos en línea
- Ocupan el ancho mínimo (en horizontal) que necesitan
- Los elementos de línea más empleados son: *<a>*, *<code>*, *<img>*, *<input>*, **...

A continuación, vamos a ilustrar la diferencia entre un elemento de bloque, como el de *párrafo (<p>)* y un elemento de línea, como el de *hiperenlace (<a>)*. En el siguiente código HTML indicamos en la cabecera *(<head>)* un código CSS que indica que los elementos de párrafo *(<p>)* tendrán un color de fondo anaranjado y los elementos de hiperenlace *(<a>)* tendrán un color de fondo celeste, de modo que podamos diferenciar bien ambos elementos.

```
<html>
    <head>
        <title> Elementos de bloque vs Elementos de línea </title>
          <style type="text/css">
            p {
                background-color: #edb879
            }
            a {
                background-color: #44bcd8
            }
  </style>
    </head>
    <body>
        <p> esto es un elemento de bloque </p>
        <p> esto es otro elemento de bloque que posee un elemento en <a href="www.google.com"> linea. </a> </p>
        <a href= "www.google.com"> esto es un elemento en línea </a>
    </body>
</html>
```

A partir del código anterior, si visualizáramos el resultado en un navegador, en este caso *Firefox*, el resultado sería el siguiente:



Podemos comprobar que el elemento de bloque *<p>* ocupa todo el ancho de línea del que dispone, y, sin embargo, el elemento de línea *<a>* solo ocupa lo imprescindible para mostrar el texto asignado.

Encabezados

Para poner títulos en el contenido de un HTML podemos usar los encabezados (también llamados cabeceras).

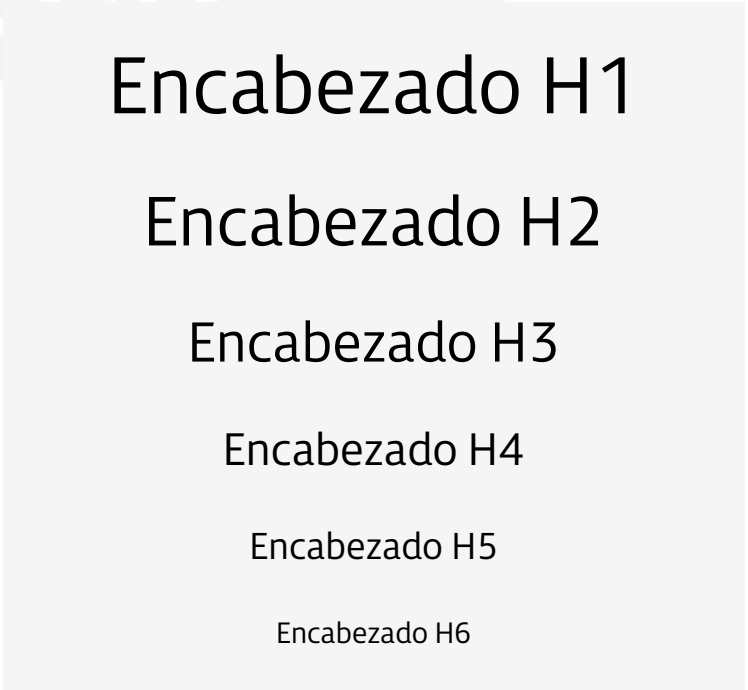
Existen seis tipos diferentes de encabezados (no hay que confundir los encabezados con la cabecera *<head>*. En inglés se denomina *headings* a los primeros y *head* a la segunda). Las marcas de apertura y cierre de cada encabezado son obligatorias.

Los encabezados que podemos usar son los siguientes:

```
<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 llegar al de *h6*, que será el menor.

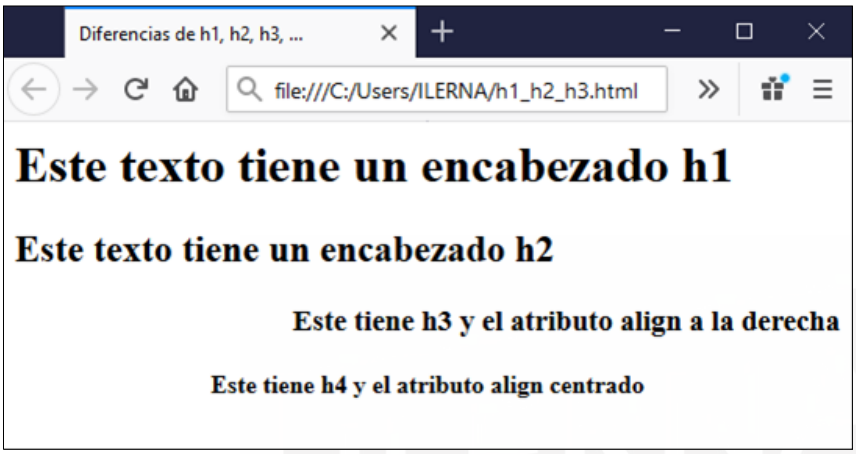
En los encabezados podemos usar el atributo *align* para alinear el texto de la cabecera al centro, a la izquierda o a la derecha, como podemos observar en el siguiente ejemplo.

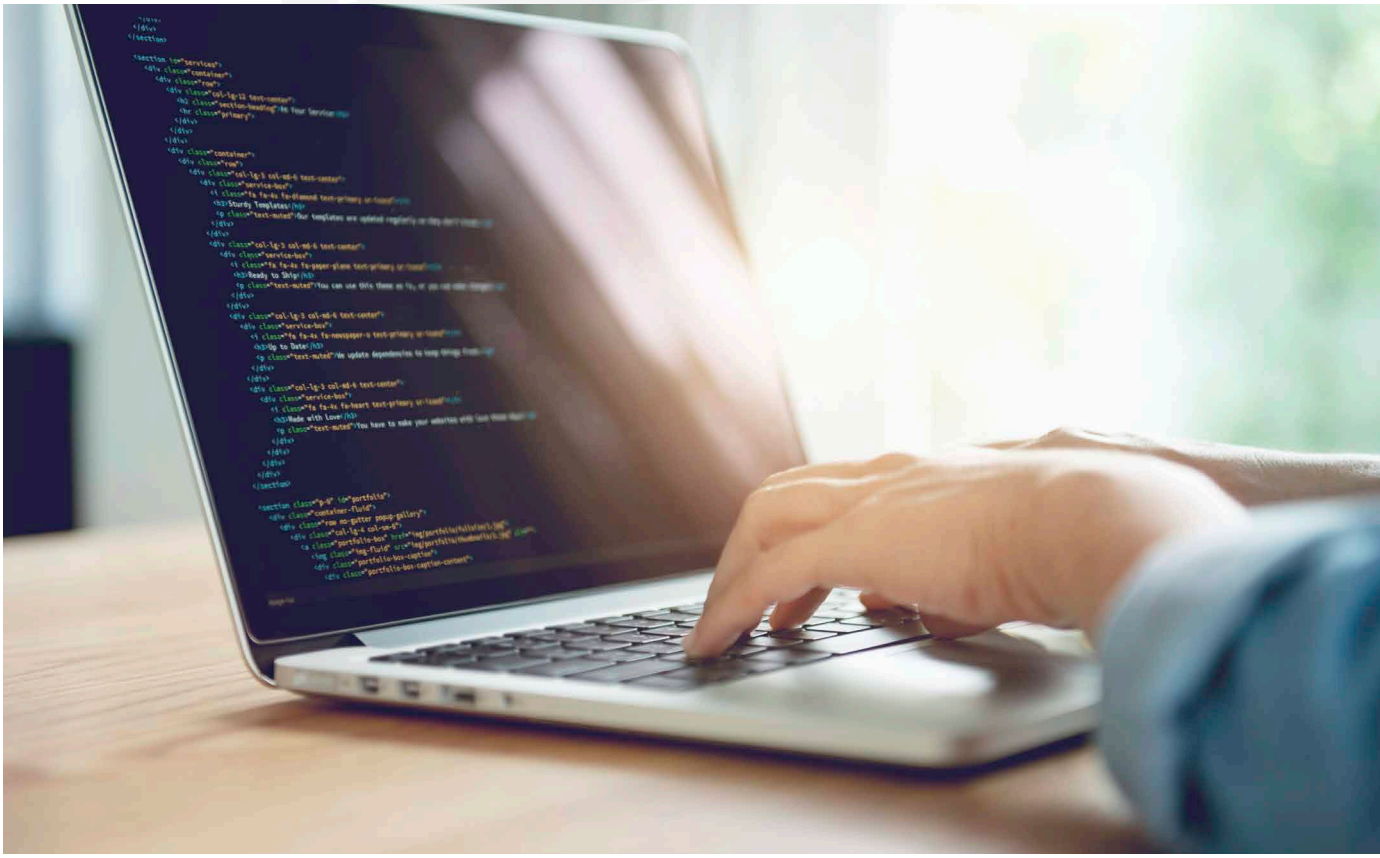


Aquí tenemos el código HTML donde establecemos cuatro líneas de contenido en el *body*, todas con un encabezado distinto y algunas con el atributo *align*, con que tiene un valor específico:

```
<html>
    <head>
        <title> Diferencias de h1, h2, h3, ... </title>
  </style>
    </head>
    <body>
        <h1> Este texto tiene un encabezado h1 </h1>
        <h2> Este texto tiene un encabezado h2 </h2>
        <h3 align="right"> Este tiene h3 y el atributo align a la derecha </h3>
        <h4 align="center"> Este tiene h4 y el atributo align centrado </h4>
    </body>
</html>
```

De este modo, la visualización en el navegador Firefox del código HTML anterior sería la siguiente:





Párrafos

Son una de las marcas más utilizadas. No pueden contener elementos de bloque en su interior. Todos los párrafos del documento web serán bloques delimitados por la marca *p*.

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

- ** :** escribe un texto en forma de párrafo.
- **<blockquote> </blockquote>:** permite visualizar una cita con el margen izquierdo mayor, produciéndose un efecto de una sangría.
- **<pre> </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.

Saltos de línea

- **
:** representa un salto de línea entre párrafos, ya que el salto de línea del teclado es ignorado por 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.*

Estilo de texto

Son una serie de elementos HTML que proporcionan elementos de estilo a una parte del texto. Así, pueden poner el texto en negrita, en cursiva o subrayarlo (*, y <u></u>*).

	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 *#* 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 es *#RRVVAA*.

Los diferentes pares de cifras hexadecimales van a oscilar entre *00* y *FF* proporcionando un rango que va de 0 hasta 255 valores diferentes.

Otra forma de expresar colores en HTML es usando una notación hexadecimal más corta, que utiliza un dígito para cada color (#JVA). En este caso, el rango de valores oscilará entre 0 y 15. Si lo preferimos, también 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, los podemos definir como enlaces que apuntan a otro sitio web, un fichero, una imagen, etc.

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

```
<a></a>
```

De este modo, 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, este debe aparecer subrayado y de otro color.

PARA + INFO

El atributo *href* es el que nos ofrece la posibilidad de crear un hiperenlace. Debemos indicar una URL, que será aquella a la que queramos acceder al clicar en el hiperenlace. **A continuación, si en el elemento <a> no indicamos el atributo href, entonces el elemento representará un marcador de posición que será al que referencie otro.**

Disponemos de otro atributo *target* (opcional) que hará referencia al destino en el que se mostrará la información disponible en esa dirección a la que nos lleva. Por ejemplo, si deseamos que el enlace (l*ink*) se abra en una pestaña o página nueva, pondremos *_blank* como valor del atributo *target,* pero si deseamos que se abra en la misma pestaña o ventana, pondremos *_self*.

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 absolutas y relativas

Las rutas **absolutas** son 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.

Las **relativas** se corresponden con aquellos enlaces cuya dirección relativa se indica en el atributo *href* del comando *a*. Suelen ser enlaces a páginas internas del 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.

No obstante, existen otros tipos de archivos que no se consideran imágenes, como por ejemplo, los 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 indicar un texto alternativo que sea capaz de representar el contenido de una imagen. Podemos utilizar esta forma en caso de que el navegador no pueda visualizar o descargar las distintas imágenes.

Las etiquetas ***<figure>*** y ***<figcaption>*** son novedosas para HTML5. Nos ofrecen 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 primer 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

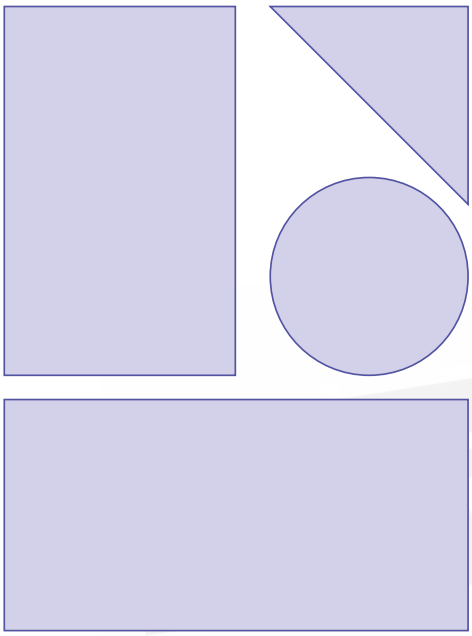
Los mapas de imagen nos permiten definir distintas zonas clicables dentro de una imagen, de tal forma que el usuario puede hacer clic sobre las zonas definidas y, cada una de estas, puede apuntar a una URL distinta.

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:

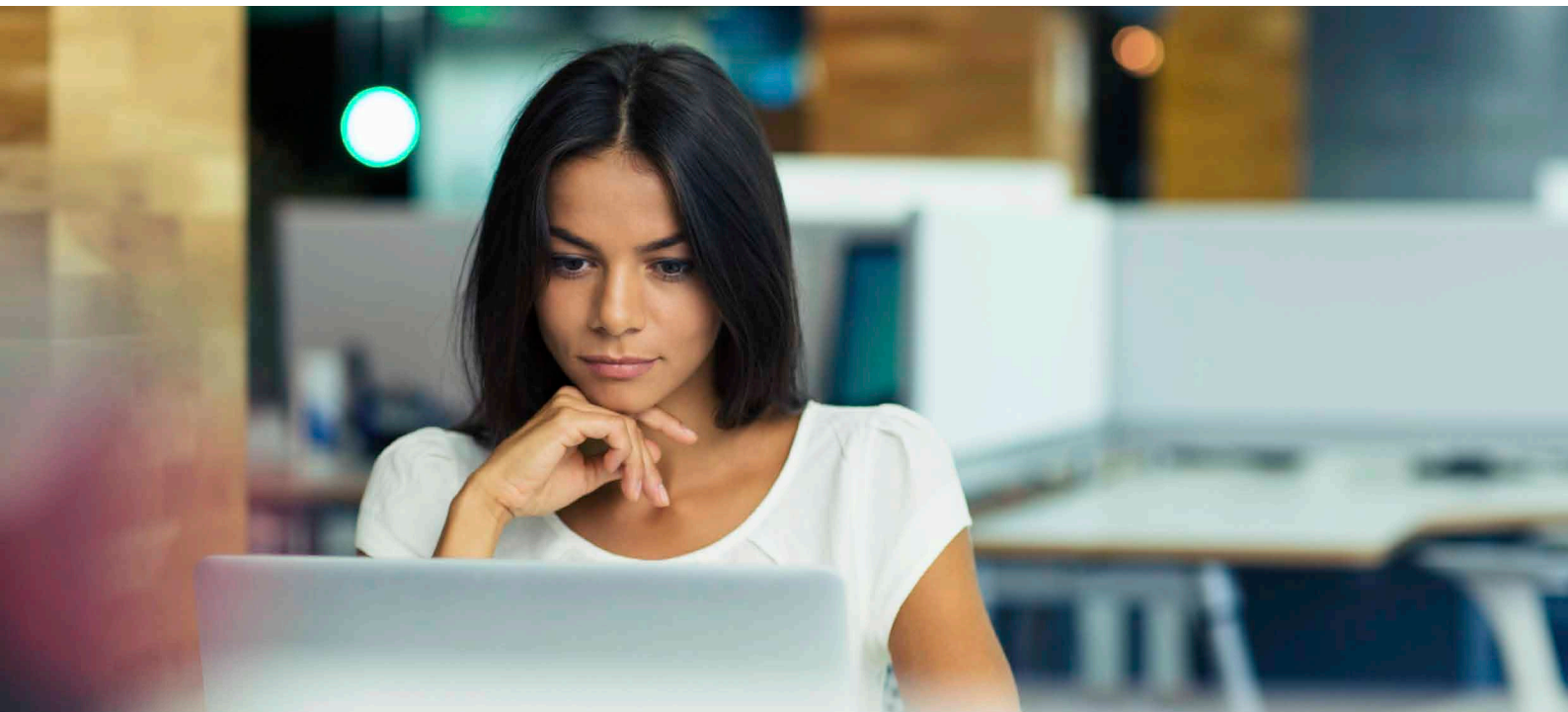
- En primer lugar, insertamos la imagen original mediante la etiqueta ***<img>.***
- A continuación, utilizamos la etiqueta ***<map>*** para definir las distintas zonas o regiones de la imagen. Definiremos cada una de estas zonas con la etiqueta ***<area>.***

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



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



<a>	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 coordenadas X e Y del vértice inferior derecho). Circular = X1, Y1, R (coordenadas X e Y del centro y radio del círculo). Poligonal = 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.

```
<imgsrc="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

Las listas en HTML permiten agrupar un conjunto de elementos o ítems relacionados en un documento.

- **Listas ordenadas:** sirven para representar los elementos de una lista de manera ordenada, usando una numeración incremental. Por defecto, dicha numeración será con números tradicionales y empezando por el 1 (1,2,3,4,...) pero usando el atributo *type* o *start* podríamos modificar la numeración. La sintaxis de una lista ordenada es la siguiente:

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

Y se muestra de la siguiente forma:

1. Apartado 1
2. Apartado 2

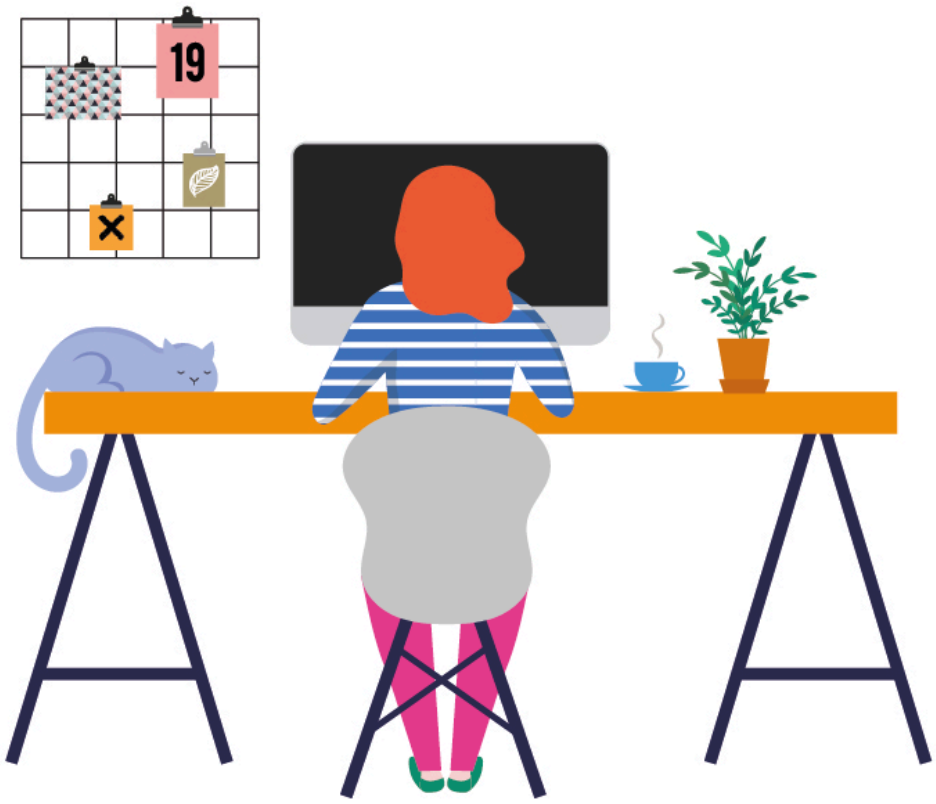
…

Como ya adelantamos, la etiqueta ***<ol>*** tiene disponibles los siguientes atributos:

- ***start*** = numero: permite seleccionar el número que deseemos que sea el primero de la lista
- ***type*** = tipo: permite seleccionar el tipo de numeración que deseemos

En HTML5, podemos diferenciar entre los siguientes tipos:

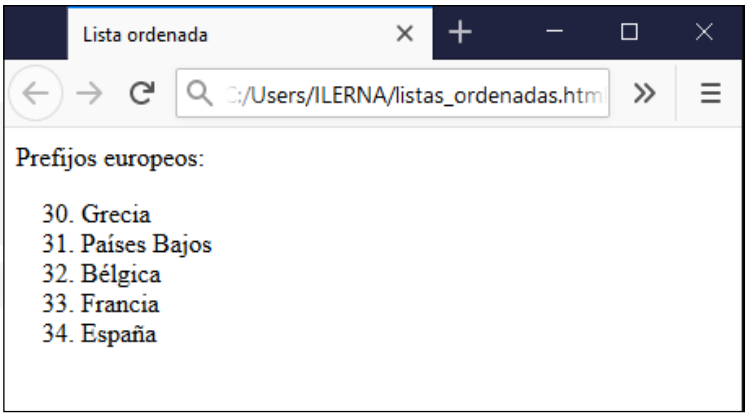
- **1**: expresa números desde 1, 2, 3, etc.
- **a**: expresa letras minúsculas a partir de a, b, c, etc.
- **A**: expresa letras mayúsculas a partir de A, B, C, etc.
- **i**: expresa números romanos desde i, ii, iii, etc.
- **I**: expresa números romanos en mayúscula desde I, II, III, etc.



Un ejemplo de código HTML con el uso de una lista ordenada sería el siguiente:

```
<html>
    <head>
        <title> Lista ordenada </title>
  </style>
    </head>
    <body>
        <p> Prefijos europeos: </p>
        <ol type= "1" start="30">
            <li>Grecia</li>
            <li>Países Bajos</li>
            <li>Bélgica</li>
            <li>Francia</li>
            <li>España</li>
        </ol>
    </body>
</html>
```

Y la visualización en un navegador, en este caso Firefox, sería la siguiente:



- **Listas no ordenadas:** ofrecen la posibilidad de poder representar los diferentes elementos de una lista sin enumerar, independientemente del lugar en el que se vayan a almacenar. A diferencia de las listas ordenadas, en este tipo de listas se usa una viñeta para marcar cada ítem. La sintaxis de una lista no ordenada es la siguiente:

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

Por ser una lista que 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.

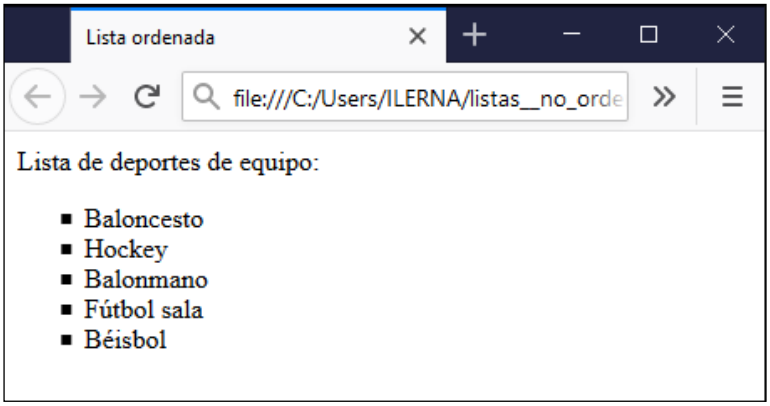
En HTML5, podemos diferenciar entre los siguientes tipos:

- **Disc**: expresa una viñeta en forma de disco.
- **circle**: expresa una viñeta en forma de círculo.
- **square**: expresa una viñeta en forma de cuadrado.

Un ejemplo de código HTML con el uso de una lista no ordenada sería el siguiente:

```
<html>
    <head>
        <title> Lista ordenada </title>
  </style>
    </head>
    <body>
        <p> Lista de deportes de equipo: </p>
        <ul type= "square">
            <li>Baloncesto</li>
            <li>Hockey</li>
            <li>Balonmano</li>
            <li>Fútbol sala</li>
            <li>Béisbol</li>
        </ul>
    </body>
</html>
```

Y la visualización en un navegador, en este caso Firefox, sería la siguiente:



- **Listas de definición o glosario**: ofrecen la posibilidad de 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.

```
<div></div>
```

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.

La etiqueta básica de HTML para incluir una tabla en un documento es la siguiente:

```
<table>… </table>
```

Filas, columnas y celdas

Ahora bien, cada tabla está formada por una serie de filas y una serie de columnas. Por tanto, para poder incluir una tabla en un documento HTML debemos usar una estructura de etiquetas, cuya sintaxis general es la siguiente:

```
<table>
<tr>
<th> … </th>
<th> … </th>
…
</tr>
<tr>
<td> … </td>
<td> … </td>
…
</tr>
<tr>
<td> … </td>
<td> … </td>
…
</tr>
…
</table>
```

La etiqueta *<tr>* hace referencia a *fila* (*r* de *row*, en inglés).

La etiqueta *<th>* hace referencia a las cabeceras de las columnas (*h* de *head*, en inglés).

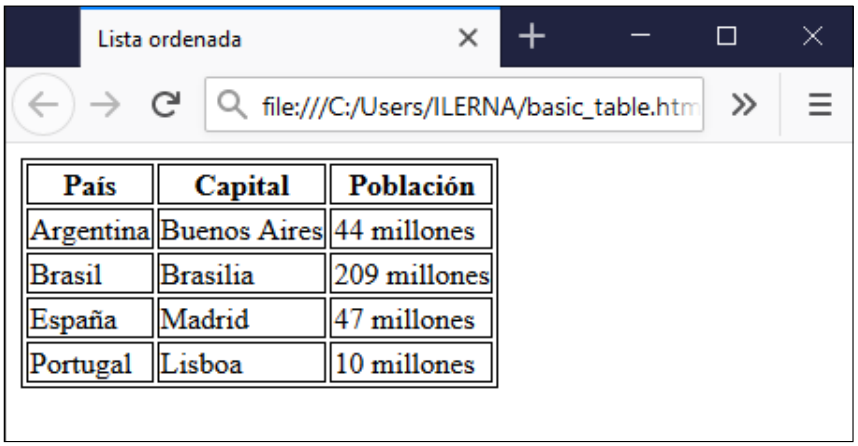
La etiqueta *<td>* hace referencia a los datos que pondremos en cada celda (*d* de *data*, en inglés).

