

Diseño e integración web

FORMADORES { IT }

info@formadoresfreelance.es · www.formadoresit.es



Ayuntamiento de
FUENLABRADA

Índice

Introducción	3
Principios de diseño de páginas web	3
Compatibilidad.....	3
Utilidad	4
Interoperabilidad	5
Acceso universal	6
Pregunta de evaluación.....	9
Accesibilidad y Usabilidad en páginas web	6
Accesibilidad.....	6
Usabilidad.....	8
Pregunta de evaluación.....	14
Herramientas para trabajar con HTML5 y CSS3	10
Creación de una página en HTML5 con Eclipse.....	12
Herramientas de edición de Eclipse.....	13
Pregunta de evaluación.....	20
Introducción a HTML	14
Marcado HTML.....	15
Pregunta de evaluación.....	23
Etiquetas principales del estándar HTML	17
Comentarios.....	17
Estructura principal del documento	18
Etiquetas de la cabecera.....	18
Etiquetas del cuerpo.....	20
Pregunta de evaluación.....	38
Práctica	38
Formularios	31
Características.....	31
Creación de formularios con HTML.....	32
Elementos de un formulario.....	33
Algunos tipos de elemento <i>input</i>	34

Pregunta de evaluación.....49

Práctica49

Etiquetas incorporadas a HTML542

Etiquetas semánticas.....42

Etiquetas en formularios.....44

Pregunta de evaluación.....58

Práctica58

Componentes multimedia HTML5.....47

Compatibilidad entre navegadores52

Consulta tablas de compatibilidad52

Usar librerías53

Pregunta de evaluación.....66

Creación de hojas de estilo.....54

Tipos de hojas de estilo.....54

Como se define un estilo56

Selectores.....56

Pseudo-clases CSS359

Estudio de las propiedades de estilo CSS360

Pregunta de evaluación.....80

Práctica80

Posicionamiento SEO.....68

Correcta redacción de los contenidos69

Metatags70

Enlaces hacia nuestra web.....70

Usar el atributo *alt* de las imágenes70

Herramientas71

Pregunta de evaluación.....83

Trabajar en equipo71

Arquitecturas de almacenamiento.....72

Pregunta de evaluación.....87

Construcción de una web completa75

Introducción

En este capítulo, nos centraremos en el diseño y desarrollo de las páginas web.

Veremos algunos principios de diseño, es decir, como debemos plantear una página web, como queremos presentarla y como queremos que el usuario interactúe con ella.

También explicaremos como desarrollar la página que hemos diseñado usando los últimos estándares como son HTML5 y CSS3 teniendo en cuenta la compatibilidad entre los distintos navegadores existentes en el mercado.

Otro aspecto importante es el posicionamiento de nuestra página en los buscadores de Internet. Se propondrán algunas técnicas que mejorarán la forma en la que encontramos nuestra página en un buscador.

Principios de diseño de páginas web

Antes de comenzar en materia de desarrollo propiamente dicho, vamos a ver algunos conceptos de diseño que debemos conocer previamente. Son directivas o consejos de como debemos enfocar nuestro desarrollo de una web.

Compatibilidad

- Dar soporte al contenido existente: los navegadores deben ser compatibles con cualquier versión de HTML, CSS o JavaScript. No todo el contenido que encontramos en la Web está desarrollado con las últimas tecnologías.
- Degradar de forma elegante (degrade gracefully): cuando diseñamos páginas web con las últimas tecnologías y estándares, debemos tener en cuenta como se van a comportar en navegadores antiguos y dar una alternativa de visualización adecuada para estos dispositivos.
- No reinventar la rueda: se debe fomentar la reutilización de elementos. Si existe algo que existe y funciona correctamente, úsalo. Si desarrollas algo nuevo, hazlo de manera que pueda ser reutilizada en el futuro.