Debe haber una correlación entre el número de cabeceras (*<th>*) que pongamos en la tabla con el número de columnas en cada fila (<td>). Es decir, si hay 5 cabeceras es porque habrá 5 columnas, por tanto si en la primera fila hay 5 elementos *<th>*, debería haber 5 elementos *<td>* en cada fila.

Vamos a entender lo expuesto con el siguiente ejemplo:

```
<html>
    <head>
        <title> Lista ordenada </title>
        <style>
            table, th, td { border: 1px solid black;}
        </style>
    </head>
    <body>
        <table>
          <tr>
            <th>País</th>
            <th>Capital</th>
            <th>Población</th>
          </tr>
          <tr>
            <td>Argentina</td>
            <td>Buenos Aires</td>
            <td>44 millones</td>
          </tr>
          <tr>
            <td>Brasil</td>
            <td>Brasilia</td>
            <td>209 millones</td>
          </tr>
          <tr>
            <td>España</td>
            <td>Madrid</td>
            <td>47 millones</td>
          </tr>
          <tr>
            <td>Portugal</td>
            <td>Lisboa</td>
            <td>10 millones</td>
          </tr>
        </table>
    </body>
</html>
```

Su visualización en un navegador, en este caso Firefox, sería la siguiente:



País	Capital	Población
Argentina	Buenos Aires	44 millones
Brasil	Brasilia	209 millones
España	Madrid	47 millones
Portugal	Lisboa	10 millones

Como podemos observar, en la tabla hay 3 columnas, por tanto en el código HMTL podemos observar que hay 3 elementos *<th>* en la primera fila y 3 elementos *<td>* en cada una de las siguientes filas.

Combinación de celdas

Es posible combinar un conjunto de celdas de una determinada tabla, 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.

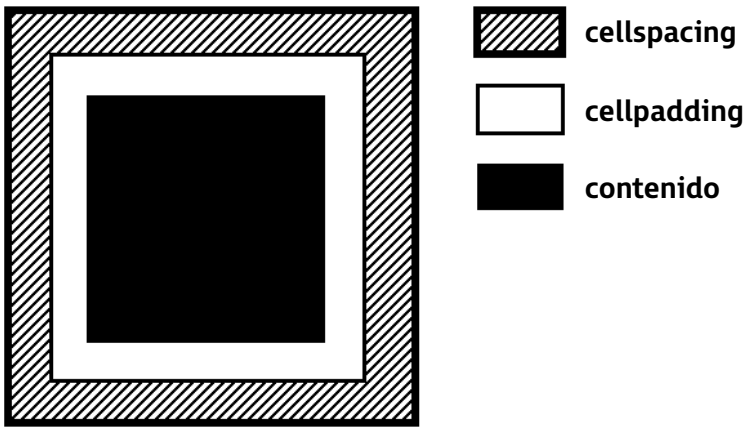
Atributos de tabla

Hasta el momento, y para versiones anteriores a HTML5, se han podido utilizar distintos atributos, como *alignybgcolor*. Sin embargo, desde la aparición de HTML5, se soportan todos estos 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.

En la siguiente imagen podemos observar la diferencia entre *cellspacing* y *cellpadding*:



Por otro lado, tenemos el elemento de tabla:

```
<caption>
---
</caption>
```

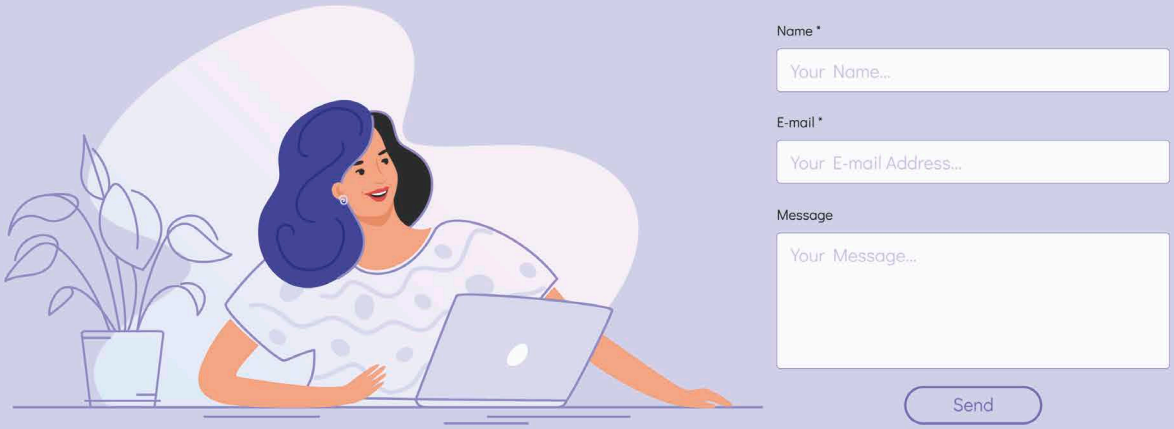
Lo usamos para poner un título a la tabla- Es decir, la información que pongamos entre las etiquetas de *<caption> ... </caption>* aparecerá justo arriba de la tabla, no en el interior, sino encima de la tabla.

Formularios

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

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

Toda la información que recogen los formularios debe tratarse por archivos que tienen que ser implementados por el propio desarrollador. De esta forma, los almacenaremos en bases de datos diseñadas previamente. No obstante, no las detallaremos en el manual, puesto que solo nos centraremos en la definición y diseño de los formularios.



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

Propiedades de los formularios

Para la sintaxis básica de un formulario utilizamos la siguiente etiqueta:

```
<form>
…
</form>
```

Por tanto, dentro de esta etiqueta podemos implementar todos los elementos que hemos nombrado previamente (checkbox, textbox…).

La etiqueta *form* consta de varios atributos bastante importantes. Podemos encontrar los siguientes:

- **action**: indica el lugar al que se envían los datos.
- **method**: indica el 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.
- **target**: 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 el formulario mediante el nombre. De esta forma, podemos llamarlo desde otro archivo. Recomendamos que el nombre de formulario sea único para facilitar su tratamiento.

Elementos de los formularios

Dentro de un formulario podemos tener distintos elementos que van a componer todo el diseño y contenido del formulario. A continuación, vamos a listar los principales elementos indicando la sintaxis de sus etiquetas y la explicación de su uso:

- **Elemento *fieldset***. Podríamos traducirlo como conjunto de campos. Su sintaxis es la siguiente:

```
<fieldset> …</fieldset>
```

  Con este elemento podemos agrupar un conjunto de elementos bajo un mismo nombre.

- **Elemento *input***. Su sintaxis es la siguiente:

```
<input>…</input>
```

  Nos permite crear varios tipos de elementos dentro de un formulario, dependiendo del valor que asignemos al atributo *type*, pero lo primero que recomendamos dentro de esta etiqueta es especificar el valor del atributo *name*, ya que de esta forma podemos identificar el elemento. Seguidamente, ahora sí, mediante el atributo type, elegiremos el tipo de elemento que deseamos tener en el formulario.

- **Elemento *textarea:***

```
<textarea>… </textarea>
```

  Podemos definir un campo de texto de grandes dimensiones para que el usuario tenga la posibilidad de escribir todo lo deseado sin ningún tipo de limitación. Este elemento se utiliza en muchos formularios cuando escribimos en un campo de observaciones.

  Los atributos de este elemento que podemos incluir en la etiqueta de apertura son 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.

- **Elemento *button*:**

```
<button type="…" > … </button>
```

Se traduce como *botón*. Es la forma que tenemos de insertar un botón en el formulario. Aunque también podemos hacerlo con el comando *input*. Con el atributo *type* indicamos al navegador qué tipo de botón deseamos: (*submit* | *reset* | *button*).

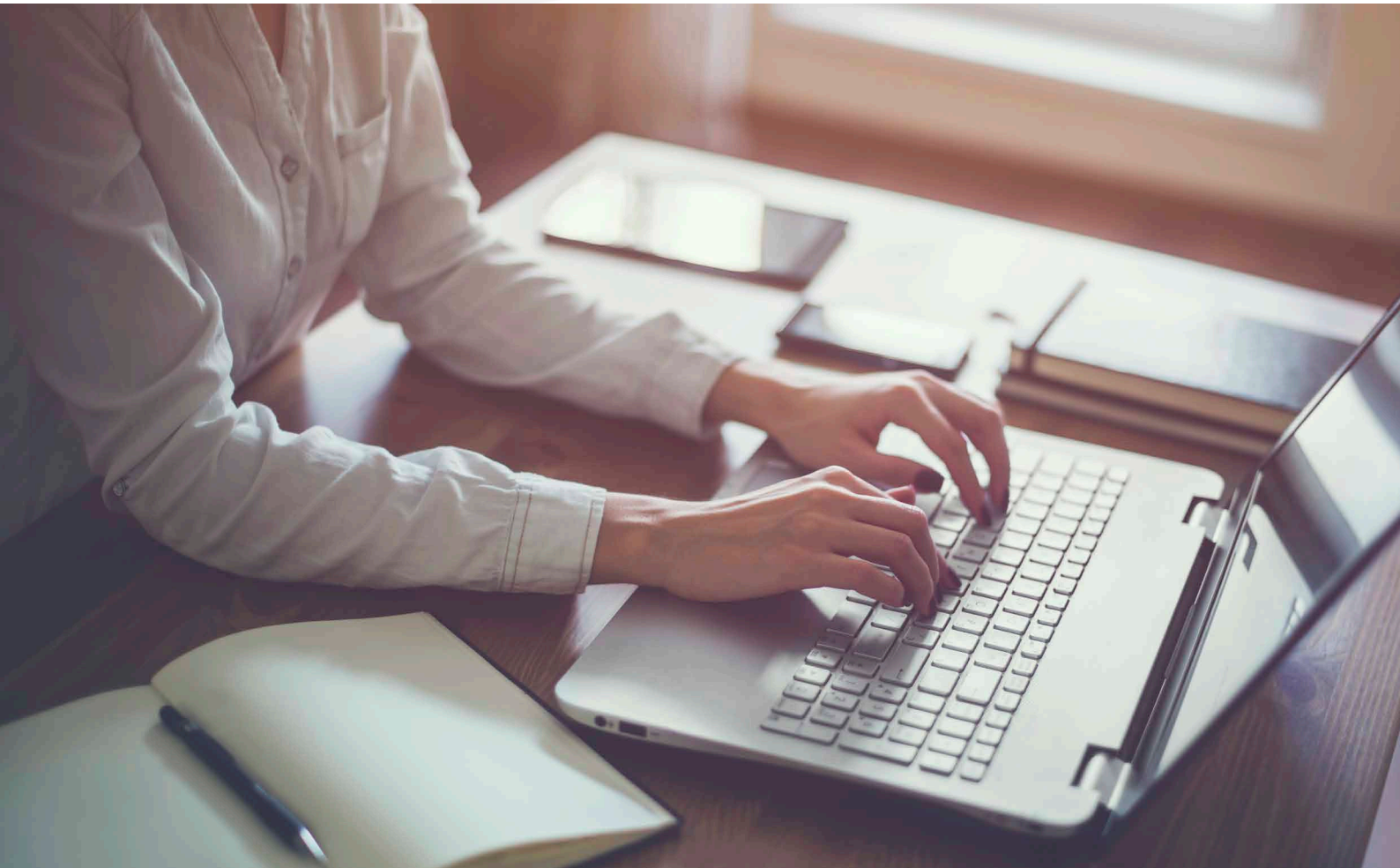
- **Elemento *select*:**

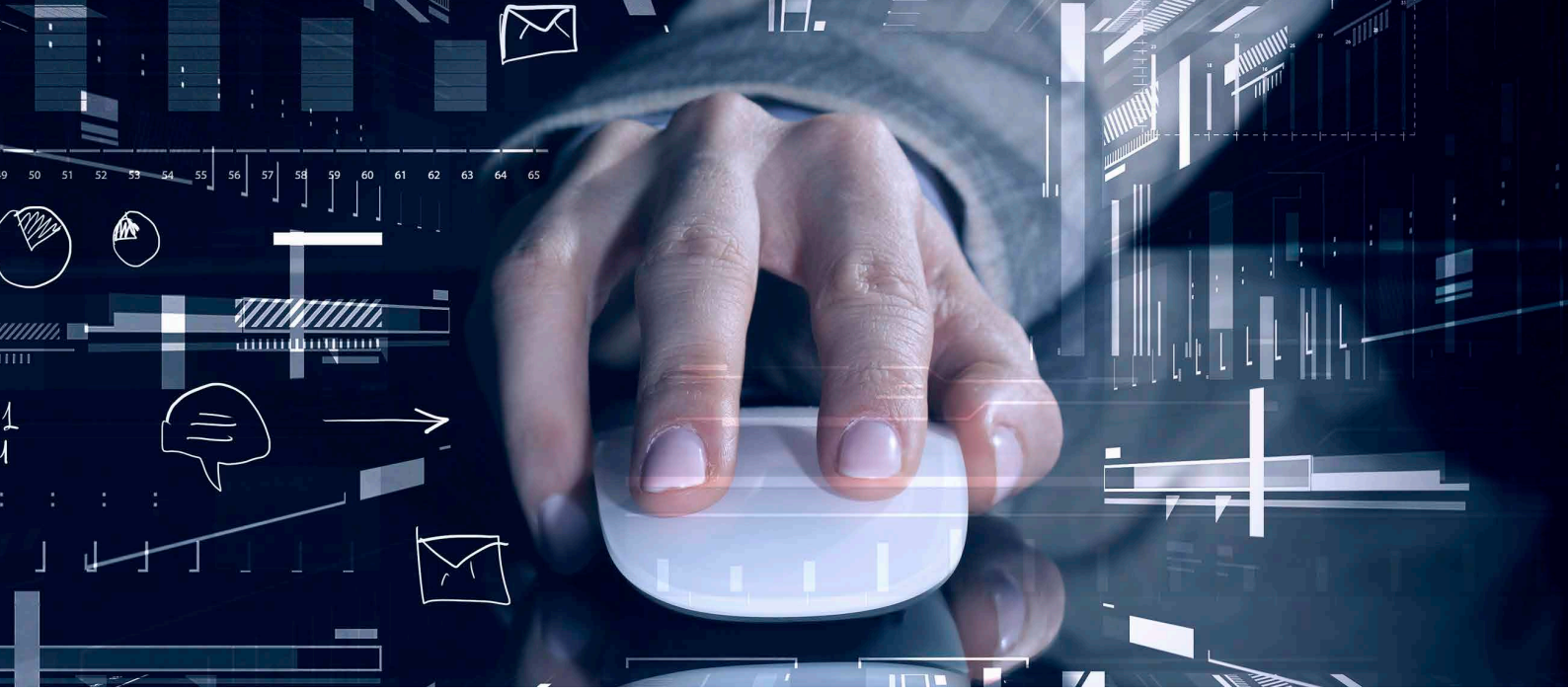
```
<select name = "…" id = "…" >
<option value = "…" > …</option>
<option value = "…" > …   </option>
…
</select>
```

Es el elemento que nos permite definir una lista desplegable con varios ítems, es decir, con varias opciones a elegir. Para ello, incluiremos tantas etiquetas *option* como opciones deseemos tener en nuestra lista.

El atributo *name* nos ayudará a tener una referencia de la información del formulario una vez este haya sido rellenado y validado.

El atributo *id* nos permite enlazar la lista desplegable con una *label*.





- **Elemento *label*:**

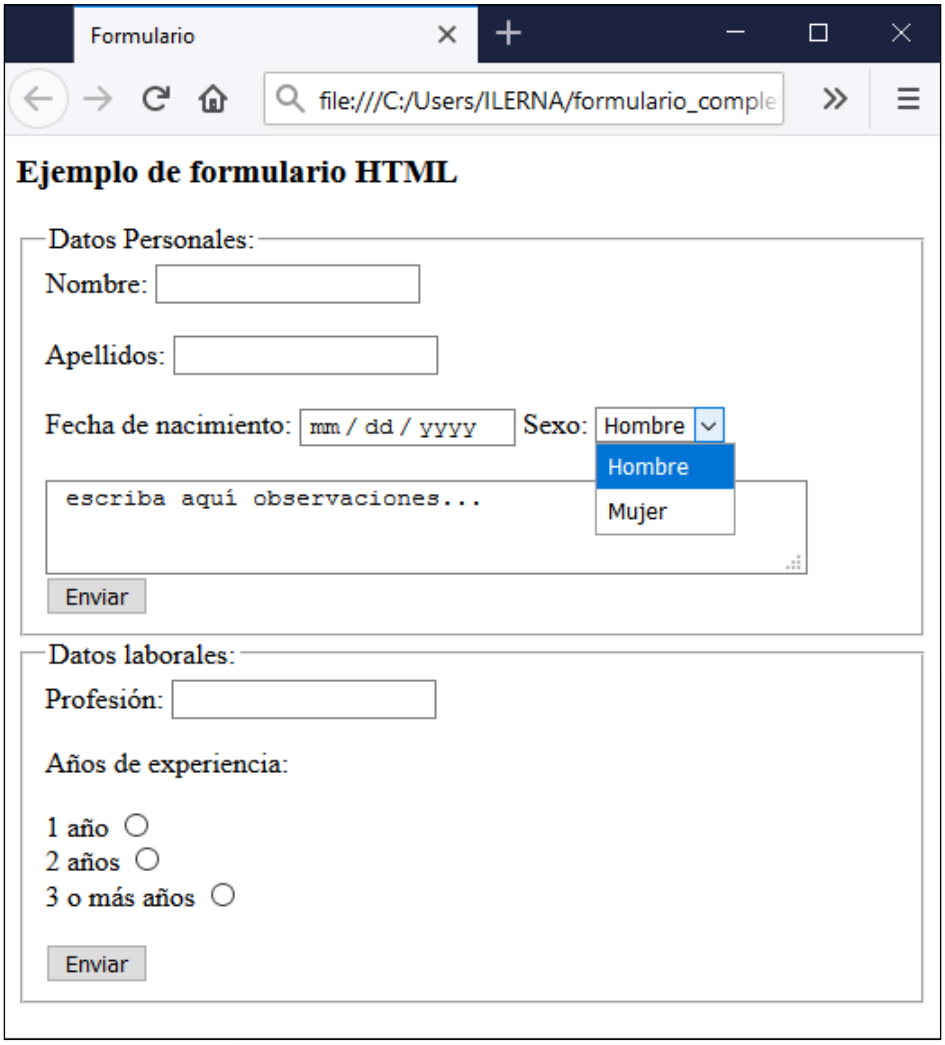
```
<label  for="…"> … </label>
```

Con este elemento relacionamos un texto o título a un determinado campo del formulario, por ejemplo, a un input. Dentro del atributo *for* indicaremos el *id* del elemento que queremos relacionar con ese *label*.

A continuación, mostramos un ejemplo de formulario completo, que hemos dividido en dos secciones (dos *fieldset*) con todos los elementos que hemos visto hasta ahora:

```
<html>
    <head>
        <title> Formulario  </title>
    </head>
    <body>
        <h3> Ejemplo de formulario HTML </h3>
        <form action="/script.php" method="get" target="_blank" name="Mi_formulario">
            <fieldset form="Mi_formulario" id ="personales">
                <legend> Datos Personales: </legend>
                <label for="Elnombre">Nombre:</label>
                <input type="text" id="Elnombre" name="Elnombre"><br><br>
                <label for="LosApellidos">Apellidos:</label>
                <input type="text" id="LosApellidos" name="LosApellidos"><br><br>
                <label for="Nacimiento">Fecha de nacimiento:</label>
                <input type="date" id="Nacimiento" name="Nacimiento">
                <label for="sexo">Sexo:</label>
                <select name="sexo" id="sexo">
                    <option value="hombre">Hombre</option>
                    <option value="mujer">Mujer</option>
                </select><br><br>
                <textarea name="masinfo" rows="2" cols="50"> escriba aquí observaciones... </textarea><br>
                <button type="submit" form="Mi_formulario" value="enviar">Enviar</button>
            </fieldset>
            <fieldset form="Mi_formulario" id ="laborales" align="center">
                <legend> Datos laborales: </legend>
                <label for="profesion">Profesión:</label>
                <input type="text" id="profesion" name="profesion">
                <p> Años de experiencia: </p>
                <label for="1">1 año</label>
                <input type="radio" name="experiencia" id="1" value="1"><br>
                <label for="female">2 años </label>
                <input type="radio" name="experiencia" id="2" value="2"><br>
                <label for="other">3 o más años</label>
                <input type="radio" name="experiencia" id="3" value="3"><br><br>
                <button type="submit" form="Mi_formulario" value="enviar">Enviar</button>
            </fieldset>
        </form>
    </body>
</html>
```

La visualización del formulario anterior en un navegador. En Firefox sería la siguiente:



Debemos tener en cuenta, que aunque haya dos elementos *fieldset* en este formulario de ejemplo, incluso con un botón de envío cada uno, el funcionamiento general de envío no se verá afectado. Es decir, al hacer clic en cualquiera de los dos botones de envío, el formulario se enviará al completo, con toda la información de ambos *fieldset* al destino que se haya indicado como un único conjunto de datos.

Marcos

Definición de marcos (*frames*)

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

También tenemos la opción de que estos *frames* interactúen. Cuando presionamos sobre un enlace en un *frame* para acceder a él, podemos 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.

Uso de marcos

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

```
<html>
<head>
<title>Prueba de frames</title>
</head>
<frameset cols="20%,80%">
<framesrc="página2.html">
<framesrc="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. 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.

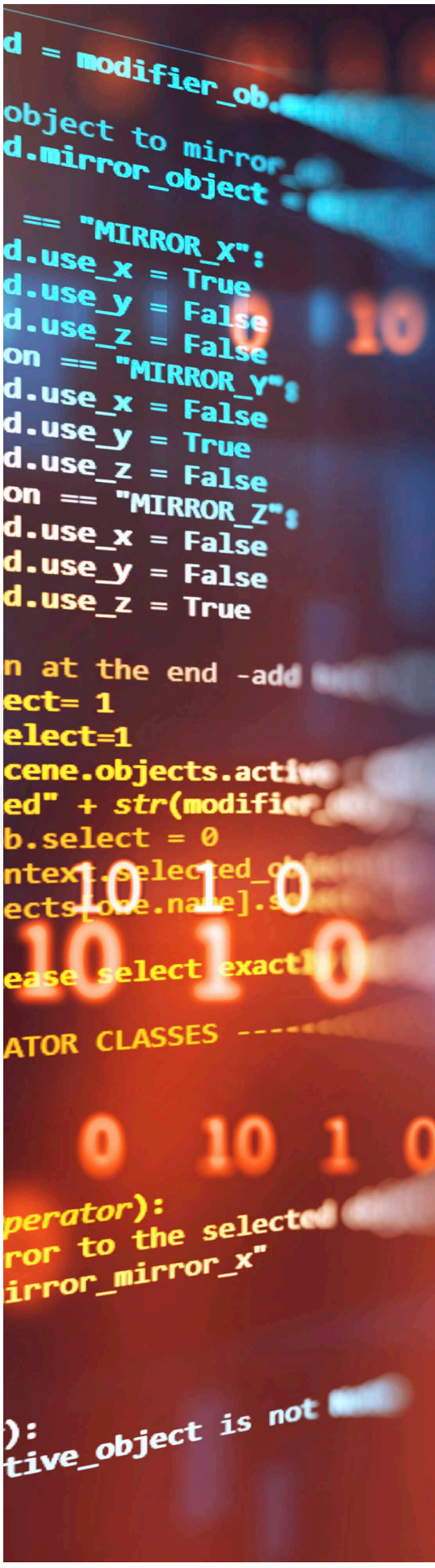
Además de este atributo, también podemos detallar, en los atributos *height* y *width*, tanto el ancho como la altura del objeto.

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:

```
<paramname=”Nombre del parámetro” value =
“Valor del parámetro”>
```



ponte a prueba

¿Cuál de estas etiquetas muestra el texto más grande VISUALMENTE en HTML?

a) <h2>Texto MUY GRANDE </h2>

b) <bigFont> Texto MUY GRANDE </bigFont>

c) <h12>Texto MUY GRANDE </h12>

d) <h1>Texto MUY GRANDE </h1>

¿Cómo se escribe un texto en forma de párrafo mediante el lenguaje HTML?

a) <p>

b)

c) <tr>

d) <td>

¿Cómo se llama el atributo que nos ofrece la posibilidad de crear un hiperenlace? Debemos indicar una URL que va a ser la que queramos acceder al hacer clic en el hiperenlace.

a) href

b) <a>

c) <u>

d) <i>

2.5. XHTML Y SUS DIFERENCIAS CON HTML

XHTML es una derivación del lenguaje de marcas XML. Podríamos decir que XHTML es una versión de HTML bajo la sintaxis de XML.

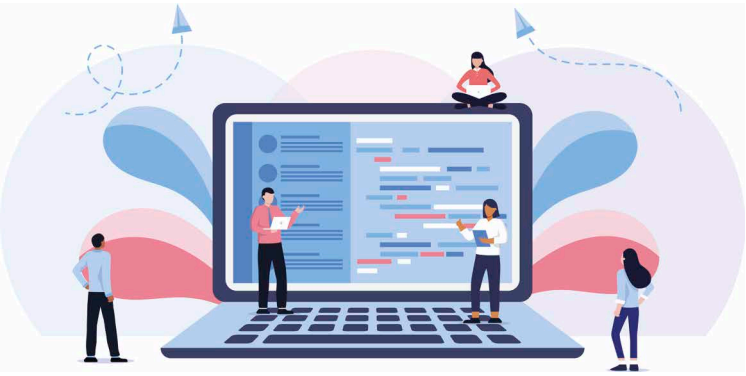
Los documentos XHTML son documentos bien formados que cumplen todas las normas sintácticas y estructurales de los documentos XML, algo que no ocurre con HTML.