- Construyendo sobre lo que ya existe: si ya hay cierta práctica extendida, es preferible adoptarla y mejorarla que prohibirla o sustituirla por algo radicalmente diferente.
- Evolución, no revolución: es preferible evolucionar progresivamente un diseño cambiando cosas sobre lo ya construido, que cambiar un diseño por otro.

Utilidad

- Resolución de problemas reales: cualquier modificación o ampliación de un diseño debe responder a una necesidad real. No se deben añadir funcionalidades innecesarias ya que podemos construir páginas o aplicaciones web con una excesiva complejidad de uso.
- Prioridad de los usuarios en caso de conflicto: en relación con el diseño de una especificación como la de HTML, CSS o JavaScript, encontramos a varios grupos de personas: los usuarios de la web, los autores del contenido, los desarrolladores de los navegadores, los redactores de las especificaciones y, por último, los puramente teóricos.

Si existiese un conflicto de intereses entre alguno de estos grupos en relación a una funcionalidad concreta, este se debería resolver según el orden anteriormente citado.

Por ejemplo, si hubiese un conflicto en relación a la representación visual de una tabla entre usuarios de la web y los desarrolladores de los navegadores web, el conflicto se debería resolver en favor de los usuarios.

- Seguro por diseño: la seguridad en una página web, debe ser tomada en consideración desde el propio diseño. Por ejemplo: accesos seguros mediante una página de login (mediante HTTPS), control de acceso de usuarios, roles, permisos, etc.

No solo debemos dar soluciones de seguridad con herramientas de implementación.

- Separación de responsabilidades: HTML nos permite crear páginas web con un contenido y presentarlo al usuario de una forma atractiva al usuario. Sin embargo, debemos tener en cuenta que separar el contenido de la presentación es una buena estrategia a seguir. Lo podemos separar mediante los siguientes elementos:

- HTML: documento en el que escribimos el contenido de una página.
- CSS: estilos que podemos aplicar sobre un documento HTML para definir cómo se va a visualizar su contenido en un navegador.

De esta forma, podemos representar un mismo contenido de formas diferentes aplicando distintas hojas de estilo CSS.

También podemos reutilizar estilos aplicados a un documento HTML a cualquier otro documento HTML que necesitemos.

- Consistencia del árbol DOM: el árbol dom (Document Object Model) es una estructura de datos en la que el navegador almacena la información que lee de una página HTML.
- Esta estructura de datos debe ser implementada de igual forma entre los distintos navegadores. Solo de esta forma se garantiza que un script contenido en una página HTML puede obtener la misma respuesta en todas las plataformas cuando accede al DOM.

Interoperabilidad

- Bien definida por comportamiento: se debe definir con precisión el funcionamiento de cada requisito de la especificación.
- Evitar complejidad innecesaria: de todas las soluciones correctas a un problema, la solución que se debe escoger en todos los casos es la más sencilla. De esta forma, será más sencillo realizar desarrollos evolutivos por nuestra parte, o por la de cualquier otro desarrollador.
- Control de errores: definir un correcto control de errores es un requisito importante para desarrollar aplicaciones interoperables.
En ningún caso, la aplicación no debe parar bruscamente su ejecución.
El usuario debe recibir notificaciones elegantes y con un contenido fácilmente interpretable por él, pero nunca detalles técnicos del error.

Acceso universal

- Independencia del medio: las páginas deben poder ser aceptadas por cualquier navegador, independientemente de la plataforma o dispositivo en el que van a ser visualizadas.
- Apoyo a diferentes idiomas: en un diseño, debemos tener en cuenta a qué público va destinado y tener en cuenta el idioma en el que se relacionan dichos grupos. Tenemos que desarrollar permitiendo que el usuario al que va dirigido el contenido pueda verlo en su idioma.
- Accesibilidad: se deben diseñar funcionalidades teniendo en cuenta que pueden ser utilizadas con personas con discapacidades. Por ejemplo, permitir estilos de alto contraste añadir textos alternativos para las imágenes (por ejemplo, si usamos un lector de páginas), ser cuidadoso en la selección de los colores (hay un alto porcentaje de personas con daltonismo).