```
<?xmlversion="1.0" encoding="UTF-8">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "https://www.w3.org/
TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns= http://www.w3.org/1999/xhtml>
    <head>
    <title> Aquí ponemos el título del documento
    </head>
    <body>
    <p> contenido del documento </p>
    </body>
</html>
```

A continuación, un ejemplo de documento XHTML:

Las diferencias más notables entre XHTML y HTML son:

- La etiqueta *<!DOCTYPE...>* y el espacio de nombres son obligatorios en XHTML, en HTML no.
- También son obligatorios los elementos *<html>* *<head>* *<title>* y *<body>*.
- En XHTML los elementos deben estar estrictamente bien anidados.
- Toda etiqueta que se abra en XHTML debe ser cerrada, siempre.
- Los elementos y atributos en XHTML deben estar siempre en minúsculas.
- Los valores de los atributos en XHTML deben estar siempre entre comillas.



2.6. Versiones de HTML y XHTML

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 HTML que pueden utilizar.

HTML llegó a convertirse en estándar en 1995 y, con el paso de los años, ha experimentado un desarrollo constante. Hace algunos años, la versión de HTML recomendada por el W3C era HTML 4.01 y ya llegó a desarrollarse la versión HTML5.

Versiones de HTML

HTML 1	Inicios a partir de 1991
HTML 2	Noviembre 1995
HTML 3	Enero 1997
HTML 4	Diciembre 1997
HTML 5	Octubre 2014

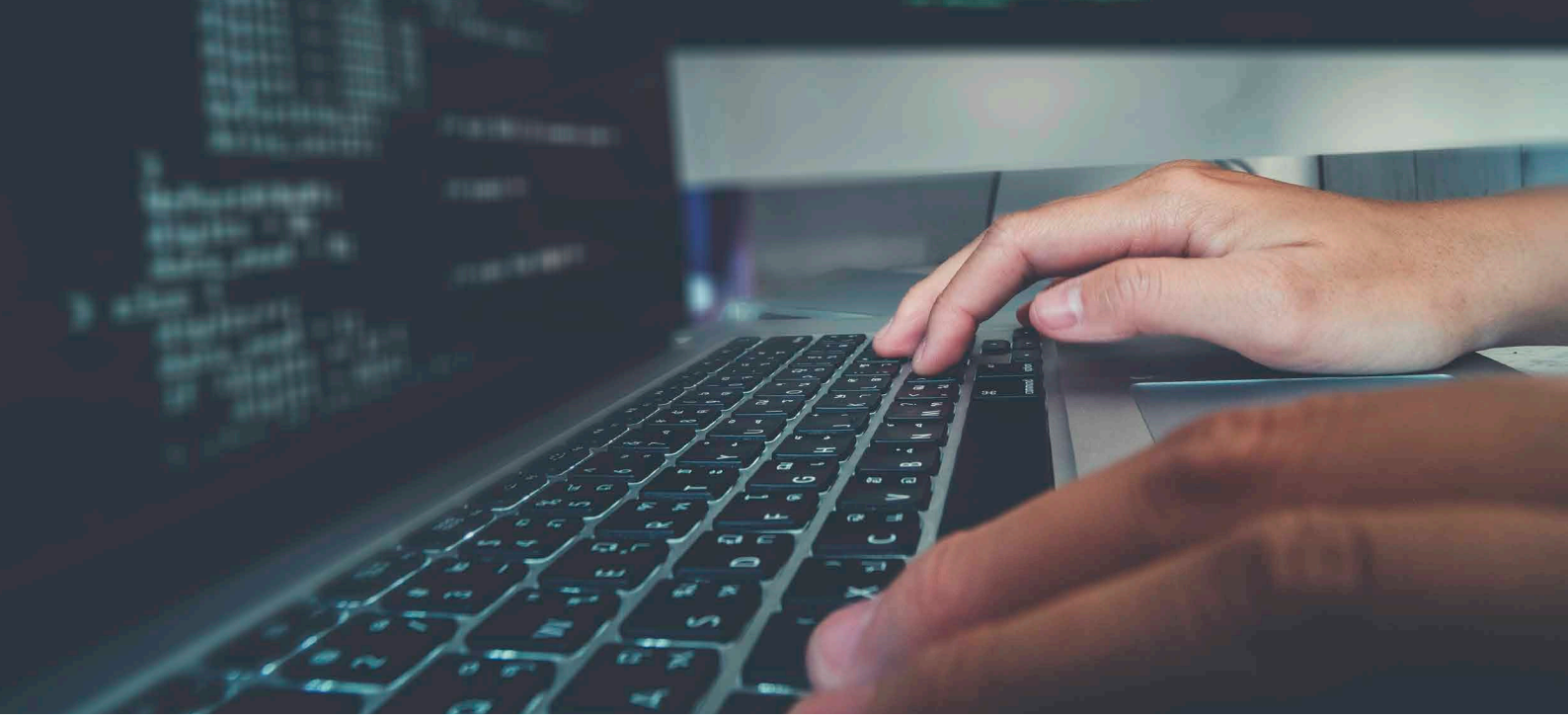
Cuando diseñamos una página web es conveniente especificar la versión de HTML con la que vamos a trabajar y lo haremos utilizando la etiqueta *<!DOCTYPE>* en la primera línea. Gracias a esta información, el navegador puede interpretar de forma correcta. Por ejemplo, HTML 4.01 cuenta con tres variantes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

Respecto a HTML 5, podemos decir que ya ha evolucionado a sus versiones HTML 5.1 y HTML 5.2.



XHTML

El lenguaje HTML fue adoptado por la comunidad en sus inicios, dando lugar a su crecimiento incorporando nuevas etiquetas sin ningún tipo de control. Esto provocó que determinados navegadores no pudieran interpretar el contenido. Esta falta de estandarización hizo que HTML fuera mas complejo que cuando empezó.

En ese momento, aparece XML y ofrece una alternativa al caos generado por el crecimiento de HTML. Sin embargo, XML está mas orientado al intercambio de información que a la presentación de esta. Una solución que se propuso fue la combinación de XML y HTML dando lugar a XHTML (eXtensible HyperText Markup Language), con dialegto de XML pero con características de HTML para la presentación.

El lenguaje XHTML surge a partir de la versión de HTML 4.0 a principios del año 2000 con el nombre de XHTML 1.0. A partir de entonces, aparecen nuevos borradores de versiones, como el XHTML 1.1 en el año 2001. Posteriormente, aparecieron las versiones 1.2 o 2.0 pero nunca llegaron a estandarizarse. En la actualidad se está trabajando sobre la versión XHTML 5.0.

La principal diferencia entre HTML y XHTML es que HTML proviene de SGML y XHTML proviene de XML. Es por este motivo, que los documentos XHTML deben cumplir las mismas normas que los documentos XML. Entre ellas podemos destacar las siguientes:

- Debe existir solamente un único elemento raíz llamado <html>.
- Se debe incluir la instrucción de procesamiento XML: <?xmlversion="1.0" encoding="UTF-8"?>.

- Debe incluir la declaración de DOCTYPE para la validación del documento frente al DTD: <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.2 Transitional//EN" http://w3.org.TE/xhtml1/DTD/xhtml1-transitional.dtd>
- Las etiquetas deben ir siempre en minúscula ya que XML diferencia entre ellas.
- Los valores de los atibutos siempre deben ir entre comillas dobles.

ponte a prueba

Queremos listar de forma numerada los nombres que hay a continuación, ¿qué nombre le pondremos a la etiqueta p2?

```
<p2>
    <li>Marta</li>
    <li>Sara</li>
    <li>Ana</li>
</p2>
```

a) ol
b) ul
c) il
d) li

En HTML, es la marca que sirve para agrupar varios elementos dentro de un bloque y permite asignarle un identificador.

a)
b) <dl></dl>
c) <form></form>
d) <img>

En el lenguaje HTML, ¿mediante qué comando se define un formulario?

a) <form> </form>
b) <table> </table>
c) <fom> </fom>
d)<img>

¿Qué nombre pondremos en la etiqueta <XXX> del siguiente HTML?

```
<!DOCTYPE html>
<html>
    <body>
        <h2>Introduce un
titulo</h2>
        <table>
            <XXX>

<YYY>Adios</YYY>
            </XXX>
        </table>
    </body>
</html>
```

a) <tr> </tr>
b) <td></td>
c) <form></form>
d) <img></img>

Señala que etiqueta podría faltar en lugar de <YYY>.

a) <tr> </tr
b) <td></td>
c) <form></form>
d) <img></img>

2.7. Herramientas de diseño web

Los lenguajes de marcas son los más utilizados para la creación de páginas web por la facilidad de poder escribir el código en cualquier herramienta de procesador de textos.

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

2.7.1. 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.

2.7.2. Editores avanzados

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

El código de una página web exige 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, se 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, cuentan con 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, que además son herramientas de código libre. Por otra parte, y aunque sea de licencia de pago, podemos mencionar Adobe Dreamweaver.

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

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

BUSCA EN LA WEB

Podemos descargarlos a través de las páginas:

www.sublimetext.com

www.notepad-plus-plus.org

2.7.3. 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 daremos 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.

Actualmente, en el mercado existen distintos navegadores, 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.



2.7.4. Otras opciones

Una alternativa son los editores tipo WYSIWYG (*what you see is what you get*), que puede traducirse 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.

ponte a prueba

Indica cuál de las siguientes opciones es una forma de añadir hojas de estilo a los documentos HTML.

a) CSS en línea

b) CSS interno

c) CSS externo

d) Todas las opciones son correctas

Indica otra forma de representar el siguiente contenido CSS.

```
h1{Font-family: Verdana;}
h1{color:red;}
```

a) h1{Font-family: Verdana; color: red;}

b) h1[{Font-family: Verdana} & {color: red;}]

c) h1[Font-family: Verdana} ; {color: red;}]

d) Ninguna opción es posible

Los atributos tienen distintas formas de seleccionar elementos. ¿Cómo lo representaremos si queremos elementos que tienen el mismo atributo con el valor que se pasa?

a) [atributo=valor]

b) [atributo]

c) [atributo |= valor]

d) Ninguna opción es coherente

En las hojas de estilo en cascada, indica cuál/es de las siguientes formas son válidas para incorporar color al contenido de la etiqueta <p>.

a) p {color: #ffo}

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

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

d) Todas las opciones son correctas

2.8. Hojas de estilo

El lenguaje HTML no está pensado para dar un formato o estilo a los documentos, sino que su objetivo es dar una estructura al documento. En posteriores versiones de HTML, se añadieron elementos que permitian cambiar la fuente o el color de algunos elementos. Esto provocó varios problemas, uno era que mezclaba la estructura con los estilos y otro que al desarrollar sitios web de grandes dimensiones, para cambiar su estilo se tenía que ir elemento por elemento.

Por esta misma razón, el consorcio W3C desarrolló lo que se conoce como CSS (Cascade Style Sheets). CSS son las hojas de estilo en cascada y permiten definir estilos de cada uno de los elementos de un documento HTML o XHTML, permitiendo realizar agrupaciones de elementos por tipo, y de esta forma modificar el estilo de todos los elementos de un tipo determinado con un único cambio.

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

Este lenguaje presenta numerosas ventajas al utilizar separación entre contenidos y presentación. Podemos destacar las siguientes:

- Necesitamos menos código a la hora de escribir.
- Es más fácil generar código y mantenerlo.
- Los documentos son más legibles.

Sintaxis

Existen tres formas diferentes de añadir hojas de estilo a los documentos HTML en función de donde colocamos el código CSS.

- **Forma 1: CSS en línea:** consiste en colocar el código CSS dentro de las etiquetas HTML usando el atributo *style*. De ese modo, los cambios de estilo CSS solo se aplican a dicho elemento y sus elementos hijos, si los hubiere. Por ejemplo:

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

En este caso, se ha utilizado dentro de la etiqueta de párrafo *<p>*, donde se ha asignado un estilo particular a la frase *hola mundo*.

- **Forma 2: CSS interno:** consiste en colocar el código CSS en la cabecera del documento HTML, es decir, dentro de la etiqueta *<head>* del documento.

  Para ello, pondremos la etiqueta *<style>*, que contendrá todo el código CSS y dará estilo a todo elemento HTML que aparezca en el *<body>*.

  Por ejemplo:

```
<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>
```

En este ejemplo, se ha otorgado estilo a los contenidos del párrafo *<p>* y de la cabecera de título *<h1>*. Al igual que en la forma de CSS en línea, con esta opción obtenemos un documento HTML con código mixto, es decir, con dos lenguajes distintos mezclados en un mismo documento, en este caso HTML y CSS.

En caso de que no nos guste la filosofía de mezclar diferentes lenguajes en un mismo documento, podemos optar por la tercera opción.

- **Forma 3: CSS externo:** con esta opción separamos el código HTML del código CSS de manera tajante, de modo que en el archivo *.html* solo habrá código HTML y todo lo relativo al estilo de la página que estamos creando estará contenido en CSS.

  Para ello, deberemos seguir los siguientes pasos:

  Primero, crearemos una hoja de estilo y la guardaremos en un archivo de solo texto con la extensión .css, por ejemplo estilo.css.

```
body {margin: 0px;}
td     {color: #000000;
          font-size: 12px;}
a      {color: #FF6600;
          font-weigth: bold;}
a:hover    {color:   #3366CC;}
```

  Y después, vamos a insertar el enlace *<link>* en cada documento, para así enlazar el documento HTML con el documento CSS.

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

Otra forma que tenemos de unir ambos 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:

1.º css en línea **+prioridad**

2.º css interno **....**

3.º 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:** los selectores son un elemento básico en CSS, porque nos ayudan a seleccionar un elemento o un conjunto de elementos HTML a los cuales se les dará un formato de estilo determinado. Podemos decir que los selectores *seleccionan* a qué elemento o tipo de elementos HTML se va a dar cierto estilo.

  La sintaxis de un selector es la siguiente:

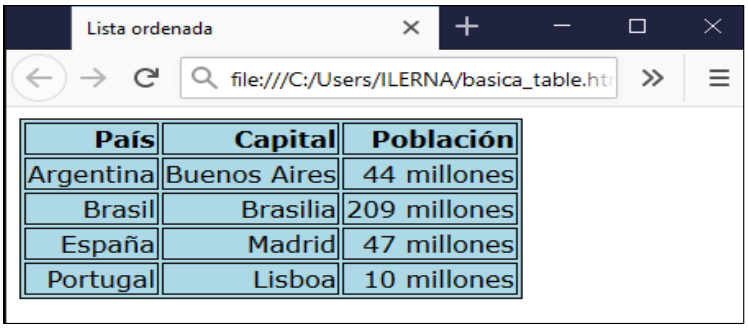
```
Selector    {propiedad: valor ; propiedad: valor; ...;}
    ↓              ↓              ↓
p           {Font-family: Verdana; Font-size: médium;}
```

Un ejemplo de código CSS mediante selectores podría ser el siguiente:

```
table {
    background-color: lightblue;
    text-align: right;
    font-family: verdana;
}
```

En este caso, estamos determinando que todos los elementos HTML de tipo tabla *<table>* tendrán un color de fondo azulado, el texto estará alineado a la derecha y el tipo de letra será de estilo *Verdana*.

Si aplicáramos ese código CSS al documento HTML de la imagen *2.4.j*, cuya visualización podíamos observar en *2.4.k*, obtendríamos la siguiente visualización del código:



País	Capital	Población
Argentina	Buenos Aires	44 millones
Brasil	Brasilia	209 millones
España	Madrid	47 millones
Portugal	Lisboa	10 millones

Pues bien, mediante reglas podemos unificar selectores más una declaración situada entre llaves. Se componen de una o varias parejas de propiedades con un valor y terminan 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 de la siguiente forma:

```
 /*…..*/
```

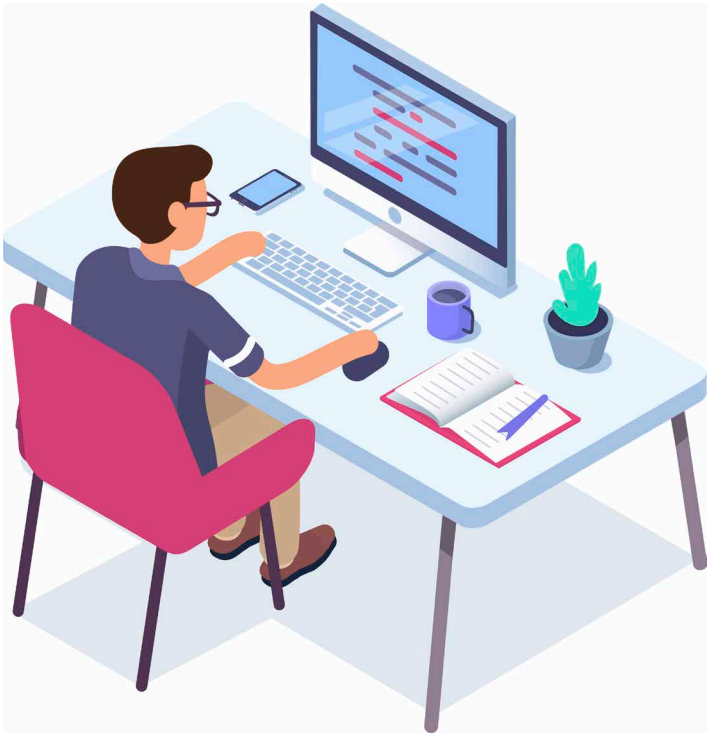
Por ejemplo, el siguiente código CSS es un comentario que no afectaría a un documento CSS:

```
/* esto es un comentario de código CSS*/
```

El cual es equivalente a:

```
/* esto es un comentario
de código CSS*/
```

Esto es debido a que no afectan los cambios de línea.



Selectores

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

Selector universal

Este selector permite seleccionar todos los elementos de una página. El siguiente fragmento de código elimina los márgenes de todos los elementos HTML.

```
* {margin: 0px}
```

Este selector se indica a través de un asterisco (*). Su uso no es muy habitual, ya que es complicado que un mismo estilo se aplique a todos los elementos de una página.

Selector de tipo o etiqueta

El selector de tipo o etiqueta permite seleccionar todos los elementos de una página cuya etiqueta sea igual que la del selector. El siguiente ejemplo modifica el estilo a todos los párrafos *(<p>)* de la página.

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

Es posible aplicar estilos a diferentes elementos de una página:

```
h1 { color: blue; }

h2 { color: green; }

p { color: black; }
```

En muchas ocasiones, queremos aplicar el mismo estilo a etiquetas diferentes. En esta situación podemos encadenar selectores, es decir, unificarlos. Veamos en el siguiente ejemplo etiquetas que comparten estilo:

```
h1 {
font-weight: normal;
font-family: Arial, Helvetica, sans-serif;
}

h2 {
font-weight: normal;
font-family: Arial, Helvetica, sans-serif;
}

p {
font-weight: normal;
font-family: Arial, Helvetica, sans-serif;
}
```

Para agrupar todas las reglas anteriores crearemos un selector múltiple, en el que los selectores se separan por comas (,) y el resultado es equivalente a las tres reglas anteriores.

```
h1, h2, p {
font-weight: normal;
font-family: Arial, Helvetica, sans-serif;
}
```

En hojas de estilo complejas es muy habitual agrupar selectores con propiedades comunes y, posteriormente, definir propiedades más específicas de esos mismos elementos. Utilizando el ejemplo anterior, aplicaremos diferentes tamaños de letra para cada uno de ellos.

```
h1 { font-size: 2em; }

h2 { font-size: 1.5em; }

p { font-size: 1.2; }
```

Selector descendiente

El selector descendiente selecciona los elementos que se encuentran en el interior de otros elementos. Es decir, cuando un elemento se encuentra entre las etiquetas de apertura y cierre de otro elemento.

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, 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><spanclass="nieto"><a>Texto3</a></span></
p>
```

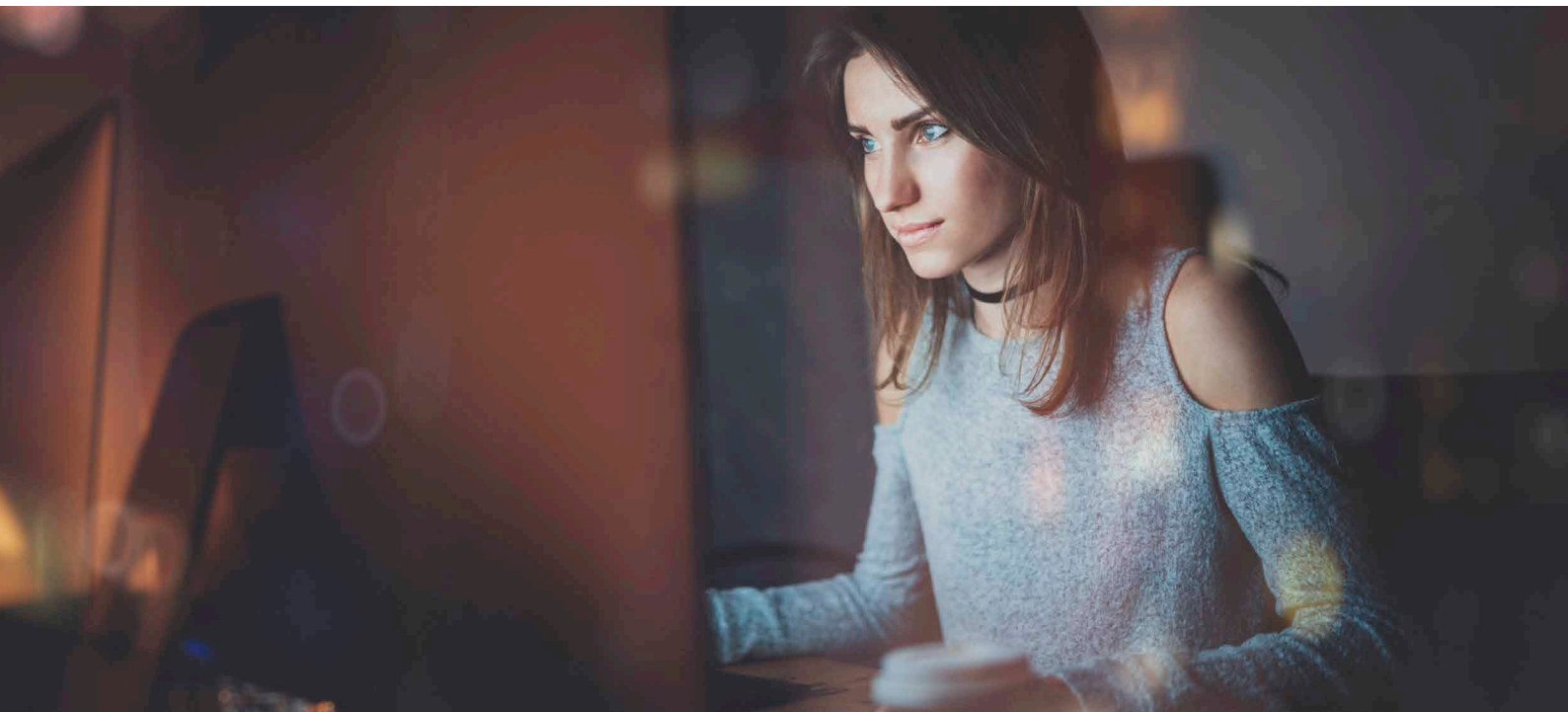
El selector *p* seleccionará tanto *Texto2* como *Texto3*. En los selectores descendientes no es necesario que un elemento se encuentre descendiente directamente del otro. La única condición es que debe estar dentro del otro elemento, independientemente del nivel de profundidad en el que se encuentre.

Y, a continuación, podemos visualizar:

Texto 1

Texto 2

Texto 3



Selector de clase

Este tipo de selector 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>*. Para diferenciarlos debemos hacer uso del atributo.

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

Podemos construir de la siguiente forma:

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

Texto 1

Texto 2

Texto 3

Como se puede observar, el selector *p.grande* se puede interpretar como *todos los elementos <p> de la página, que tengan el atributo class con el valor grande.*

- **Selector de ID:** nos ofrece la posibilidad de seleccionar una sola línea de una página completa.

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

Texto 1

Texto 2

Texto 3

En este ejemplo, el selector *peque* se puede interpretar como *todos los elementos <p> de la página, que tengan el atributo id con el valor peque.*

PARA + INFO

Este tipo de selectores solo permiten seleccionar un elemento de la página, ya que el valor del atributo id no se puede repetir en más de un elemento diferente en una misma página.

3 DEFINICIÓN DE ESQUEMAS Y VOCABULARIOS DE XML

Los esquemas XML son las descripciones y formalismos que imponen una estructura y sintaxis en un determinado lenguaje XML, y, por tanto, regulan la creación de cualquier documento XML.

Como ya sabemos, existen múltiples lenguajes XML, así que cada lenguaje tendrá su propio esquema XML que determinará sus propias normas.

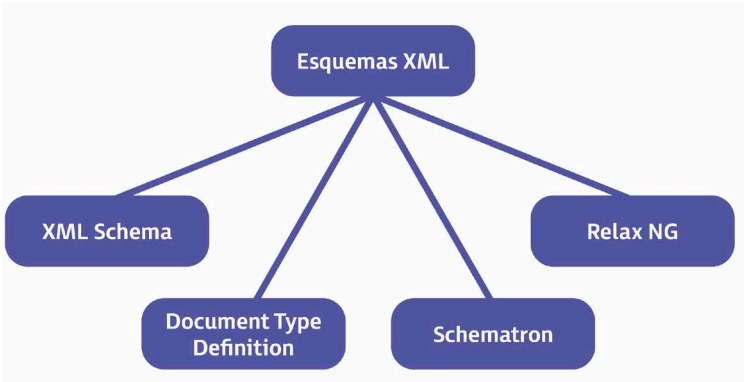
Además, debemos tener en cuenta que existen distintas técnicas o estándares para formalizar un esquema XML. Es decir, podemos formalizarlo con DTD (Document Type Definition) o con el estándar XML Schema (propuesto por la W3C) u otras técnicas.

En este tema veremos algunos de los diferentes esquemas XML, así como sus estructuras, reglas, sintaxis, la forma de validarlos o herramientas que podemos utilizar para crearlos.

3.1. Descripción de la información de documentos XML. Tecnologías

Un esquema XML es una descripción o formalización de la estructura y reglas sintácticas de un documento XML. En el esquema, por tanto, se describirán todas las normas y restricciones que debe seguir un documento XML.

Existes distintos tipos de lenguajes de esquemas XML, algunos de ellos son los siguientes:



El lenguaje de esquema nativo de XML es el DTD (Document Type Definition), dado que no solo regula documentos XML. Ya lo hacía para documentos anteriores a XML, como SGML (recordemos que XML es un lenguaje de marcas que derivó de SGML).

No obstante, hoy en día existen lenguajes de esquemas XML más potentes y versátiles que DTD, como puede ser XML Schema, propuesto en 2001 por la W3C, o RELAX NG,

otro lenguaje de esquema XML que tiene la propiedad de ser bastante simple, a la vez que cumple con las reglas sintácticas propias de los documentos XML (los DTDs, por ejemplo, aunque regulen la estructura y reglas de un documento XML no cumplen con las mismas reglas sintácticas que un documento XML, como por ejemplo, la de tener siempre una etiqueta de cierre).

También tenemos otros lenguajes de esquemas XML como es Schematron. Este destaca por permitir expresar reglas de forma más compleja que otros lenguajes de esquema XML más limitados.

En cualquier caso, hemos de tener siempre presente que sea cual sea el lenguaje de esquema que usemos, un esquema XML tiene la función básica de especificar las reglas, estructura y sintaxis de un documento XML.

3.2. Asociación con documentos XML

Para poder validar un documento XML, este debe estar asociado a un esquema concreto. La forma de asociar un documento XML con un esquema determinado variará en función del lenguaje de esquema que vayamos a tratar.

3.2.1. Asociación de un documento XML con DTD

Para indicar el DTD de un documento XML utilizaremos el prólogo del documento XML. Un ejemplo de prólogo de documento XML serían las dos primeras líneas del siguiente documento XML:

```
<?xmlversion="1.0" encoding="ISO-8859-1"
standalone="no">
<!DOCTYPE persona SYSTEM "personas.dtd">
<persona>
<dni>48311765</dni>
<nombre> Pedro </nombre>
<apellido> Martínez </apellido>
</persona>
```

Como ya sabemos, la primera línea nos indica qué versión de XML se va a utilizar, qué tipo de codificación se va a usar y si el documento requiere de un archivo externo o no para definir su estructura.

La segunda línea, que es la que nos interesa, es la que asocia el documento XML con su DTD. En este caso se declara que el documento XML es de tipo privado (SYSTEM) y que su DTD está en un archivo externo (*personas.dtd*).

En este ejemplo, el DTD es externo, pero podría haber sido interno. En la etiqueta *<!DOCTYPE>* se deben especificar dos datos esenciales: la ubicación de las reglas del DTD y el carácter del documento, es decir, si es público o privado.

- **La ubicación** puede ser**:**
  - **Interna**: cuando las reglas se encuentran en el mismo documento XML.
  - **Externa:** cuando las reglas se encuentran en un archivo independiente.
  - **Mixta**: combina los dos anteriores. Las reglas se encuentran en ambas partes.
- **El carácter** puede ser:
  - **De uso privado:** identificado por la palabra SYSTEM.
  - **De uso público**: identificado por la palabra PUBLIC. Debe ir con la palabra FPI (*formal public identifier* o identificador público formal), que identifica el 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

Formada 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 de documento descrito.
- Cuarto campo: indica el idioma del documento.

3.2.2. Asociación de un documento XML con XML Schema

Para asociar un documento XML con un esquema de tipo XML Schema, usaremos el espacio de nombres dentro de las etiquetas de los propios elementos del documento XML.

Por ejemplo:

```
<?xmlversion="1.0" encoding="UTF-8" standa-
lone="no">
<persona xmlns:xsi ="http://www.w3.org/2001/
XMLSchema-instance"
xsi:noNamespaceSchemaLocation = "personas.
xsd">
<dni> 48311765 </dni>
<nombre> Pedro </nombre>
<apellido> Martínez </nombre>
</persona>
```

Como podemos observar, situamos la referencia al esquema dentro de la etiqueta raíz <*persona*>, usando el espacio de nombres.

ponte a prueba

Primero de todo, es necesario saber que el objetivo de un DTD es especificar aquellos elementos que deben aparecer, junto con el orden que deben seguir. ¿A qué se refiere la siguiente definición: es una declaración de un tipo de elemento que nos señala si existe un elemento a un determinado documento XML?

a) <!ELEMENT>

b) <!ATTLIST>

c) <!NOTIFICATION>

d) Ninguna opción es correcta

¿Cuál de las siguientes opciones es un atributo que podemos diferenciar en un DTD?

a) CDATA

b) ID

c) NMTOKEN

d) IDREF

e) Todas las opciones son correctas

¿Cómo se representa en el documento XML teniendo la siguiente declaración del atributo en el DTD?

```
<!ATTLIST hora zona CDATA
"GMT+1" #REQUIRED>
```

a) <hora zona="GTM+1">13:00</hora>

b) <zona hora="GTM+1">13:00</zona>

c) <hora zona="GTM+1">13:00</zona>

d) Ninguna opción es correcta

Indica cuál de las siguientes opciones es un componente que utilizamos para declarar un elemento y comprobar si existe en el documento XML.

a) xs:element

b) xs:attribute

c) xs:schema

d) Todas las opciones son correctas

¿Cuál de las siguientes afirmaciones sobre los tipos de datos complejos es correcta?

a) Se asignan a aquellos elementos que poseen elementos o atributos descendientes

b) Puede estar vacío, es decir, que no tiene contenido, aunque sí atributos

c) Puede tener contenido mixto, que significa, mezcla de contenido textual con elementos descendientes

d) Todas las opciones son correctas

3.3. Creación de descripciones. Estructura, reglas y sintaxis

A continuación, vamos a ver la estructura de dos tipos de esquemas XML: los DTD y los XML Schemas. Aunque a veces los términos nos puedan resultar algo enredados, no debemos confundir el concepto de *Esquemas XML* con uno de los tipos de esquemas en concreto: *Schema XML*". Como hemos comentado anteriormente, es un tipo de esquema más, como Document Type Definition (DTD) o Schematron.

3.3.1. Estructura y componentes de un 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 en un determinado documento XML.

Sintaxis

```
<!ELEMENT nombre_elementomodelo_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). Los podemos describir como #PCDATA y deben aparecer entre paréntesis. Por ejemplo:

```
<ELEMENT titulo (#PCDATA)>
```

Elementos descendientes

Es conveniente que aparezcan entre paréntesis y se basan 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 0 o 1 vez.
*	El elemento (o secuencia de elementos) puede aparecer de 0 a N veces.
+	El elemento (o secuencia de elementos) puede aparecer de 1 a N veces.

- **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.

<!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_atributotipo_atributocaracter
    nombre_atributotipo_atributocaracter
    …
>
```

Donde ***nombre_atributo*** se debe corresponder con un nombre XML válido y *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 ID 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 blanco 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.

<!ENTITY>

Elemento bastante avanzado en cuanto al diseño de DTD. Su declaración hace referencia a un tipo determinado de entidad. Podemos distinguir las siguientes referencias:

Referencia a entidades generales:

- **Internas:**

Sintaxis

```
<!ENTITY nombre_entidaddefinición_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_entidadtipo_usourl_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:**

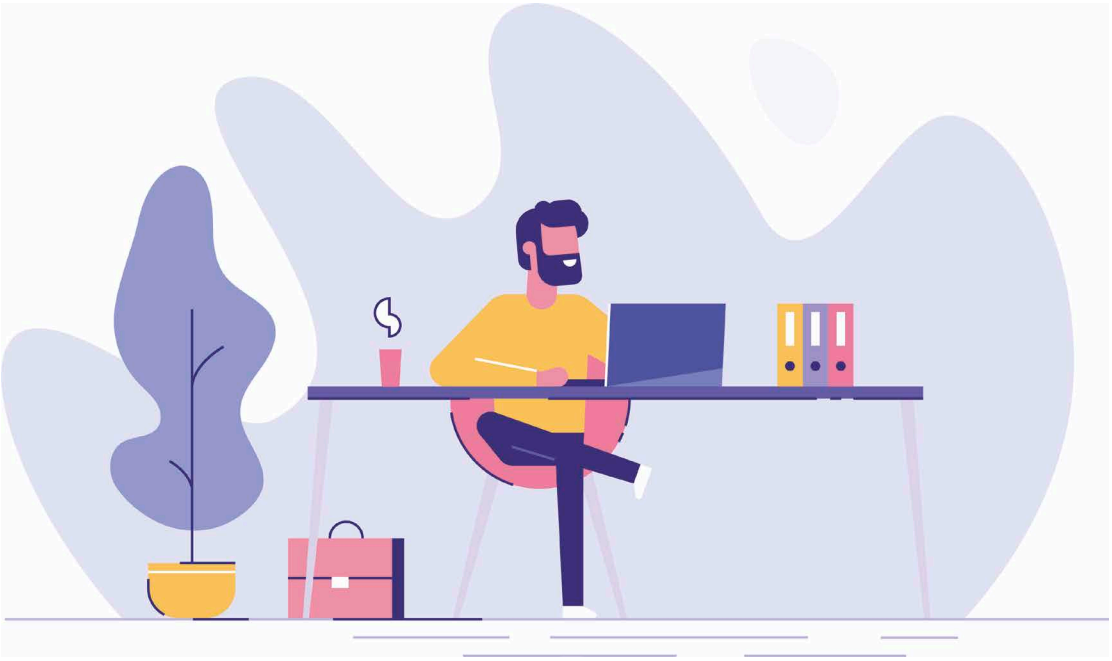
```
Sintaxis
<!ENTITY % nombre_entidaddefinición_entidad>
```

- **Externas:**

```
Sintaxis
<!ENTITY % nombre_entidadtipo_usofpiurl_archivo>
```

Entidades que no están procesadas (unparsed)

```
Sintaxis
<!ENTITY % nombre_entidadtipo_usofpivalor_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">
```

3.3.2. Estructura y componentes de XML Schema

Un XML Schema debe cumplir una serie de reglas:

- El nombre correspondiente al elemento raíz se llamará *<schema>*.
- Debe disponer de un espacio de nombres.

BUSCA EN LA WEB

http://www.w3.org/2001/XMLSchema

El uso de prefijos es opcional, aunque recomendable. Los prefijos más comunes son *xs* o *xsd*. Por ejemplo:

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

Como un documento bien formado dispone de un elemento raíz, como mínimo, para los esquemas XML necesitaremos, 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:schemaxmlns: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:schemaxmlns: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 utilizaremos una serie de componentes. A continuación, nombramos algunos de ellos, imprescindibles en este proceso:

- **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.
    - **version**: 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:

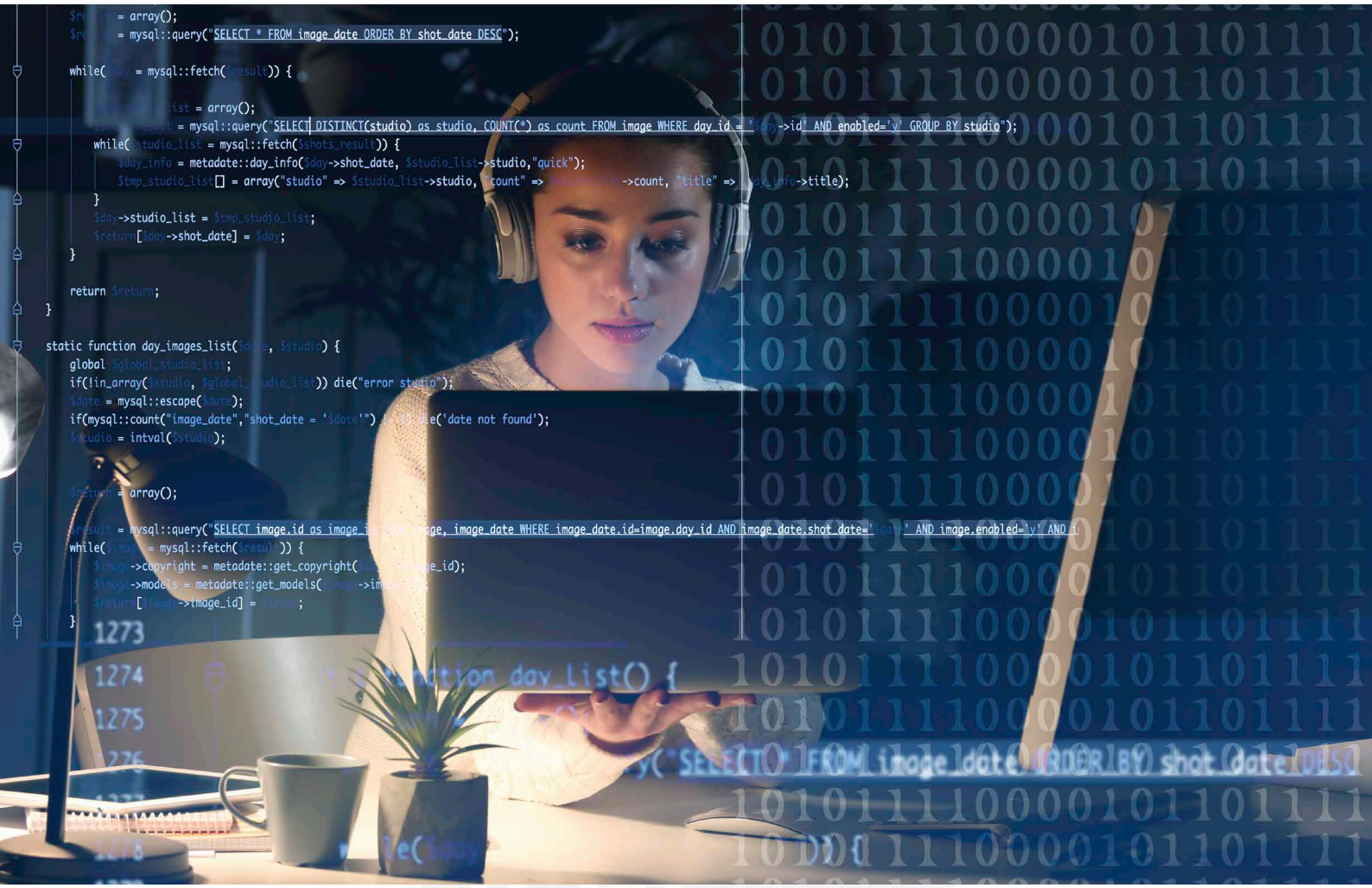
- **Tipos de datos predefinidos:** están organizados de forma jerárquica y podemos diferenciar sobre unos 45 tipos de datos predefinidos diferentes. Cada tipo debe ser de la misma forma que sea el tipo de su padre.
  - **Definición del tipo predefinido especial: xs: anyType:** lo encontramos en la jerarquía 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 en distintas categorías:

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 pueden dividirse 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 predefinida.
  - 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 valor 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 0.	Texto
xs:minLength	Especifica el mínimo número de caracteres (o elementos de una lista) permitidos. Tiene que ser mayor o igual que 0.	Texto
xs:maxLength	Especifica el máximo de caracteres (o elementos de una lista) permitidos. Tiene que ser mayor o igual que 0.	Texto
xs:fractionDigits	Especifica el número máximo de posiciones decimales permitidas en números reales. Tiene que ser mayor o igual que 0.	Números con parte decimal
xs:totalDigits	Especifica el número exacto de dígitos permitidos en números. Tiene que ser mayor que 0.	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.

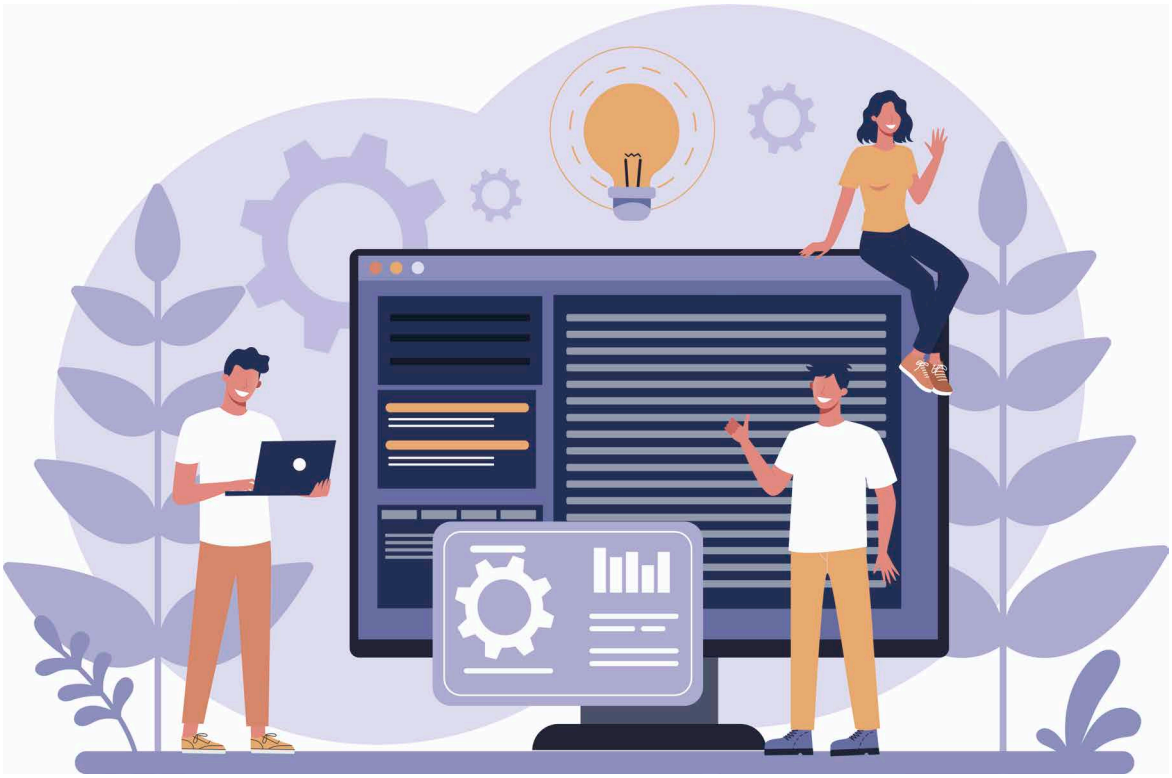
  – Atributos optativos principales:

  - **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**: señala si el tipo es derivable según la restricción, extensión o por ambas.

3.4. Validación

En secciones anteriores, hemos visto algunas de las reglas que debe cumplir un documento XML para estar bien formado. Ahora bien, no es lo mismo que un documento XML esté bien formado a que esté validado.

Un documento XML estará bien formado si cumple con las especificaciones de la W3C y, por tanto, es apto desde el punto de vista sintáctico. Sin embargo, diremos que un documento XML está validado si el documento es correcto conforme a su esquema, el cual es particular, no general para todos los XML existentes.



3.4.1. Validación de documentos XML con esquemas tipo DTD

Aunque hoy apenas se utiliza, esta técnica de validación consiste, mediante un DTD, en especificar aquellos elementos que deben aparecer, junto con el orden que deben seguir, los que son obligatorios y los que no, etc.

3.4.2. Validación de documentos XML con esquemas tipo XML Schema

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.
- Permite 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.

¡RECUERDA!

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

3.5. Herramientas de creación y validación

- **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 las 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.

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 y que permiten comprobar si un documento está bien formado son las mismas que pueden determinar si permiten validar un documento que cuente con un esquema XML. Existen una serie de herramientas para llevar a cabo esta función. A continuación, citamos una de ellas:

BUSCA EN LA WEB

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:

BUSCA EN LA WEB

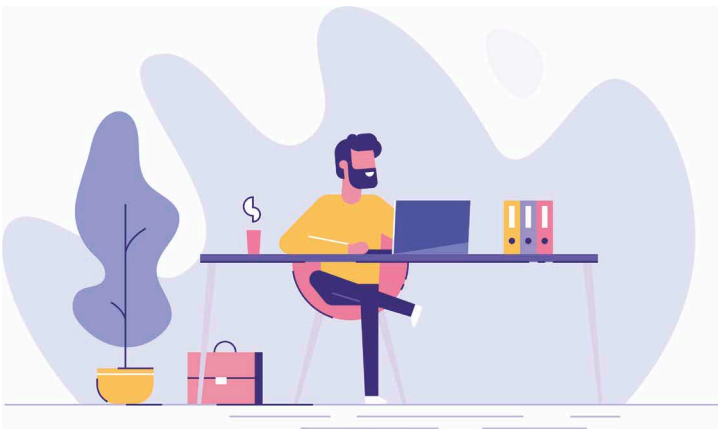
Herramientas online:

www.freeformatter.com/xsd-generator.html

PARA + INFO

Editores XML que cuentan con esta funcionalidad:

XMLSPY de Altova



4 APLICACIÓN DE LOS LENGUAJES DE MARCAS A LA SINDICACIÓN DE CONTENIDOS

RSS (*Really Simple Syndication* o *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 utilice un software específico que esté diseñado para leer contenidos RSS (*FeedReader*).

La información que se encuentra disponible en un sitio web se puede compartir de varias formas diferentes. Una bastante sencilla es suscribiendo la fuente a un agregador de noticias. Otra, compartiendo aquella información insertada en otros sitios web. Así, la persona encargada de recibir las noticias actúa también como emisor. Es 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 a difundir.

Es necesario, que señalemos la existencia de otro programa que realiza las mismas funciones. Es Atom, también deriva de XML, aunque añade una serie de ventajas, entre las que podemos destacar la resolución de algunas ambigüedades que ofrecía RSS y la extensión de su alcance. En cualquier caso, ambas tecnologías proveen de información y contenidos actualizados.

Estos dos lenguajes se utilizan para guardar diferentes contenidos que se quieran 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 principales ventajas destacamos:

- 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. Es bastante rápido a la hora de transmitir los datos, por lo que es muy recomendable para dispositivos móviles.
- Ofrece la posibilidad de filtrar información de diferentes sitios sin perder tiempo utilizando la agrupación.

ponte a prueba

Es un lenguaje derivado del XML y tiene como función principal compartir o distribuir la información que se encuentra en la web.

a) RSS

b) XQuery

c) XSchema

d) Ninguna opción es correcta.

4.1. Ámbitos de aplicación

La sindicación o redifusión web tienes varios ámbitos de aplicación. Desde diarios digitales hasta redes sociales, pasando por aplicaciones de trabajo o programas de descargas.

No debemos relacionar exclusivamente la sindicación o redifusión web solo con las webs de blogs. Con la RSS podemos compartir el contenido de una página web en tiempo real con otros usuarios. 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.

Por tanto, el ámbito de aplicación de la sindicación o redifusión web es amplio. Muchos tipos de datos pueden ser objeto de redifusión web.

Para permitir la visualización de los contenidos actualizados vía RSS debemos suscribirnos en el enlace web correspondiente al que deseamos sindicarnos. Desde el punto de vista de los suscriptores, la redifusión de contenidos permite, entre otras cosas, la actualización profesional.

4.2. Estructura de los canales de contenidos

Los documentos RSS son documentos XML, por tanto, han de cumplir las normas sintácticas establecidas por la W3C para todos los archivos XML. Además, deben tener una declaración donde se informe sobre la versión que se está utilizando.

Podemos desgranar los archivos RSS en 3 partes:

Etiqueta	Función
<?xml… ?>	Información general sobre la versión XML a utilizar y el tipo de codificación de caracteres.
<rss … >	Información sobre la versión RSS que se va a utilizar .
<channel>	Contenido del documento.

Primero pondremos la etiqueta de declaración del documento:

```
<?xml versión = "…" encoding = "… " ?>
```

Después pondremos la etiqueta de apertura *<rss>*, en la que indicaremos el tipo de versión RSS que vamos a utilizar. Este es el elemento raíz del documento.

```
<rss> … </rss>
```

Y, a continuación, tenemos el elemento *<channel>* que describe el canal RSS en cuestión.