Accesibilidad y Usabilidad en páginas web

Son dos conceptos que, en ocasiones se confunden o se mezclan entre sí. A pesar de que pueden parecer que ambas cosas van dirigidas en el mismo sentido (métodos y técnicas para diseñar correctamente nuestra página web), en muchos casos entran en conflicto.

A continuación, vemos en detalle cada uno de los conceptos.

Accesibilidad

Cuando los sitios web están diseñados pensando en la accesibilidad, todos los usuarios pueden acceder en condiciones de igualdad a los contenidos. Por ejemplo, cuando un sitio tiene un código XHTML semánticamente correcto, se proporciona un texto equivalente alternativo a las imágenes y a los enlaces se les da un nombre significativo, esto permite a los usuarios ciegos utilizar lectores de pantalla o líneas Braille para acceder a los contenidos. Cuando los vídeos disponen de subtítulos, los usuarios con dificultades auditivas podrán entenderlos plenamente. Si los contenidos están escritos en un lenguaje sencillo e ilustrados con diagramas y animaciones, los usuarios con dislexia o problemas de aprendizaje están en mejores condiciones de entenderlos.

Si el tamaño del texto es lo suficientemente grande, los usuarios con problemas visuales puedan leerlo sin dificultad. De igual modo, el tamaño de los botones o las áreas activas adecuado puede facilitar su uso a los usuarios que no pueden controlar el ratón con precisión. Si se evitan las acciones que dependan de un dispositivo concreto (pulsar una tecla, hacer clic con el ratón) el usuario podrá escoger el dispositivo que más le convenga.

Lo mencionado en los párrafos anteriores se puede resumir en **pautas de accesibilidad**: estas pautas explican cómo hacer accesibles los contenidos de la web a personas con discapacidad. Las pautas están pensadas para todos los diseñadores de contenidos de la web y para los diseñadores de herramientas de creación. El fin principal de estas pautas es promover la accesibilidad.

Estas pautas son una especificación del W3C que proporciona una guía sobre la accesibilidad de los sitios de la web para las personas con discapacidad. Han sido desarrolladas por la Iniciativa de Accesibilidad en la Web (WAI) del W3C.

Hacer un sitio Web accesible puede ser algo sencillo o complejo, depende de muchos factores como, por ejemplo, el tipo de contenido, el tamaño y la complejidad del sitio, así como de las herramientas de desarrollo y el entorno.

Muchas de las características accesibles de un sitio se implementan de forma sencilla si se planean desde el principio del desarrollo del sitio Web o al comienzo de su rediseño. La modificación de sitios Web inaccesible puede requerir un gran esfuerzo, sobre todo aquellos que no se "etiquetaron" correctamente con etiquetas estándares de XHTML, y sitios con cierto tipo de contenido, como multimedia.

Cuando se desarrolla o rediseña un sitio Web, la evaluación de la accesibilidad de forma temprana y a lo largo del desarrollo permite encontrar al principio problemas de accesibilidad, cuando es más fácil resolverlos. Técnicas sencillas, como es cambiar la configuración en un buscador, pueden determinar si una página Web cumple algunas de las pautas de accesibilidad. Una evaluación exhaustiva, para determinar el cumplimiento de las pautas, es mucho más compleja.

Hay herramientas de evaluación que ayudan a realizar evaluaciones de accesibilidad. No obstante, ninguna herramienta en sí misma puede determinar si un sitio cumple o no las pautas de accesibilidad. Para determinar si un sitio Web es accesible, es **necesaria la evaluación humana**.

Usabilidad

Son técnicas que ayudan a los seres humanos a realizar tareas en entornos gráficos de ordenador.