```
<channel> … </channel>
```

Debemos tener en cuenta que el elemento *<channel>* puede contener uno o más elementos *<item>*:

```
<?xml versión="1.0" encoding="utf-8"?>
<rss versión="2.0">
<channel>
    <!-- Aquí se insertan las propiedades
del canal -->
<title> … </title>
<link>...</link>
<description>…</description>
<item>
<!—Contenido entrada publicada -->
</item>
<item>
…
</item>
</channel>
</rss>
```

Un ejemplo podría ser el siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
  <channel>
    <title>Página deportiva</title>
    <link>https://www.tododeporte.org</link>
    <description>Información sobre deporte mundial</description>
    <item>
      <title>Resultados de la semana</title>
      <link>https://www.tododeporte.org/xml/resultados</link>
      <description>Resultados deportivos de la semana 16</description>
    </item>
    <item>
      <title>Últimas novedades</title>
      <link>https://www.tododeporte.org/xml/novedades</link>
      <description>Noticias deportivas de última hora</description>
    </item>
  </channel>
</rss>

```

Concepto y estructura RSS

youtu.be/MlnaOIYVTq4

4.3. Elementos principales de un RSS

4.3.1. <channel>

Su función principal es describir la fuente RSS mediante tres elementos obligatorios. Son:

- **<title>**: 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>.** Un ejemplo básico de RSS con todos los elementos obligatorios podría ser el siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">
  <channel>
    <title> Noticias Online de Última Hora </title>
    <link> http://www.diarioultimahora.es </link>
    <description> Diario en español con noticias actualizadas </description>
    <item>
      <title> Rusia anuncia la vacuna del covid19</title>
      <link> http://www.diarioultimahora.es/internacional/rusia_vacuna_sputnikv</link>
      <description> El presidente ruso defiende la vacuna sputnik V ante las críticas
      internacionales por las dudas sobre sus efectos secundarios</description>
    </item>
    <item>
      <title> Reunión por la paz en Colombia </title>
      <link> http://www.diarioultimahora.es/internacional/colombia_proceso_paz </link>
      <description> Nuevo intento de las autoridades colombianas y los grupos rebeldes
      por conseguir una paz definitiva en el país</description>
    </item>
  </channel>
</rss>
```

Como podemos observar, el elemento *channel* posee los tres elementos obligatorios, *<title>*, *<link>* y *<description>*.

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

Elemento	Descripción
<category>	Define una o más categorías 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 tolive"*)
<WebMaster>	Define la dirección email del webmaster del canal

4.3.2. <ITEM>

Cada elemento *<item>* puede definir un determinado artículo en el canal RSS. En este caso, también debe contar con tres elementos fundamentales:

- **<title>**: hace referencia al título del artículo.
- **<link>:** define el hipervínculo al artículo.
- **<description>:** se encarga de describir el canal.

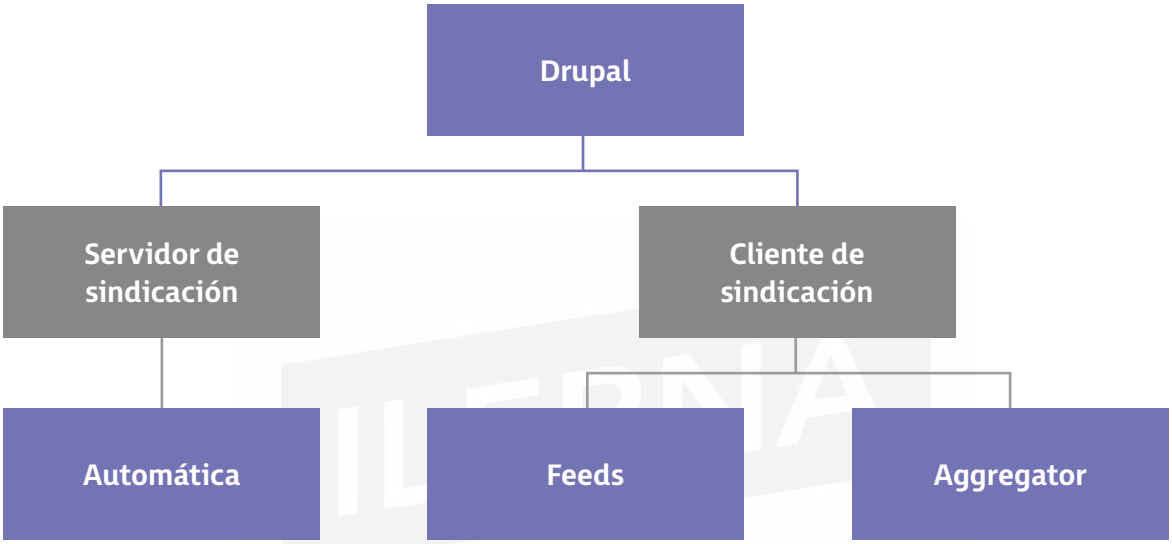
Los elementos opcionales más importantes de *<item>* son los siguientes:

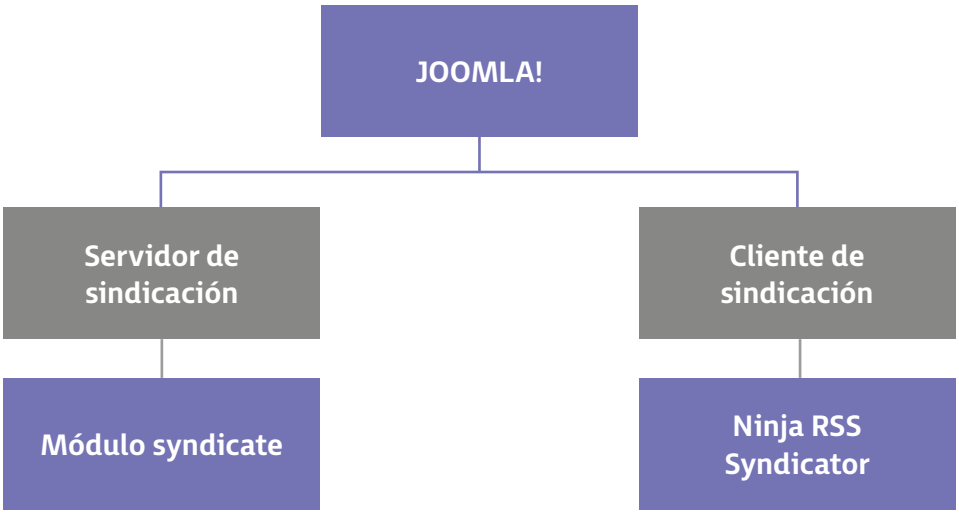
Elemento	Descripción
<author>	Especifica el e-mail del autor del artículo
<category>	Define la categoría/s a las que pertenece el artículo
<comments>	Permite enlazar a los comentarios sobre ese tema
<enclosure>	Permite incluir un archivo multimedia
<guid>	Define un identificador único para el artículo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

4.4. Tecnologías de creación de canales de contenidos

Para crear 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 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*, utilizan también servidores web Apache y gestores de bases de datos MySQL.





ponte a prueba

Indica cuál de las siguientes opciones son elementos se necesita en la etiqueta <channel> para describir la fuente RSS.

a) <title>, <link> y <description>

b) <rating>, <language> y <link>

c) <title>, <description> y <rating>

d) <description>, <language> y <link>

Dentro del elemento <channel>, ¿qué etiqueta permite información inmediata de los cambios de canal?

a) <cloud>

b) <copyright>

c) <generator>

d) <pubDate>

¿Cómo se llama el elemento que cuenta con la posibilidad de poder definir un determinado artículo en el canal RSS?

a) <item>

b) <title>

c) <link>

d) <channel>

4.5. Validación y publicación

Validación del archivo RSS

Después de haber creado un archivo RSS, debemos comprobar que el archivo cumple con todas las normas y requisitos, tanto XML como RSS. Para realizar dicho proceso usamos la validación.

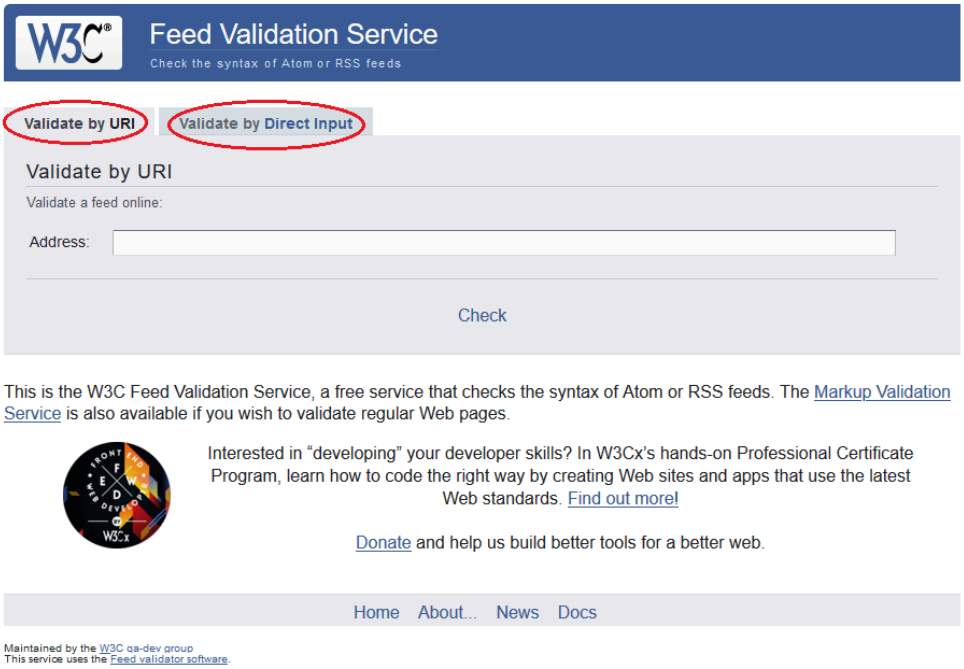
Existen múltiples webs desde las cuales podemos realizar el proceso de validación de un archivo RSS. **W3C** posee su propio validador de RSS y lo ofrece gratuitamente a la comunidad.

BUSCA EN LA WEB

En la siguiente dirección podemos usar el validador oficial de la W3C para validar archivos desarrollados en RSS o Atom:

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

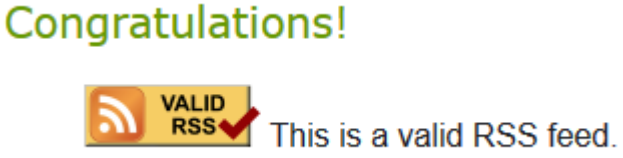
Al entrar en dicho enlace, nos encontraremos lo siguiente:



Como podemos observar en la imagen del validador, aparecen marcadas en rojo las dos opciones existentes para validar un archivo RSS:

- **Validate by URI**: en este caso indicaríamos la ubicación del archivo RSS por medio de una dirección URI. El archivo RSS debe estar accesible en la red.
- **Validate by Direct Input**: validación por código directo. En este caso colocaríamos el código RSS directamente en el cuadro de texto disponible.

Sea cual sea la opción elegida, una vez indicado el archivo RSS al validador, debemos hacer clic en *check* y el sistema nos devolverá un resultado. En caso de que el validador no encuentre errores en el código RSS también nos lo indicará:



Publicación del archivo RSS

Una vez que ya hemos escrito el código y realizado su validación correspondiente, debemos subirlo a un servidor web. Seguiremos estos pasos:

- Primero insertamos el enlace en la página de inicio para 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.

4.6. Directorios de canales de contenidos

Existen una serie de canales de contenidos de donde podemos extraer 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 el portal web.

Los directorios de canales de contenidos nos ofrecen las siguientes ventajas:

- Permiten el acceso a los ficheros RSS.
- Ayudan a la localización de los archivos RSS a cualquier usuario.
- Clasifican y catalogan por temáticas (u otros criterios) los ficheros RSS.

Existen múltiples directorios de canales de contenidos en la web. Aquí ponemos algunos ejemplos:

BUSCA EN LA WEB

El geoportal de Infraestructuras de Datos Espaciales de España (IDEE) es un listado de canales RSS para suscribirnos a los que nos interesen:

https://www.idee.es/web/idee/inicio

BUSCA EN LA WEB

Directorio de canales de contenidos ofrecido por el Ministerio de Industria, Comercio y Turismo del Gobierno de España:

https://www.mincotur.gob.es/es-es/Paginas/rss.aspx

4.7. 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:

- **Agregadores de escritorio:** aquellos que se pueden instalar en el ordenador de un usuario.
- **Agregadores en línea:** no es necesario que el usuario los instale. Solo hay que darse de alta en el sitio para utilizarlo.
- **Agregadores como *plugins*:** existen algunos navegadores como Firefox, Nestcape, Opera y algunos más, que los incluyen entre sus programas para ofrecer un servicio al usuario.

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

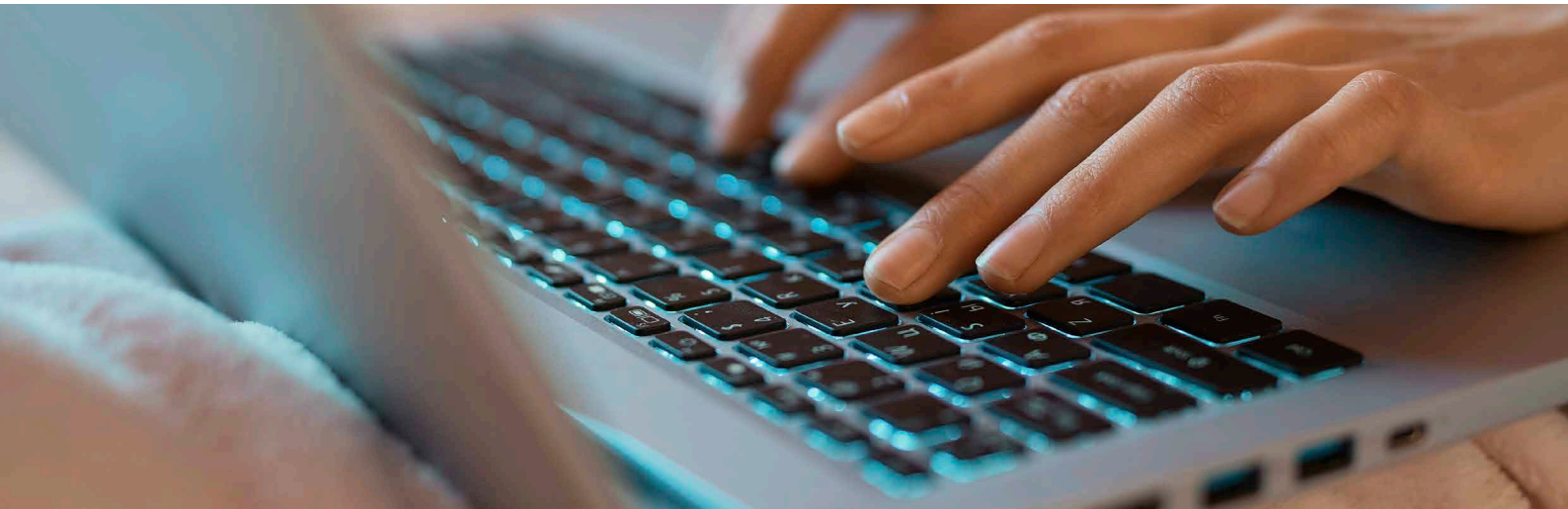
Cómo utilizar RSS para ofrecer información

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

El código necesario para 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 con 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*.



ponte a prueba

Dentro del elemento <item>, ¿qué etiqueta permite incluir un archivo multimedia?

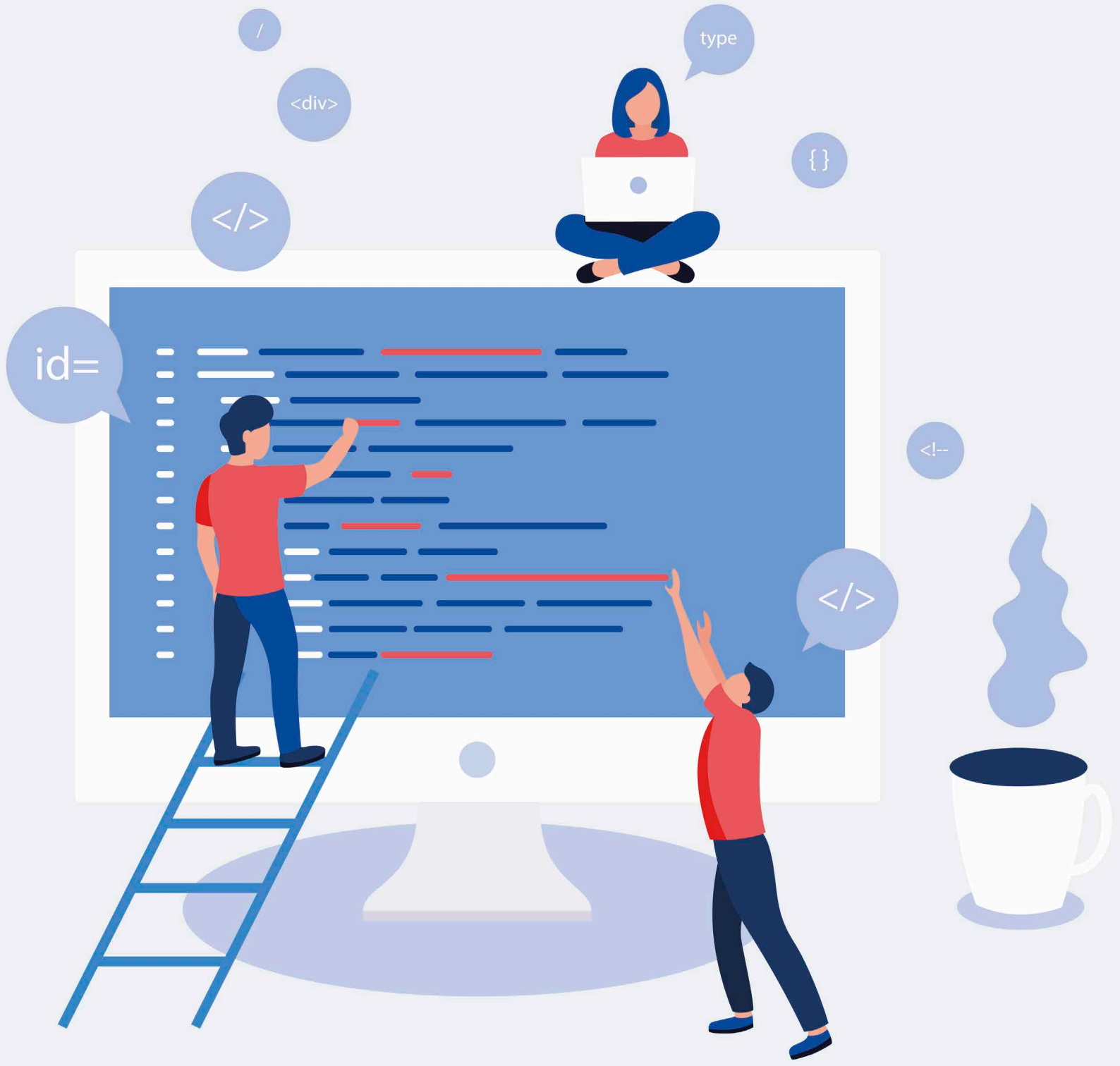
a) <enclosure>
b) <guid>
c) <source>
d) <author>

Una vez creado el archivo RSS, ¿mediante qué herramienta podemos validarlo?

a) W3C
b) PDF validator
c) WC3
d) 3WC

¿Cuál es el lector RSS que se puede instalar en el ordenador de un usuario?

a) Agregador de escritorio
b) Agregador en línea
c) Agregador como plug-in
d) Todas las opciones son correctas



5 CONVERSIÓN Y ADAPTACIÓN DE DOCUMENTOS XML

XSL es el acrónimo de *Extensible Stylesheet Language*, que traduciríamos como "lenguajes de hojas de estilo extensibles". Hace referencia a una familia de lenguajes XML que son usados para transformar y presentar documentos XML.

El W3C creó tres especificaciones para el XSL:

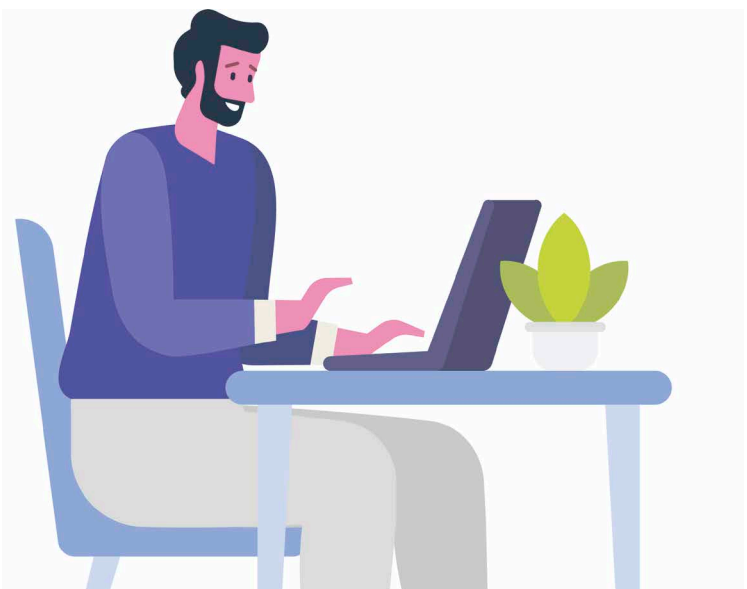
- **XSLT**: es un lenguaje XML diseñado para transformar documentos XML a otras sintaxis.
- **XPath**: es un lenguaje de consulta que usamos para crear expresiones que recorren un documento XML a través de sus nodos para acceder a determinadas secciones.
- **XSL-FO**: es un lenguaje XML que usamos para dar una presentación visual a un documento XML. Comúnmente usado para la generación de PDF.

5.1. Necesidades y ámbitos de aplicación

En 1997, grandes compañías informáticas, una de ellas Microsoft, pidieron crear un lenguaje de presentación y estandarización para documentos XML. A raíz de esa petición se formó el W3C Working Group, que desarrolló los lenguajes XSL.

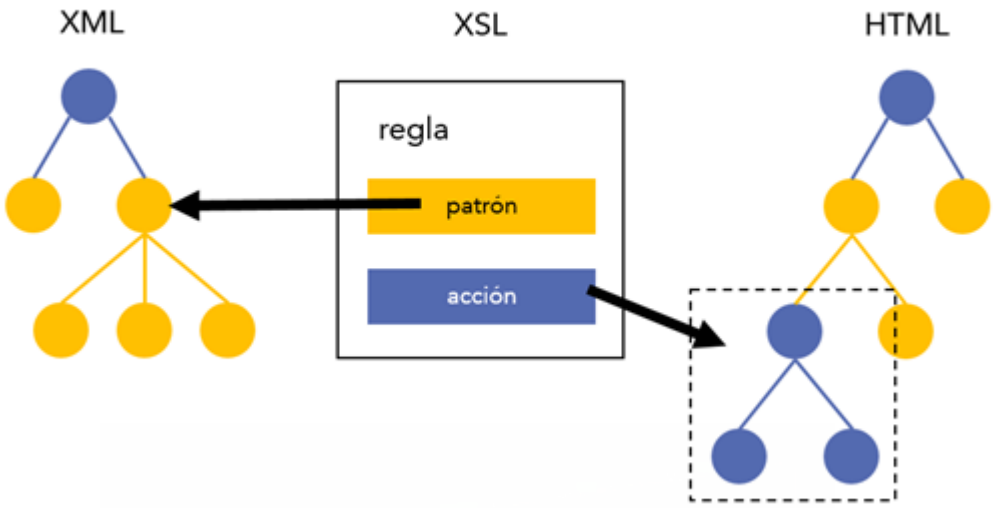
Debemos tener en cuenta que diferentes documentos XML pueden tener DTD distintos, por lo que la estandarización en la presentación y la transformación de documentos XML a otras sintaxis se convierte en una necesidad.

Como veremos a continuación, podemos aplicar la transformación de documentos XML a muchos ámbitos y formatos. Por ejemplo, por distintos motivos puede ser necesario transformar un documento XML en otro tipo HTML o pasar de un documento XML a formato de texto u otras opciones que fueran necesarias. Además, nos permite presentar la información XML de manera más legible para una impresión o una lectura por pantalla.



XSLT (*extensible Stylesheet Language for transformations*)

Es un lenguaje estándar de W3C que se recomienda por primera vez en 1999. Está 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.

Hojas de estilo: 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:
  - No puede cambiar el orden de los elementos pertenecientes a un documento HTML.
  - No tiene permitido llevar a cabo distintas operaciones con los elementos.
  - 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 permite 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 XSLT. 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 JavaScript, que cuenta con una serie de funcionalidades para el 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 aplicar una única regla a cada nodo.

5.2. Descripción y utilización de estructura, sintaxis y plantillas

Estructura básica de una hoja de transformaciones

Una hoja de transformaciones tiene que estar 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, donde xsl corresponde al prefijo:

```
http://www.w3.org/1999/XSL/Transform
```

- 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:

- 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

Existe un grupo de instrucciones que nos van a permitir generar un gran número de salidas. Destacaremos los siguientes:

- **xsl:stylesheet y xsl:transform:** son elementos que podemos utilizar para 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.

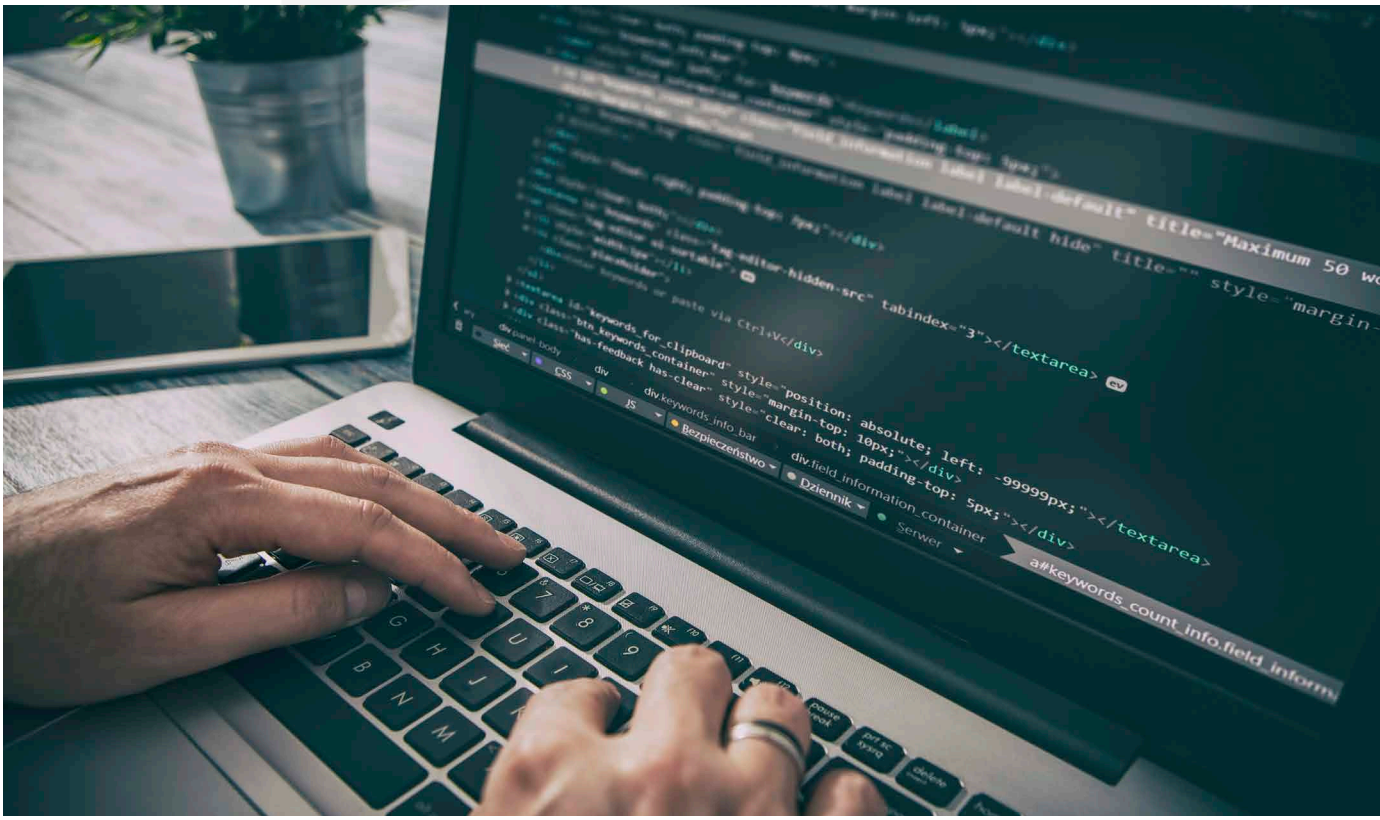
– Atributos optativos principales:

- **name**: asigna un nombre a una plantilla.
- **match**: señala el patrón *XPath* 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, sacaremos 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). Cuando las aplicamos de forma automática van 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 cada vez valores diferentes a la salida. En 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**: la podemos utilizar 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 lo 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 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 lo 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 emplear al ordenar.
- **data-type**: tipo de 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.

ponte a prueba

¿Cuál es el lenguaje que puede transformar aquellos documentos XML a diferentes formatos?

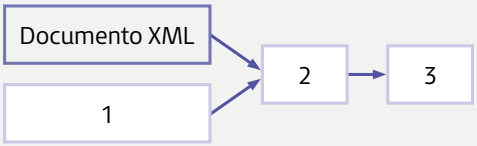
a) XSLT

b) XPath

c) XSL-FO

d) Ninguna opción es correcta

Completa el esquema de procesamiento XSLT.



a) 1: Hoja de transformaciones XSLT – 2: Procesador XSLT – 3: Documento transformado

b) 1: Procesador XSLT – 2: Hoja de transformaciones XSLT – 3: Documento transformado

c) 1: Procesador XSLT – 2: Hoja de transformaciones XSLT – 3: Documento inicial

d) 1: Documento inicial – 2: Procesador XSLT – 3: Hoja de transformaciones XSLT

5.3. Elaboración de documentación

Elaboración de documentos XML con hojas de estilo

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

El W3C aprobó en junio de 1999 la recomendación que define cómo vincular documentos XML con hojas de estilo CSS. De acuerdo con esta recomendación, se realiza mediante la instrucción *<?xml-stylesheet ?>* de forma similar a cómo 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:

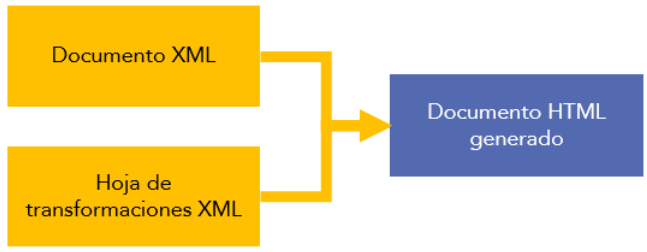
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="ejemplo.css"?>
<persona>
<nombre>Juan</nombre>
<localidad>Lleida</localidad>
</persona>
```

Generación de nuevos elementos y atributos

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

- **xsl:element**: permiten 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 elementos descendientes también se copiarán.
- **xsl:copy**: permite realizar la copia de un determinado nodo sin hacer la de sus descendientes.

  – Atributos optativos principales:

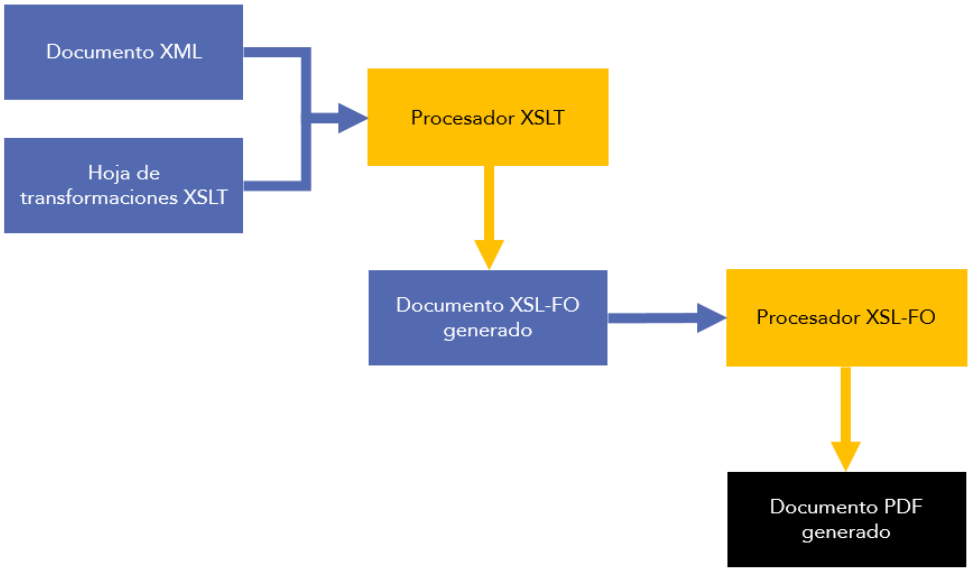
  ◦ **use-attribute-sets**: muestra una lista con los diferentes conjuntos de atributos separados por un espacio, que se pueden aplicar al nodo de salida.

5.4. XSL- FO

Como ya indicamos al comienzo de esta unidad formativa, **XSL-FO** (*XSL- Formatting Objects* u "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 Processor* - 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 transformaciones 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:

- 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>
```

- A continuación, generamos el documento que se podrá visualizar en formato pdf a partir del documento FO:

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

Objetos de formateo

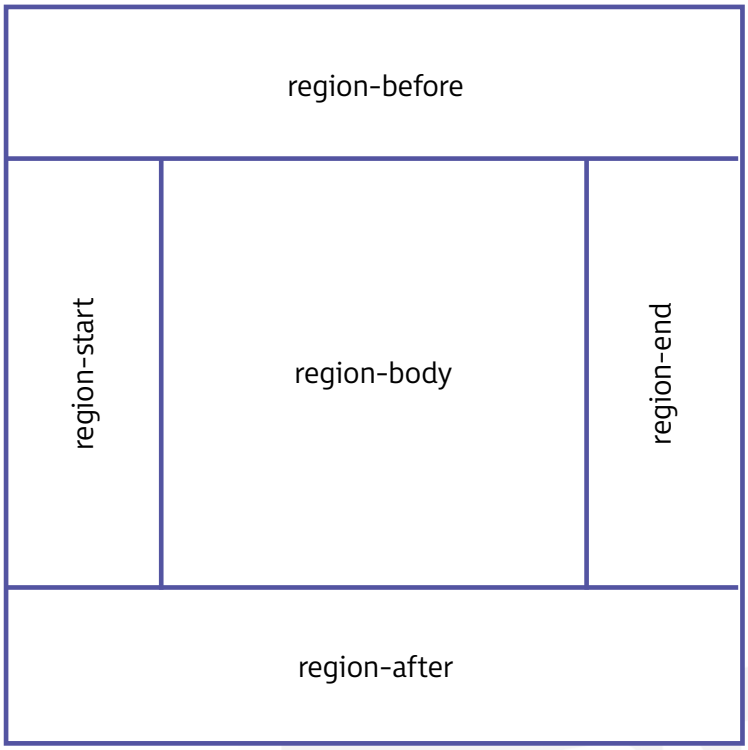
Las etiquetas que se utilizan en este lenguaje están siempre relacionadas con distintos elementos de maquetado, como páginas, párrafos, bloques y tablas. En este caso, XSL-FO utiliza distintas áreas rectangulares para visualizar la salida. Entre sus áreas más importantes, destacamos:

- **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 bloque 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 un texto dentro de las áreas de bloque. 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>:*

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

Entre sus principales prioridades, destacamos:

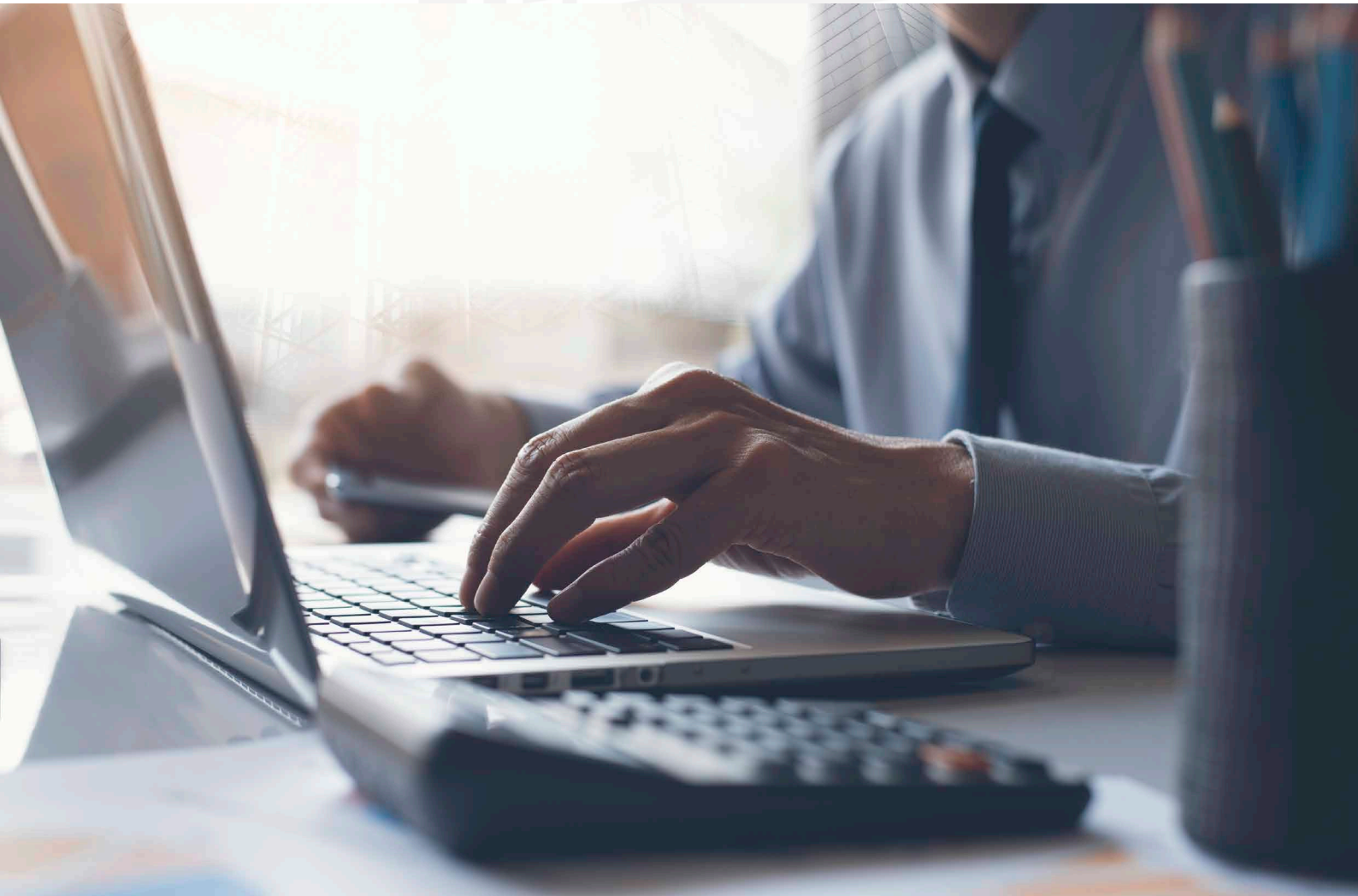
– **Maximum-repeats**: cuenta el máximo número de repeticiones que pueden aparecer.

- **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 a 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:

```
=<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 pueden repetirse en diferentes páginas. Entre sus principales prioridades, destacamos:

  – **flow-name**: para indicar dónde 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 fueran 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 asignadas las distintas etiquetas utilizadas en forma de marcadores. Estas etiquetas deben formar parte de un objeto *<fo:block>*.

```
Contenido::=(<fo:block> | <fo:block-contai-
ner> | <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-contai-
ner> | <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 a 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:ta-
ble-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:ta-
ble-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 permite 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 insertar una imagen.
- **fo:leader:** ofrece 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.

ponte a prueba

Dentro de los elementos básicos de una hoja de transformaciones, 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". ¿De qué elementos estamos hablando?

a) xsl: output

b) xsl: stylesheet

c) xsl: template

d) Ninguna de las opciones es correcta

Dentro de las instrucciones de control, permite repetir una lista de elementos para poder realizar diferentes transformaciones sobre ellos. ¿De qué estamos hablando?

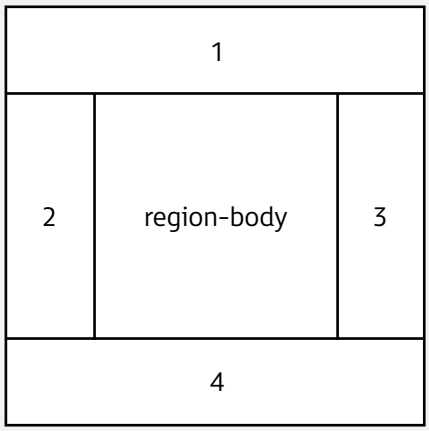
a) xsl: for- each

b) xsl: sort

c) xsl: if

d) Ninguna opción es correcta

En referencia a los objetos de formateo XSL-FO, coloca la región correspondiente en el número 4.



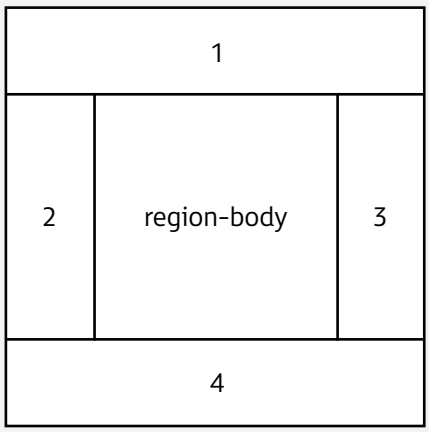
a) region-before

b) region- after

c) region-start

d) región-end

En referencia a los objetos de formateo XSL-FO, ¿cuál es la región correspondiente en el número 1?



a) region-before

b) region-start

c) region-end

d) region-after

6 GESTIÓN Y ALMACENAMIENTO DE INFORMACIÓN EN XML

En este tema vamos a estudiar cómo gestionar el almacenamiento de la información en documentos XML.

Para ello, vamos a analizar las ventajas e inconvenientes que tiene trabajar con la información en formato XML, qué tecnologías existen, como los tipos de lenguajes que trabajan con documentos XML y cómo podemos manipular la información en XML, es decir, cómo realizar búsquedas en un documento XML, cómo extraer información o cómo insertar información.

Veremos qué sistemas gestores de bases de datos relacionales más clásicos se han adaptado para poder almacenar y tratar información con XML (bases de datos con XML no nativas).

Por contra, también veremos qué sistemas gestores de bases de datos trabajan de manera nativa con XML, así como sus herramientas y técnicas para tratar la información en XML.

6.1. Sistemas de almacenamiento de información XML. Tecnologías

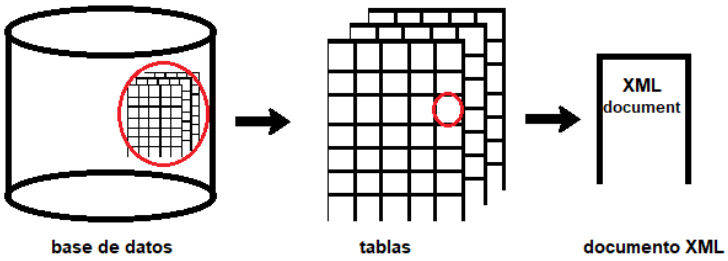
Las bases de datos se pueden definir como un sistema de almacenamiento de información relacionada entre sí, de manera estructurada, y accesible digitalmente.

Existen diversos tipos de bases de datos: relacionales, jerárquicos, en red… sin embargo, también podemos usar los documentos XML como un sistema estructurado y accesible para almacenar información, lo cual rompe por completo con la filosofía de las bases de datos tradicionales.

En una base de datos relacional nos encontraríamos con tablas, registros, etc., sin embargo, en un sistema de almacenamiento de información XML nos encontramos con elementos y atributos.

Por tanto, debemos tener en cuenta que si vamos a almacenar y tratar información de documentos XML podemos hacerlo:

- Con una base de datos relacional adaptada a la información XML (base de datos XML no nativa).
- Con una base de datos nativa XML, tecnología que está actualmente en desarrollo.



Las ventajas de usar una base de datos relacional adaptada para XML son:

- Las bases de datos relacionales poseen una estructura regular y homogénea, en contraposición a la irregularidad de la información XML.
- Las bases de datos relacionales están muy implantadas hoy en día. Luego con una BBDD adaptada a XML no será necesario cambiar todo el paradigma del sistema de información de una institución o empresa si esta quisiese o necesitase comenzar a trabajar con información en XML.

Las desventajas de usar una base de datos relacional adaptada a XML son:

- Posee una estructura mucho más densa y compleja (tablas, índices, *triggers*…).
- Las bases de datos relacionales estructuralmente difieren de la tecnología XML. Luego, la realización de una consulta XML en una base de datos relacional es más compleja.

6.2. Bases de datos relacionales con XML. Inserción de información

En apartados anteriores hemos podido ver que no es lo mismo un sistema gestor de bases de datos relacional adaptado para tratar información XML que un sistema de bases de datos XML nativo.

En este apartado vamos a ver algunas operaciones con bases de datos relacionales que poseen una implementación para XML.

Este tipo de bases de datos suelen centrarse en los documentos, de modo que gestionan información contenida en un documento estructurado, en este caso documentos XML.

Dos de los sistemas gestores de bases de datos relacionales adaptados a la información XML son:

- SQL Server.
- ORACLE.

SQL server

Posee una versión para el manejo de la información de tipo XML. Para ello, se crea un tipo de columna especial para el contenido de un documento XML y se tiene en cuenta el esquema XML de dicho documento XML (como los DTD o los *schematron*).

El tipo de campo especial que se usa para XML es XMLTYPE.

Podemos crear una tabla, mediante la sentencia **CREATE TABLE**, visto en módulos anteriores, donde uno de los campos (de las columnas) debe ser de tipo XMLTYPE. Por tanto, a la hora de crearnos un registro en una tabla, la información que va a la columna tipo XMLTYPE 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).

La sintaxis de una operación de inserción en SQL Server es la siguiente:

```
Insert
    Expresión (
                { as first | as last } Into | after | before
          )
```

Otra forma de ejecutar una consulta en este lenguaje es a través de la función ***XMLQuery()***, en la cual recibirá como parámetro la consulta, además del campo XML. Existe, por otra parte, la cláusula ***XMLTable()***, mediante la cual podemos tratar el resultado de una consulta con XQuery.

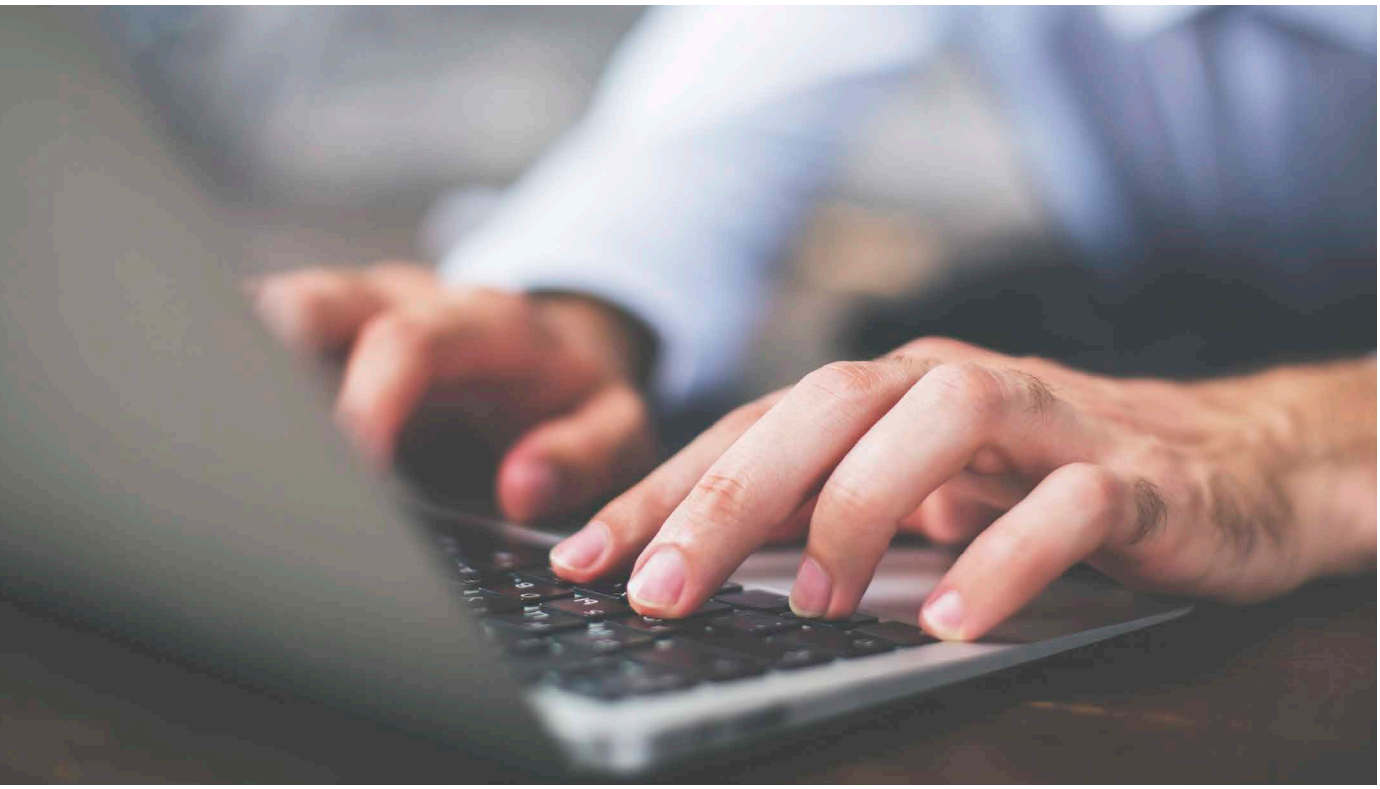
Podemos crear las tablas de tipo de XML, al igual que las bases de objetos a partir de las sentencias en XQuery.