3 conceptos de gran importancia:

- Las aplicaciones son creadas para seres humanos que quieren realizar una tarea de una forma simple y eficaz utilizando para ello un ordenador y una web.
- La usabilidad permite que la tarea se realice de una forma sencilla y siguiendo unos pasos adaptados al comportamiento humano.
- Al hacer referencia al término "**tarea**", estamos haciendo referencia a cualquier acción que el usuario puede realizar en nuestra página o aplicación web.

El diseño de sitios web deben seguir los siguientes principios:

1. Anticipación: el sitio web debe anticiparse a las necesidades del usuario.
2. Autonomía: los usuarios deben tener el control sobre el sitio web. Se deben sentir que conocen el entorno y no perderse en un sitio web de tamaño excesivamente grande para poder navegar sin necesidad de ayuda o indicaciones.
3. Los colores: han de utilizarse con precaución para no dificultar el acceso a los usuarios con problemas de distinción de colores.
4. Consistencia: las aplicaciones deben ser acordes a lo que el usuario espera de ellas sin encontrarse elementos desconcertantes (principio de mínima sorpresa).
5. Eficiencia del usuario: los sitios web se deben centrar en la productividad del usuario, no del propio sitio web. Por ejemplo, en ocasiones tareas con mayor número de pasos son más rápidas de realizar para una persona que otras tareas con menos pasos, pero más complejas.
6. Reversibilidad: un sitio web ha de permitir deshacer las acciones realizadas.
7. La ley de Fitts: indica que el tiempo para alcanzar un objetivo con el ratón está en función de la distancia y el tamaño del objetivo. Las acciones más

habituales para un usuario deben estar lo más cerca posible de la posición previa del cursor y la superficie del elemento debe ser suficientemente grande.

8. Reducción del tiempo de latencia: se debe optimizar el tiempo disponible en la medida de lo posible permitiendo al usuario realizar otras tareas mientras se completa la tarea en curso.
9. Aprendizaje: los sitios web deben requerir un mínimo proceso de aprendizaje y deben poder ser utilizados desde el primer momento.
10. El uso adecuado de metáforas: puede facilitar el aprendizaje de un sitio web, pero su uso inadecuado de estas puede dificultarlo.
11. Protección del trabajo de los usuarios: se debe asegurar que los usuarios nunca pierden su trabajo como consecuencia de un error.
12. Legibilidad: el tamaño de fuente debe ser suficientemente grande. El color de los textos debe resaltar con respecto al fondo para que sea visible. Por ejemplo, no es muy adecuado añadir texto blanco sobre una imagen muy clara ya que esto dificulta su lectura.
13. Seguimiento de las acciones del usuario: conociendo y almacenando información sobre su comportamiento previo se ha de permitir al usuario realizar operaciones frecuentes de manera más rápida.
14. Interfaz visible: es preferible evitar el uso de elementos ocultos de navegación que deben ser manipulados por el usuario. Por ejemplo, menús desplegables, indicaciones ocultas, etc.



Ilustración 1: principios de usabilidad

Herramientas para trabajar con HTML5 y CSS3

Para trabajar con HTML5 vamos a usar el entorno Eclipse.

Se trata de un entorno de desarrollo multiplataforma y permite desarrollar en múltiples lenguajes tanto de programación como de marcado, uso de UML, etc.

El IDE eclipse es de libre distribución y puede descargarse desde su web oficial (www.eclipse.org), más concretamente la versión para desarrolladores JEE. Esta versión contiene una serie de plugins pre-instalados para desarrollo de aplicaciones web con Java. Aunque el lenguaje Java se encuentra fuera del ámbito de este curso, esta versión también contiene el editor para HTML5, que es la funcionalidad que vamos a usar.

En la web oficial de Eclipse, se define como "An IDE for everything and nothing in particular" (un IDE para todo y para nada en particular). Eclipse es, en el fondo, únicamente un almacén (workbench) sobre el que se pueden montar

herramientas de desarrollo para cualquier lenguaje, mediante la implementación de los plugins adecuados. La arquitectura de plugins de Eclipse permite, además de integrar diversos lenguajes sobre un mismo IDE, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, etc.