```
CREATE TABLE nombre_tabla OF XMLTYPE
```

De esta 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.



6.3. Técnicas de búsqueda de información en documentos XML

Para acceder y consultar datos en un documento XML como lo haríamos en una base de datos relacional, debemos disponer de un lenguaje de consulta específico, que sea capaz de recorrer el documento XML y realizar ciertas operaciones básicas.

Dos de los lenguajes de consultas más populares para XML son los siguientes:

- **XPath**: lenguaje de consulta bastante sencillo, que dispone de una gran cantidad de expresiones que nos permiten recorrer el árbol de nodos en un documento XML y acceder a sus diferentes partes.

- **XQuery**: es otro lenguaje de consulta para documentos XML con aspectos añadidos de funcionalidad, como la de manipulación de la información en un documento XML.

A continuación, vamos a estudiar ambos lenguajes de consulta y manipulación de información en formato XML.

Debemos tener en cuenta que una no es alternativa a la otra, sino que se complementan. Es decir, XPath es una forma de recorrer los nodos de una estructura XML y XQuery utiliza XPath para recorrer los nodos. Además, permite realizar una serie de acciones para manipular esa información.

ponte a prueba

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. ¿Qué es Xpath?

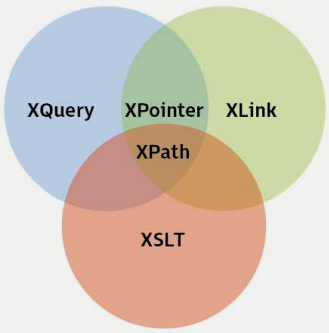
a) Un lenguaje diseñado para acceder a documentos XML

b) Un lenguaje diseñado para transformar documentos XML

c) Un lenguaje diseñado para dar formato de salida a los documentos XML

d) Todas las opciones anteriores son correctas

¿Cuál de las siguientes afirmaciones son ciertas en XQuery?



a) Permite realizar consultas

b) Permite extraer información

c) Es orientado a bases de datos

d) Permite procesar información

e) Todas las opciones anteriores son correctas

¿Qué significa el siguiente código de expresión en lenguaje XPath: e1/e2?

a) Elemento e2 es hijo directo de su padre e1

b) Elemento e1 es hijo directo de su padre e2

c) Elemento e2 es el directorio raíz

d) Todas las opciones son correctas

¿Qué significa el siguiente código de expresión en lenguaje XPath: @atrib?

a) Atributo que se identifica con el nombre atrib

b) Atributo que se identifica con el nombre atributo

c) Atributo que se identifica con el nombre Atrib

d) Todas las opciones son correctas

6.4. Lenguajes de consulta, extracción y manipulación de la información en XML

6.4.1. XPath

XPath es un 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 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 puede 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 el 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, comienza a derivar el resto. Cada documento tiene un único nodo raíz y se caracteriza por ser un nodo que no tiene padre y por 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 primitivos más básicos son *string* para las cadenas de caracteres; *number*, para representar la información numérica; *boolean* representa el tipo de datos (Sí/No o True/False) y, por último, como estamos ante un lenguaje similar a DOM (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, vemos 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 qué 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).
.	Es el nodo actual en el que nos encontramos, similar al funcionamiento del sistema operativo Linux, pero llevado a lenguaje de marcas.
..	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.

BUSCA EN LA WEB

Vistas las expresiones más básicas de este lenguaje, el siguiente paso es la ejecución de las mismas. Lo podemos realizar de 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

La forma alternativa es la descarga de un entorno de desarrollo como BaseX. La última opción es 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 queremos es trabajar con los nodos de una determinada expresión.

Veamos 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 tal que así: */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, debemos escribir */coches/@matrícula/data()* o bien, */coches/@matrícula/string()*, ya que en algunos analizadores la primera forma no se puede ejecutar.

Acceso a elementos con rutas simples

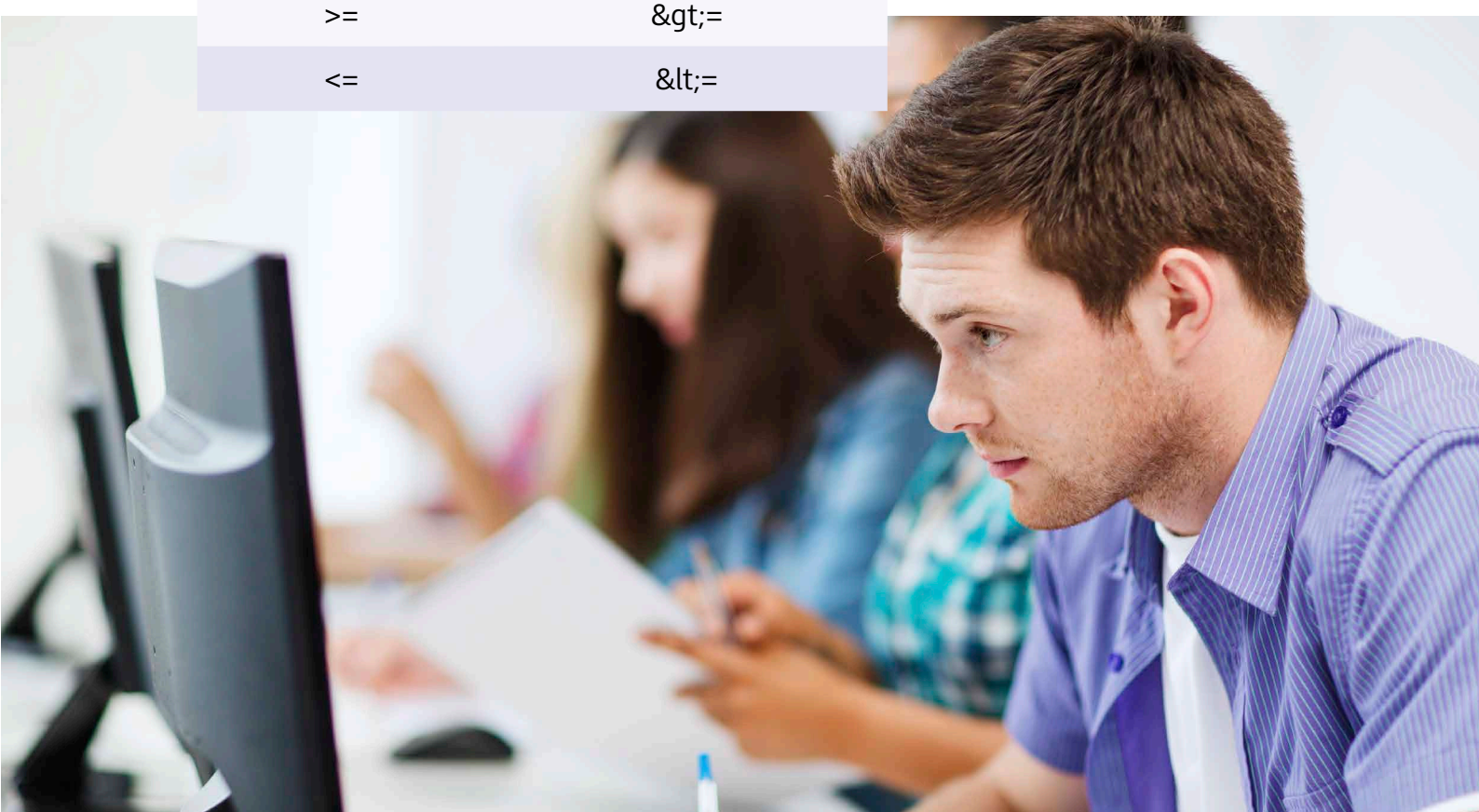
Son una serie de rutas a las que podemos acceder mediante un directorio absoluto, es decir, empezando desde el nodo raíz del sistema, devolviendo el conjunto de nodos que se encuentre en dicha ruta.

Elementos del lenguaje

Avanzando en la sintaxis del lenguaje XPath vamos a generar expresiones más complejas a partir de los elementos básicos ya vistos. Esto 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:

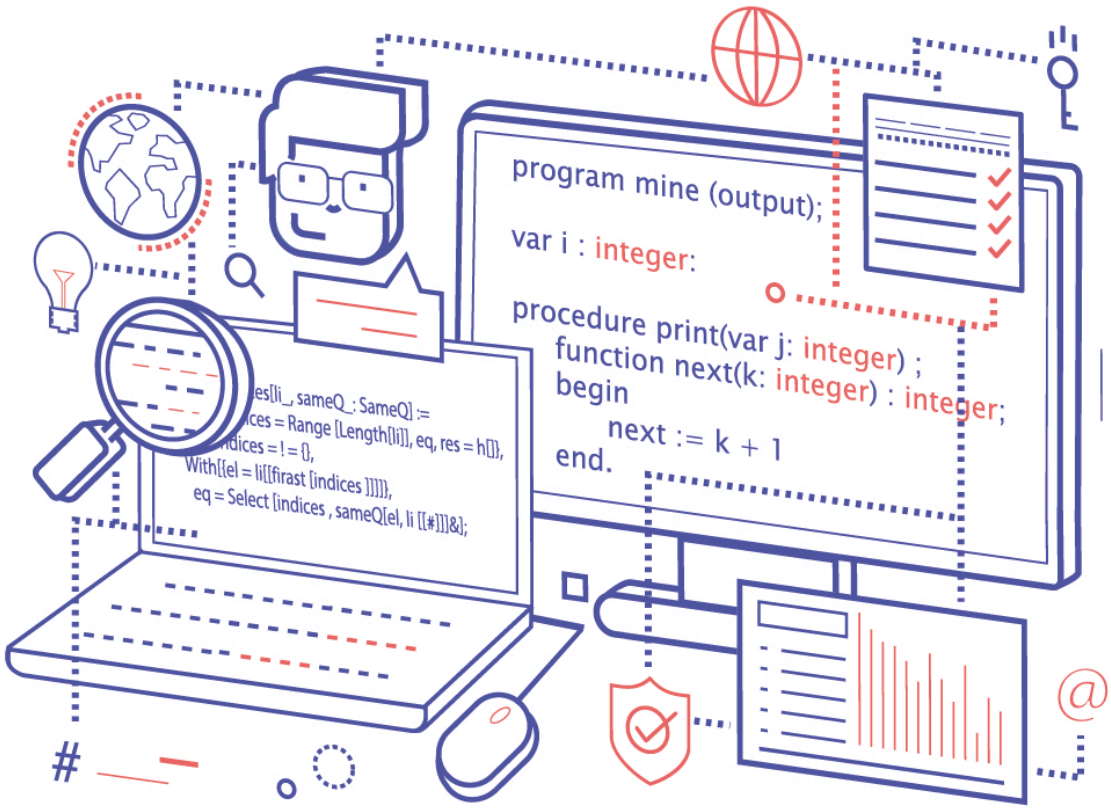
Función	Devuelve	Ejemplo
substring()	Subcadena	Substring('Beatles', 1, 4) = Beat
starts-with()	Cadena comienza por	starts-with('XML','X') = true
contains()	Cadena contiene	contains('XML','XM') = true
normalize-space()	Espacios normalizados	Normalize-space('The end') = 'The end'
translate()	Cambia caracteres 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 descendiente 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 varios tipos de valores:

- **Literales:** que van entre dobles comillas para acceder a ese valor concreto.
- **Variables**: para utilizar variables en Xpath debemos usar el símbolo del $, de este modo podemos referir a un valor contenido dentro de la propia variable.
- **Valores recuperados:** permiten acceso mediante otras expresiones.

Ejemplo de valores literales:

```
Acceder a un coche cuya matrícula es 0011
/coche/matrícula[@valor=”0011”]
```

Ejemplo de referencia a una variable:

```
/coche/$variable
```

Ejemplo de otras expresiones:

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

```
/coche/marca[@valor=”Opel”]/persona/@dni
```

PARA + INFO

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. Esta es una condición indispensable.

Función	Uso
shelf::*	Devuelve el propio nodo de contexto.
child::	Devuelve los nodos *hijo* del nodo de contexto.
parent::*	Devuelve el nodo padre del nodo contexto.
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 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 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.

A continuación, vamos a mostrar lo que podría ser un ejemplo de documento XML, para recorrer y localizar nodos con XPath:

```
<?xml versión=”1.0” encoding=”UTF-8” ?>
<biblioteca>
    <libro>
          <titulo>  100 años de soledad </titulo>
          <autor> Gabriel García Márquez</autor>
          <editorial>Espasa S.A.</editorial>
          <genero>Realismo mágico </genero>
          <año> 1966 </año>
    </libro>
    <libro>
          <titulo>Africanus</titulo>
          <autor> Santiago Posteguillo</autor>
          <editorial> Ediciones C </editorial>
          <genero> Novela histórica </genero>
          <año> 2006 </año>
    </libro>
    <libro>
          <titulo> 1984     </titulo>
          <autor> George Orwell </autor>
          <editorial>Seck&Warg</editorial>
          <genero> Distopía </genero>
          <año> 1949  </año>
    </libro>
</biblioteca>
```

A partir del documento XML anterior:

Con la siguiente expresión referimos a todos los nodos de autores de libros:

```
/biblioteca/libro/autor
```

Con la siguiente expresión referimos a la editorial del segundo libro:

```
/biblioteca/libro[2]/editorial
```

Con la siguiente expresión referimos a los textos contenidos en el interior de todos los nodos correspondientes a los títulos de los libros:

```
/biblioteca/libro/titulo[text()]
```

6.4.2. XQuery

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

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*, *orderby*, etc. y, por otra parte, también podemos compararlo al lenguaje XPath descrito anteriormente, 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.

Al ser un lenguaje de expresiones, XQuery también se compone 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, booleanos, 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, orderby, 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 a tratar.
  - **let**: permite declarar variables para que, a continuación, podamos asignarle valores.
  - **where**: acompaña la sentencia *for*, permitiendo introducir una condición para que la estructura iterativa se repita mientras se cumpla dicha condición.
  - **orderby**: 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.

6.5. Sistemas gestores de bases de datos nativos XML. Herramientas para el tratamiento y almacenamiento de la información en XML

Las bases de datos para XML son una alternativa a las bases de datos relacionales, de tal forma que podemos sustituir la filosofía de tablas y campos por información estructurada en documentos XML, con sus correspondientes elementos y atributos.

Una de las herramientas más útiles para las bases de datos en XML es el software *BaseX*, que como motor de bases de datos XML puede trabajar con dos lenguajes de consulta XML, como XML o XQuery.

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 XM. Así, mientras 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 bases de datos nativos podemos encontrar **BaseX** y **eXist**.

BaseX

BaseX es un 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. Con esta herramienta utilizaremos los lenguajes XPath o XQuery para poder realizar las pertinentes consultas y, de esta forma, extraer 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

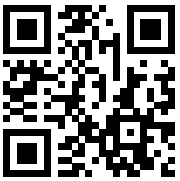
En la interfaz gráfica de esta herramienta podemos visualizar fácilmente los distintos apartados para realizar las operaciones, como la barra de herramientas, el editor de consulta, las visualizaciones de los datos y la línea de comando. En esta última, ejecutaremos las cláusulas para poder trabajar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

En las bases de datos nativas existen diferentes técnicas para la realización de consultas. Podemos ejecutar tres tipos de sentencia:

- **Opción *Command***: en este apartado podemos indicar 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 XPath que detallaremos más adelante.
- **Opción *XQuery***: indicaremos los diferentes comandos para realizar las consultas. Utilizaremos el lenguaje XQuery.

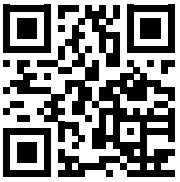
BUSCA EN LA WEB

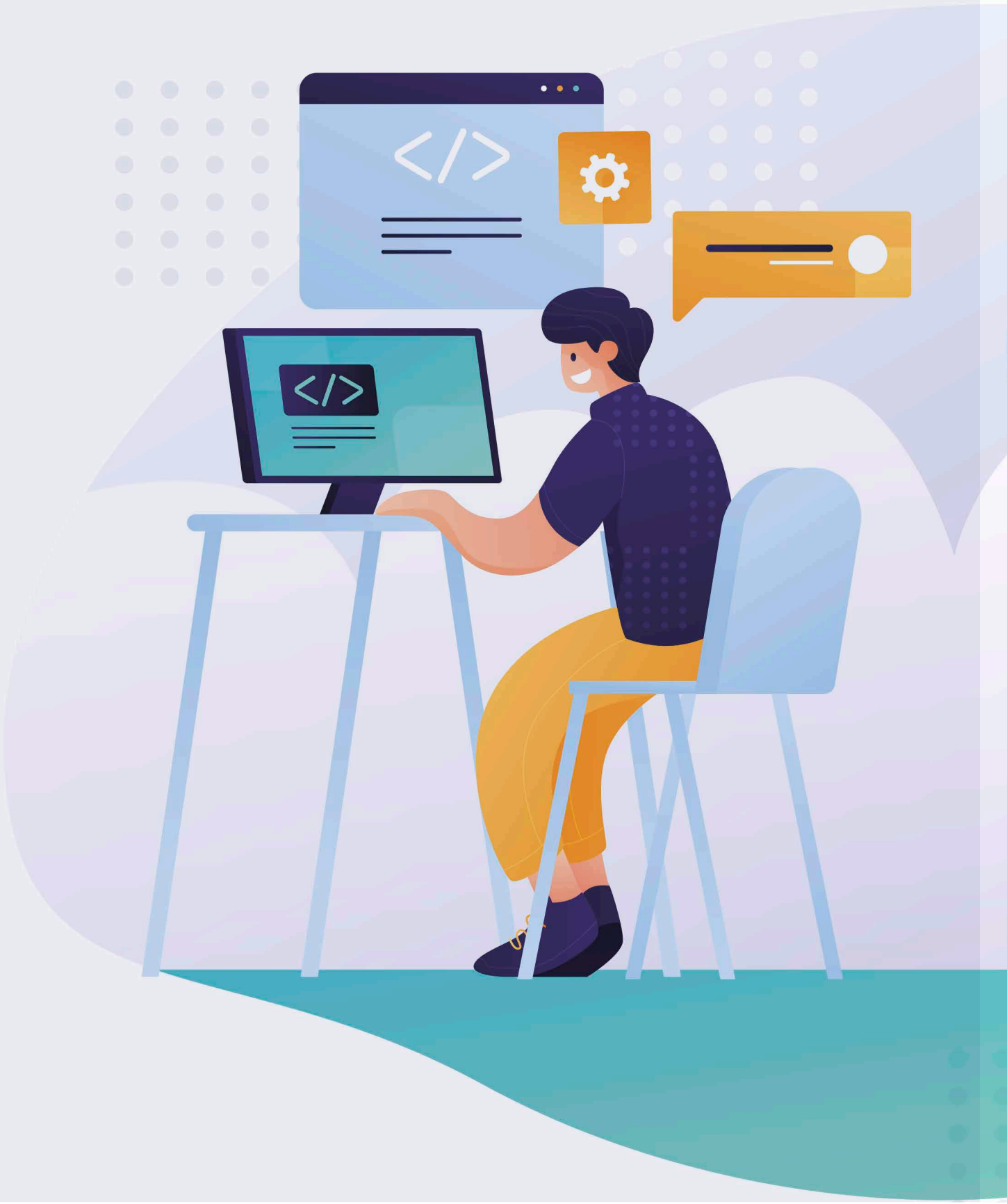
BaseX
http://basex.org



BUSCA EN LA WEB

eXist (software privativo)
http://exist-db.org





7

SISTEMA DE GESTIÓN EMPRESARIAL

En este tema nos centraremos en los sistemas de gestión empresarial. Este tipo de herramientas de software son útiles, no solo para obtener una visión global de una empresa o institución, sino también para gestionarla de manera eficiente.

El concepto de *sistemas de gestión empresarial* abarca diferentes puntos, desde los habituales ERP (*enterprise resource planning o* sistemas de gestión de recursos empresariales) hasta Los KM (*knowledge management* o gestión del conocimiento), pasando por muchos otros elementos de gestión empresarial.

Gracias a este tipo de herramientas de software podremos tomar decisiones más acertadas para el beneficio de la empresa u organización.

También vamos a conocer las distintas herramientas de **BI** (*business intelligence* o inteligencia del negocio), que nos permiten almacenar aquella información sobre los diferentes aspectos de una actividad determinada.

Como ya adelantamos, el principal factor de las aplicaciones de inteligencia del negocio son los **ERP**, sistemas modulares integrados de gestión empresarial.

Uno de los componentes más importantes de los ERP son los **CRM**, que son un tipo de aplicaciones destinadas a la gestión de clientes.

7.1. 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 ordenadores para poder registrar en ellos toda la información que se considere necesaria. Hablamos, por ejemplo, de los archivos de texto, de sistemas gestores de bases de datos bastante sofisticados, sistemas expertos, sistemas CBR (*case base reasoning* o razonamiento basado en casos), etc.

Los sistemas de información han ido evolucionando con el tiempo hasta conseguir la automatización de bastantes tareas.

Desde el momento en que hemos contado con una gran cantidad de datos para procesar y recuperar, hemos tenido la oportunidad de 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 aprender en profundidad acerca del posicionamiento de

Concepto ERP & Business Intelligence

youtu.be/E8LQjEKvI5I

estos procesos, además de 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 cosas, automatizar y agilizar procesos, registrar resultados y favorecer la extracción de conclusiones.

A continuación, vamos a citar algunas familias de aplicaciones con sus particularidades:

- **ERP** (*enterprise resource planning* o planificación de recursos empresariales): aplicaciones integrales y modulares.
- **CRM** (*customer relationships management* o gestión de relaciones con clientes): aplicaciones con un amplio rango de funcionalidades.
- **CM** (*change management* o gestión del cambio): distintas herramientas que tienen como fin facilitar a los empleados los cambios que tengan que modificar.
- **BA** (*business analytics* o 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* o extraer, transformar y cargar datos): estas aplicaciones tienen como función fusionar aquellos datos de diferentes orígenes.
- **SFA** (*sales force automation system* o 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* o gestión de procesos de negocio).
- **MRP** (*material resource planning* o planificación de los recursos materiales).
- **SRM** (*supplier relationship manager* o gestión de relaciones con proveedores).
- **KM** (*knowledge management* o gestión del conocimiento).

ponte a prueba

¿Qué son los sistemas ERP?

a) Distintas herramientas que tienen como fin facilitar a los empleados los cambios que tengan que modificar.

b) Las distintas herramientas que vamos a emplear para analizar datos y resultados hasta que podamos sacar las conclusiones oportunas.

c) Aplicaciones con un amplio rango de funcionalidades.

d) Aplicaciones integrales y modulares.

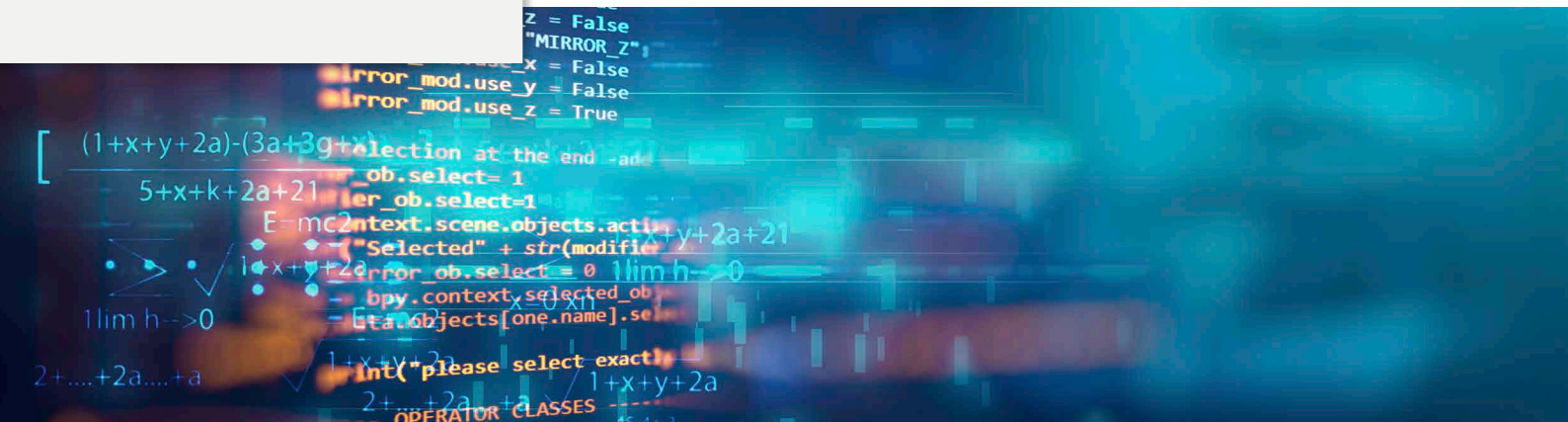
¿Cómo se definen las aplicaciones de inteligencia de negocio?

a) Aplicaciones que se manejan de forma inteligente y hacen todo por el usuario.

b) Aplicaciones que consiguen, entre otras cosas, automatizar y agilizar procesos, registrar resultados y favorecer la extracción de conclusiones.

c) Procesos del mundo de los negocios donde todo se gestiona y centraliza en una misma aplicación.

d) No existen las aplicaciones de inteligencia de negocio, es un término usado para vender ciertas aplicaciones de una forma más atractiva.



7.2. Sistemas de Gestión de recursos empresariales (ERP)

La función principal de los ERP es facilitar el flujo de información que hay entre los trabajadores y los elementos de una empresa u organización. Gracias a los ERP, la operatividad y comunicación entre los distintos componentes de una empresa es más fluida.

Los **ERP** suelen ser muy versátiles, en el sentido de que son multiplataforma, es decir, pueden funcionar en diferentes sistemas operativos (Linux, Windows, Mac…). 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 la opción de combinarlos.
- **Parametrizable**: consiguen adaptarse a las diferentes necesidades de una empresa.
- **Escalable**: ofrecen capacidad de crecimiento.

Algunos de los ERP más empleados actualmente son:

BUSCA EN LA WEB

Odoo

https://www.odoo.com/es_ES/

BUSCA EN LA WEB

SAP

http://www.sap.com/spain

A continuación, vamos a centrarnos con más detalle en estos ERP.

7.2.1. SAP (*SYSTEM ANWENDUNGEN AND PRODUCTE* O 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 (*Financial*)**: 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.

Mediante la codificación de programas encargados de realizar una función determinada, los ERP se adaptan a las diferentes necesidades de la empresa. SAP incorpora un lenguaje de programación de cuarta generación propio. Es 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.

7.2.2. Odoo

Odoo es un ERP que ofrece una alternativa de software libre a la privativa SAP. Antiguamente era conocida como OpenERP.

Oddo, concretamente, dispone de dos versiones de ERP:

una versión de licencia libre, para la comunidad (licencia LGPL) y una versión privativa (comercial, de pago).

El sistema de gestión de recursos empresariales de Odoo cuenta, entre otros, con estos módulos disponibles:

- **CRM (*Customer relationship management*)**: módulo enfocado al cliente.
- **POS (*Point of sale*)**: punto de venta.
- **Gestión de proyectos**: módulo que organiza, analiza y planifica los proyectos.
- **Hoja de horas**: módulo que monitorea los tiempos de productividad.
- **Inventario**: módulo que gestiona el material de los almacenes.
- **Compra**: gestiona módulo que gestiona los pedidos a los almacenes.
- **Facturación**: módulo que gestiona los contratos, facturas, etc.
- **Contabilidad**: módulo similar al de facturación que gestiona la contabilidad de la empresa, realiza facturas electrónicas, activos, pasivos…
- **MRP (*Material requirements planning*)**: módulo de producción. Entre otros objetivos, se calculan los tiempos de manufactura.

7.3. Software de gestión de relaciones con los clientes (CRM)

CRM (*customer relationship management* o gestión de relaciones con clientes) es un software informático orientado a la relación y comunicación con los clientes.

Además de ser un software de gestión empresarial, influye y determina enormemente en la visión que un cliente tiene de su proveedor. Su función principal es mejorar la relación con los clientes y mantenerlos.

Existen dos tipos de CRM. Los que son un módulo más dentro de un ERP y los independientes. Las tareas principales que realizan los CRM son:

- Almacenar datos.
- Lanzar ofertas.
- Realizar informes.
- Llevar a cabo campañas publicitarias.

Algunos CRM son:

- **Salesforce.**
- **Zoho CRM.**
- **PipeDrive.**
- **SugarCRM.**

BUSCA EN LA WEB

SugarCRM

http://www.sugarcrm.com

A continuación, vamos a ver con más detenimiento el CRM **SugarCRM**.

De la misma forma que los ERP, cuenta con una serie de módulos, que son:

- Ventas.
- Marketing.
- Servicio al cliente.

Y estas son sus características principales más destacadas:

- 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: Windows, Linux y Mac OS.

Podemos ejecutar esta aplicación en 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 por correo electrónico y 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). Una vez registrados, tenemos la oportunidad de elegir un perfil entre los cinco que tenemos disponibles. A partir de ahí, ya tendremos acceso a gestión de ventas, director de ventas, director de marketing, representante de clientes o administrador de SugarCRM.

Además, también podemos seleccionar el idioma en el que queremos realizar la demostración.

7.4. Instalación de un sistema de gestión empresarial

La gran mayoría de sistemas de gestión empresarial requieren una instalación en una máquina o máquinas locales para funcionar correctamente. En este punto vamos a aprender a instalar un ERP con licencia de software libre: Odoo.

Para ello, lo primero que debemos hacer es ir a la página de Odoo, donde se ofrece el ERP de software libre para la comunidad.

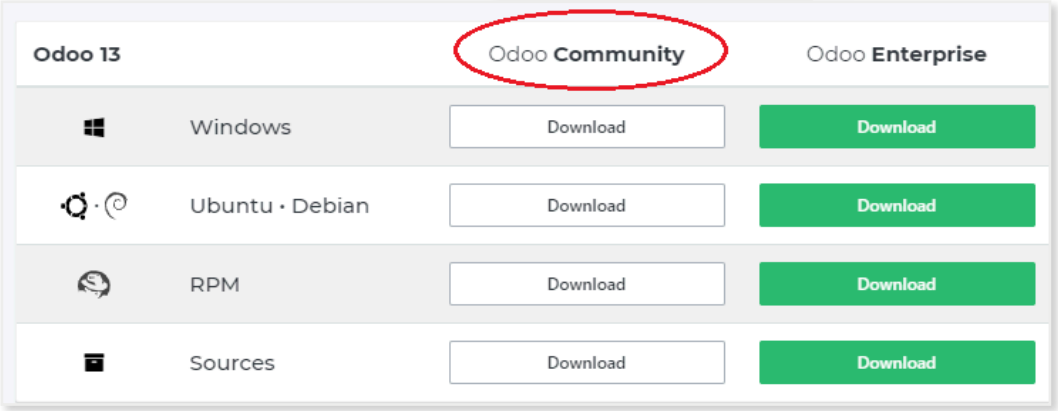
BUSCA EN LA WEB

Versión de licencia libre de Odoo

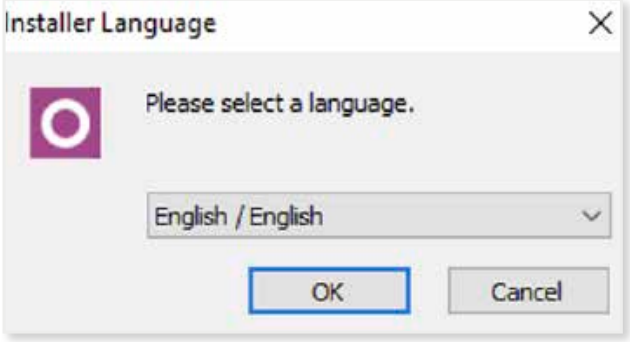
https://www.odoo.com/es_ES/page/community

Para descargarlo, Odoo nos pedirá algunos datos generales. No es necesario cumplimentarlos en su totalidad, pero algunos sí serán requeridos (el teléfono por ejemplo, no es necesario).

En esa misma página, pero más abajo, están los enlaces de descarga del software. Haremos clic en el botón de descarga correspondiente, en función del sistema operativo de la computadora donde lo queramos instalar. En cualquier caso, deberá ser una descarga de la sección OdooCommunity (licencia libre).

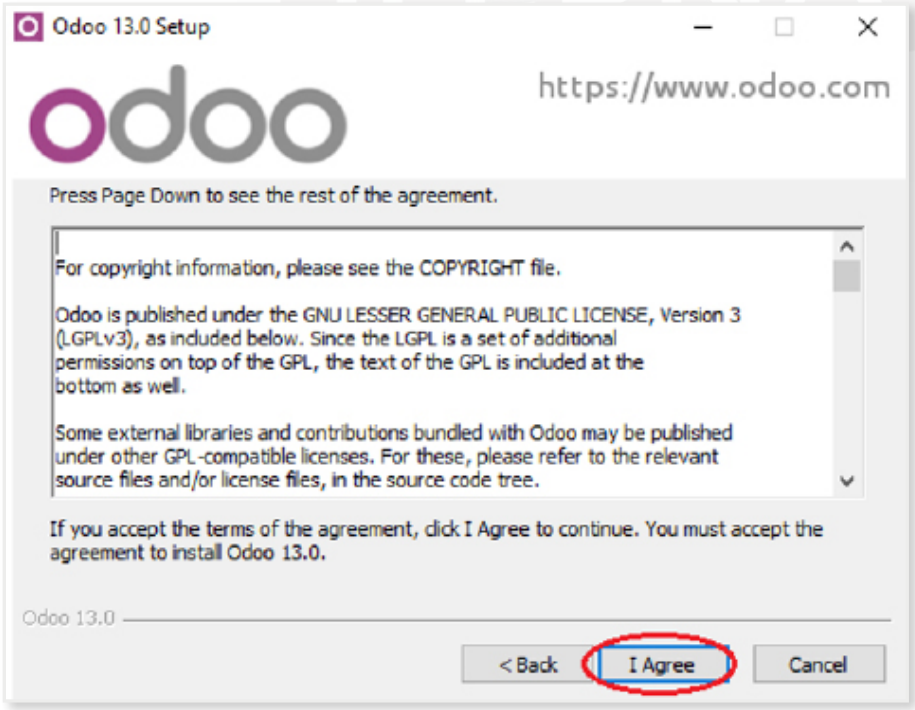


Una vez se haya descargado el archivo de instalación, lo ejecutaremos. Lo primero que nos pedirá es seleccionar el idioma disponible. En la versión de Odoo 13 (la más actualizada en el momento de esta edición del libro de la asignatura), deberemos elegir entre inglés y francés.

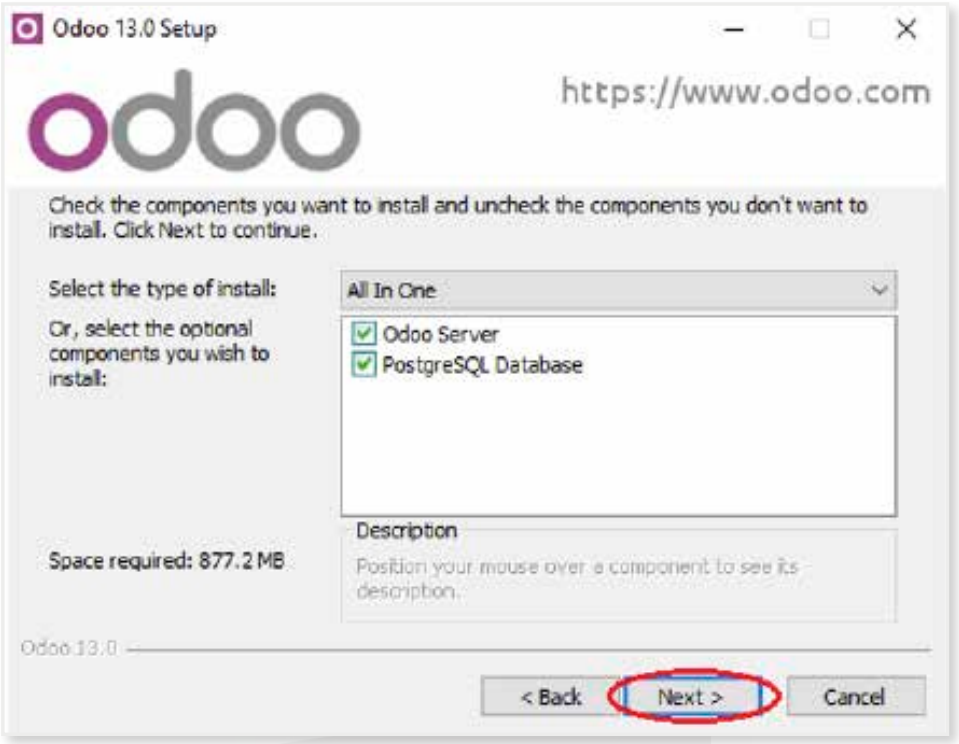


La instalación mostrará un mensaje de bienvenida. Deberemos hacer clic en *Next*.

Después, deberemos leer y aceptar los términos de uso del software.

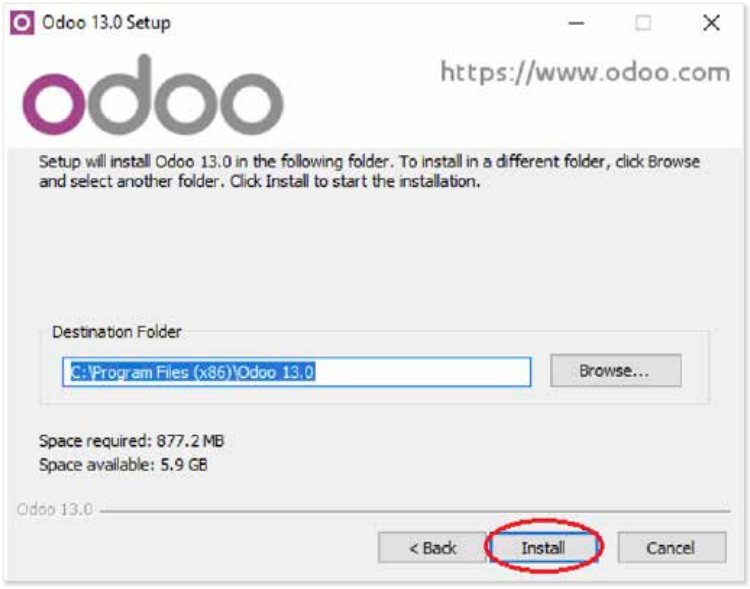


El siguiente paso es elegir los componentes que deseamos instalar. En nuestro caso, elegimos *All in One*, es decir, todos los componentes con sus módulos por defecto.

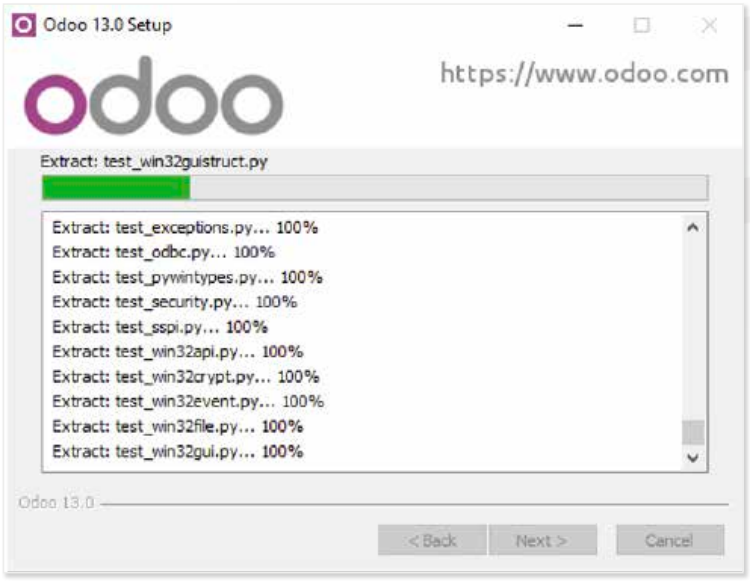


Indicamos los datos del puerto a utilizar, el nombre de usuario y la contraseña (el dato del puerto puede variar en función de la computadora y el sistema operativo).

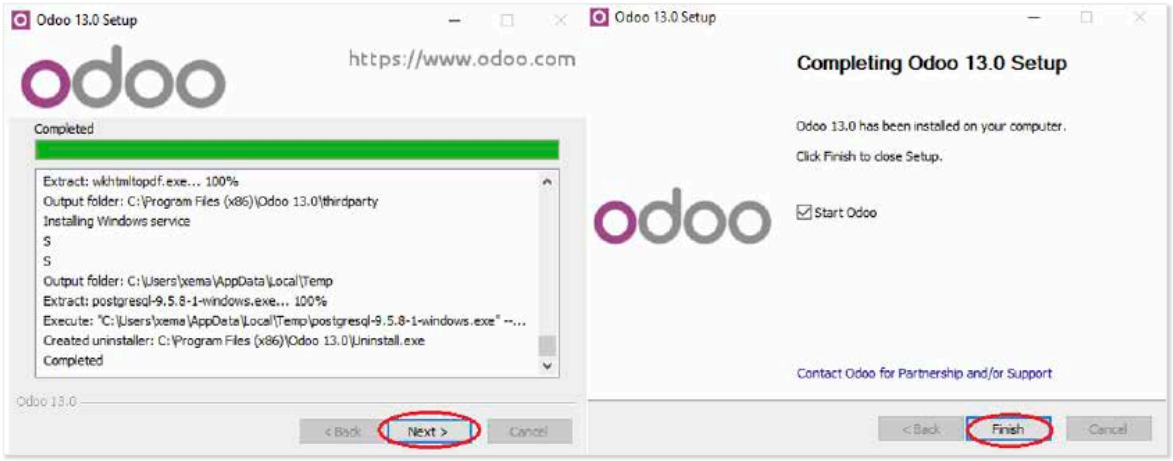
Indicamos en qué directorio deseamos instalar el software ERP.



Tras ello, comenzará la instalación, que podrá demorarse unos minutos.



Una vez terminado el proceso de instalación, haremos clic en *Next* y *Finish*.



ponte a prueba

La gestión empresarial es la actividad que tiene como objetivo principal aumentar la productividad y competitividad de una empresa.

a) Verdadero.

b) Falso.

¿Cuál de los siguientes programas es ejemplo de EPR?

a) SAP.

b) Odoo.

c) OpenBravo.

d) Todas las opciones son correctas.

¿Cuál de las siguientes opciones es una característica/as de la planificación de recursos empresariales (ERP)?

a) Integral.

b) Modular.

c) Parametrizable.

d) Escalable.

e) Todas las opciones son correctas.

Indica cuál de las siguientes opciones es una desventaja de los ERP.

a) La instalación y puesta en marcha de un sistema ERP es una inversión que cuesta dinero y tiempo.

b) Optimización de los procesos que se desarrollen en la empresa.

c) Análisis del estado de la empresa desde una visión global.

d) Acceso a toda la información de una empresa de manera modular y basada en roles.

Indica los módulos que podemos encontrar en un ERP.

a) Módulo de Recursos Humanos.

b) Módulo de logística.

c) Módulo de finanzas.

d) Módulo de ventas.

e) Módulo de facturación.

f) Todas las opciones son correctas.

¿Cuáles son las principales tareas de la gestión de relaciones con el cliente (CRM)?

a) Almacenar datos.

b) Lanzar ofertas.

c) Realizar informes.

d) Llevar a cabo campañas publicitarias.

e) Todas las opciones son correctas.

7.5. Adaptación y configuración de un ERP. Integración de módulos

Cuando realizamos la instalación de un ERP en una empresa, es decir, cuando instalamos en las computadoras de una empresa u organización un ERP, debemos darle mucha importancia a la adecuación del software a la empresa u organización.

Esto implica diversos aspectos:

- Darle al ERP una imagen y estilo corporativo de la empresa u organización.
- Colocar el logo y estilo de la empresa en el ERP.
- Asociar el e-mail al ERP.
- Crear los usuarios necesarios en el ERP que correspondan a cada trabajador.
- Configurar los permisos de cada usuario en función de sus necesidades.

Además, debemos elegir los módulos adecuados que la empresa u organización va a necesitar de todo el repertorio que el ERP nos ofrece. Por tanto, esto implica tanto no instalar módulos innecesarios, como no dejarse ninguno esencial.

Por ejemplo, para una empresa con muchos trabajadores será conveniente tener un módulo de Recursos Humanos en el ERP, pero tal vez para una empresa muy pequeña no sea tan necesario.

Por otro lado, una empresa que posea un gran almacén, tendrá necesidad de llevar un control de inventario, pero otra empresa de otra naturaleza podría no necesitarlo.

El software ERP normalmente nos ofrece una serie de módulos predefinidos a instalar fácilmente, pero también es posible diseñar y desarrollar nuestros propios módulos a medida, ya sea por terceros o por desarrolladores de la misma empresa u organización.

7.6. Elaboración de informes

En una empresa u organización se manejan infinidad de datos de todo tipo, formato y departamentos. Es conveniente organizar la información y, para ello, los ERP nos ofrecen la posibilidad de crear informes personalizados.

El uso de los informes ayuda a mejorar la productividad de la empresa, de modo que el análisis de la información está muy influenciado por los informes presentados a trabajadores y directivos que toman decisiones.

Hay informes de todo tipo. Desde aspectos internos de la empresa, como material almacenado, información de los trabajadores, horas de productividad, compras y ventas realizadas, hasta datos externos a la empresa, como el volumen del mercado, posibilidad de futuros clientes, etc.

Existen dos maneras de crear un informe: de cero, es decir, realizar un informe nuevo desde el principio o modificando un informe ya existente.

Gracias a la usabilidad que presentan los ERP no es necesario disponer de muchos conocimientos técnicos para la creación de informes. Esto ayuda muchísimo a la integración en la empresa.

7.7. Integración con aplicaciones ofimáticas. Exportación de la información

En muchas ocasiones, cuando instalamos un nuevo ERP en una empresa aparece la necesidad de lidiar con el paradigma de trabajo anterior para implantar el nuevo. Esto suele ocurrir con las herramientas ofimáticas, como por ejemplo Microsoft Office, Libre Office u otras.

Para ello, los ERP permiten la integración del mismo ERP con otras herramientas, como las ofimáticas. Esta propiedad nos permitirá trabajar de manera más integrada debido a que el flujo de información entre el ERP y la aplicación ofimática será automático.

Además, la mayoría de programas ERP permiten almacenar los documentos creados con la aplicación ofimática dentro del mismo ERP.

Para realizar esta integración, podemos hacerlo instalando el módulo pertinente en el ERP y añadiendo el complemento necesario en la aplicación informática.

Esta facultad de muchos ERP nos facilita la exportación de datos del ERP a la aplicación ofimática o viceversa. Por ejemplo, podríamos tener la necesidad de pasar la información de una hoja de cálculo al ERP, por ejemplo, con un listado de clientes o los albaranes de un día.

BUSCA EN LA WEB

Por ejemplo, Odoo posee un complemento para realizar la conexión con Microsoft Office 365:

apps.odoo.com/apps/modules/11.0/odoo_office365/

BIBLIOGRAFÍA / WEBGRAFÍA

- Juan Manuel Castro Ramos, José 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.
- https://docs.oracle.com
- https://www.odoo.com/es_ES/
- https://es.wikipedia.org/wiki/XHTML

solucionario

1.3. Herramientas de edición

¿En cuántos tipos de marcas podemos dividir los lenguajes de marcas?

d) Todas las opciones son correctas.

¿Qué tipos de herramientas podemos utilizar en un documento XML?

d) Todas las opciones son correctas.

¿De qué forma se organiza la información de un documento XML?

a) Jerárquica.

¿Qué nombre falta en la etiqueta p1?

```
<profesor>
      <nombre> Roberto </nombre>
      <apellido> Nadal <p1>
</profesor>
```

a) /apellido.

1.5. Elaboración de documentos XML bien formados

En el siguiente ejemplo XML, ¿cuál es el elemento raíz?

```
<alumno>
      <nombre>Roberto</nombre>
      <segundo_nombre>Mesa</segundo_
nombre>
      <edad>19</edad>
</alumno>
```

a) alumno.

¿Cuántos "hermanos" tiene la etiqueta <nombre> en el siguiente ejemplo XML?

```
<alumno>
      <nombre>Roberto</nombre>
      <segundo_nombre>Mesa</segundo_
nombre>
      <edad>19</edad>
</alumno>
```

b) 2.

Se definen una serie de reglas que asignan unas restricciones sobre la estructura del documento XML.

a) Definición tipo de documento (DTD).

2.2. Identificación de etiquetas y atributos de HTML

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

a) Verdadero.

Definen una serie de elementos que forman el léxico del lenguaje HTML. Se encuentran entre los signos de menor que (<) y mayor que (>). ¿De qué estamos hablando?

a) Etiquetas.

Las etiquetas abiertas cuentan con una única palabra reservada para indicar el inicio y fin de la etiqueta a la vez. Señala cuál de las siguientes etiquetas son de tipo abierto.

d) Todas las opciones son correctas.

2.3. Estructura básica

Es la etiqueta que se utiliza para el inicio del cuerpo.

a) <body>.

¿Cómo se representan los comentarios en lenguaje HTML?

a) <!- -Esto es un comentario- ->.

2.4. Elementos de un HTML

¿Cuál de estas etiquetas muestra el texto más grande VISUALMENTE en HTML?

d) <h1>Texto MUY GRANDE </h1>.

¿Cómo se escribe un texto en forma de párrafo mediante el lenguaje HTML?

a) <p>.

¿Cómo se llama el atributo que nos ofrece la posibilidad de crear un hiperenlace? Debemos indicar una URL que va a ser la que queramos acceder al hacer clic en el hiperenlace.

a) href.

solucionario

2.6. Versiones de HTML y XHTML

Queremos listar de forma numerada los nombres que hay a continuación, ¿qué nombre le pondremos a la etiqueta p2?

```
<p2>
     <li>Marta</li>
     <li>Sara</li>
     <li>Ana</li>
</p2>
```

a) ol.

En HTML, es la marca que sirve para agrupar varios elementos dentro de un bloque y permite asignarle un identificador.

a) .

En el lenguaje HTML, ¿mediante qué comando se define un formulario?

a)<form> </form>.

¿Qué nombre pondremos en la etiqueta <XXX> del siguiente HTML?

```
<!DOCTYPE html>
<html>
   <body>
      <h2>Introduce un titulo</h2>
         <table>
            <XXX>
               <YYY>Adios</YYY>
            </XXX>
         </table>
   </body>
</html>
```

a)<tr> </tr>.

Señala que etiqueta podría faltar en lugar de <YYY>.

```
<!DOCTYPE html>
<html>
   <body>
      <h2>Introduce un titulo</h2>
         <table>
            <XXX>
               <YYY>Adios</YYY>
            </XXX>
         </table>
   </body>
</html>
```

b) <td></td>.

2.7. Herramientas de diseño web

Indica cuantas formas existen de añadir hojas de estilo a los documentos HTML.

d) Todas las opciones son correctas.

Indica otra forma de representar el siguiente contenido CSS.

```
h1{Font-family: Verdana;}
h1{color:red;}
```

a) h1{Font-family: Verdana; color: red;}.

Los atributos tienen distintas formas de seleccionar elementos. ¿Cómo lo representaremos si queremos elementos que tienen el mismo atributo con el valor que se pasa?

a) [atributo=valor].

En las hojas de estilo en cascada, indica cuál/es de las siguientes formas son válidas para incorporar color al contenido de la etiqueta <p>.

d) Todas las opciones son correctas.

3.2. Asociación con documentos XML

Primero de todo, es necesario saber que el objetivo de un DTD es especificar aquellos elementos que deben aparecer, junto con el orden que deben seguir. ¿A qué se refiere la siguiente definición: es una declaración de un tipo de elemento que nos señala si existe un elemento a un determinado documento XML?

a) <!ELEMENT>.

¿Cuál de las siguientes opciones es un atributo que podemos diferenciar en un DTD?

e) Todas las opciones son correctas.

¿Cómo se representa en el documento XML teniendo la siguiente declaración del atributo en el DTD?

```
<!ATTLIST hora zona CDATA "GMT+1"
#REQUIRED>
```

a) <hora zona="GTM+1">13:00</hora>.

Indica cuál de las siguientes opciones es un componente que utilizamos para declarar un elemento y comprobar si existe en el documento XML.

a) xs:element.

solucionario

¿Cuál de las siguientes afirmaciones sobre los tipos de datos complejos es correcta?

d) Todas las opciones son correctas.

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

Es un lenguaje derivado del XML y tiene como función principal compartir o distribuir la información que se encuentra en la web.

a) RSS.

4.4. Tecnologías de creación de canales de contenidos

Indica cuál de las siguientes opciones son elementos se necesita en la etiqueta <channel> para describir la fuente RSS.

a) <title>, <link> y <description>.

Dentro del elemento <channel>, ¿qué etiqueta permite información inmediata de los cambios de canal?

a) <cloud>.

¿Cómo se llama el elemento que cuenta con la posibilidad de poder definir un determinado artículo en el canal RSS?

a) <item>.

4.7. Agregación

Dentro del elemento <item>, ¿qué etiqueta permite incluir un archivo multimedia?

a) <enclosure>.

Una vez creado el archivo RSS, ¿mediante qué herramienta podemos validarlo?

a) W3C.

¿Cuál es el lector RSS que se puede instalar en el ordenador de un usuario?

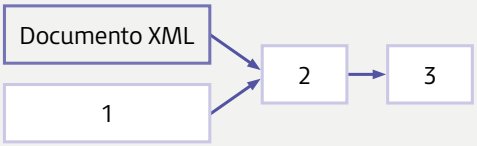
a) Agregador de escritorio.

5.2. Descripción y utilización de la estructura, sintaxis y plantillas

¿Cuál es el lenguaje que puede transformar aquellos documentos XML a diferentes formatos?

a) XSLT.

Completa el esquema de procesamiento XSLT.



a) 1: Hoja de transformaciones XSLT – 2: Procesador XSLT – 3: Documento transformado.

5.4. SXL-FO

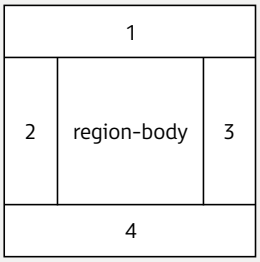
Dentro de los elementos básicos de una hoja de transformaciones, es el elemento encargado de definir el formato del documento de salida. Este elemento corresponde a un nivel superior y tiene que aparecer como "hijo de". ¿De qué elementos estamos hablando?

a) xsl: output.

Dentro de las instrucciones de control, permite repetir una lista de elementos para poder realizar diferentes transformaciones sobre ellos. ¿De qué estamos hablando?

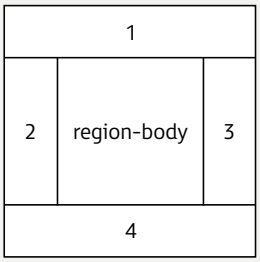
a) xsl: for- each.

En referencia a los objetos de formateo XSL-FO, coloca la región correspondiente en el número 4.



b) region-after.

En referencia a los objetos de formateo XSL-FO, ¿cuál es la región correspondiente en el número 1?



a) region-before.

solucionario

6.3. Técnicas de búsqueda de información en 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. ¿Qué es Xpath?

d) Todas las opciones anteriores son correctas.

¿Cuál de las siguientes afirmaciones son ciertas en XQuery?

e) Todas las opciones anteriores son correctas.

¿Qué significa el siguiente código de expresión en lenguaje XPath: e1/e2?

a) Elemento e2 es hijo directo de su padre e1.

¿Qué significa el siguiente código de expresión en lenguaje XPath: @atrib?

a) Atributo que se identifica con el nombre atrib.

7.1. Inteligencia del negocio

¿Qué son los sistemas ERP?

d) Aplicaciones integrales y modulares.

¿Cómo se definen las aplicaciones de inteligencia de negocio?

b) Aplicaciones que consiguen, entre otras cosas, automatizar y agilizar procesos, registrar resultados y favorecer la extracción de conclusiones.

7.4. Instalación de un sistema de gestión empresarial

La gestión empresarial es la actividad que tiene como objetivo principal aumentar la productividad y competitividad de una empresa.

a) Verdadero.

¿Cuál de los siguientes programas es ejemplo de EPR?

d) Todas las opciones son correctas.

¿Cuál de las siguientes opciones es una característica/as de la planificación de recursos empresariales (ERP)?

e) Todas las opciones son correctas.

Indica cuál de las siguientes opciones es una desventaja de los ERP.

a) La instalación y puesta en marcha de un sistema ERP es una inversión que cuesta dinero y tiempo.

Indica los módulos que podemos encontrar en un ERP.

f) Todas las opciones son correctas.

¿Cuáles son las principales tareas de la gestión de relaciones con el cliente (CRM)?

e) Todas las opciones son correctas.