El IDE Eclipse es, únicamente, una de las herramientas que se engloban bajo el denominado *Proyecto Eclipse*. El *Proyecto Eclipse* aúna tanto el desarrollo del IDE Eclipse como de algunos de los plugins más importantes (como el JDT, plugin para el lenguaje Java, o el CDT, plugin para el lenguaje C/C++).

Este proyecto también alcanza a las librerías que sirven como base para la construcción del IDE Eclipse (pero pueden ser utilizadas de forma completamente independiente), como por ejemplo, la librería de widgets SWT. A partir de la versión Helios del IDE de Eclipse y superiores (con los plugins para desarrolladores de Java EE) proporciona soporte para el desarrollo de HTML5. Las versiones anteriores también tienen un editor de HTML, pero no admite HTML 5.

En concreto, permite:

- Crear páginas HTML 5 con una nueva plantilla.
- Proporciona autocompletado de código para los elementos de HTML5.
- Ofrece un editor de propiedades para los atributos de HTML5.
- Ofrece un editor WYSIWYG para el desarrollo visual de páginas HTML5.

Creación de una página en HTML5 con Eclipse

Crear un documento HTML 5 es fácil. Simplemente hay que estos pasos:

- 1. Seleccione File → New → Other.
- 2. Expandir Web y seleccionar archivo HTML. Pulsar 'Next'.
- 3. Escribir un nombre de archivo, seleccionar la carpeta principal y pulsar 'Next'.

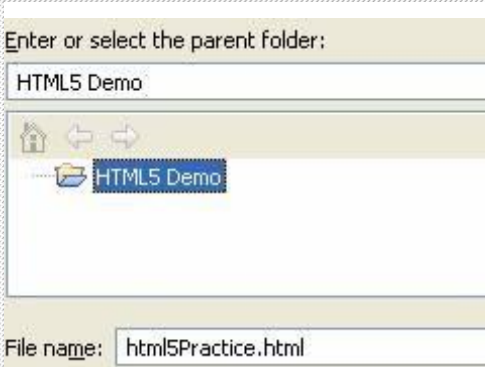


Ilustración 2: nuevo documento html5 (1)

- 4. Seleccione nuevo archivo HTML5 para la plantilla y pulsar 'Finish' para terminar.

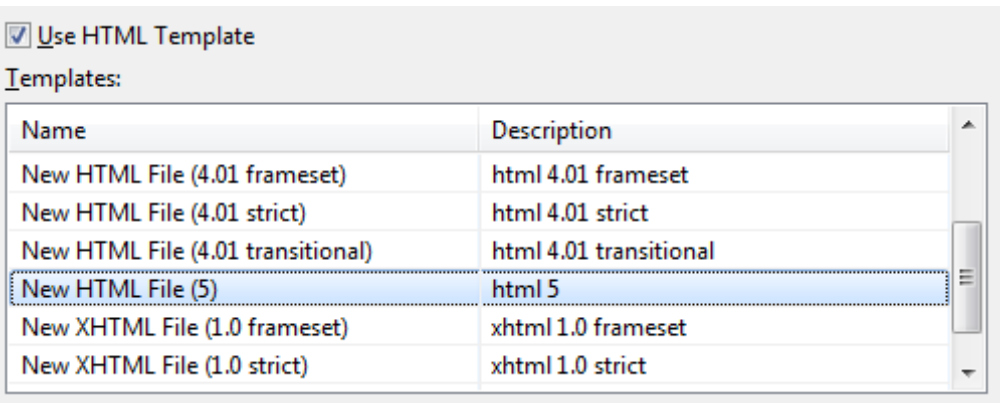


Ilustración 3: nuevo documento html5 (2)

El archivo se abrirá con el editor de HTML, que proporciona resaltado de sintaxis (añade diferentes colores y tipos de letra en función del elemento que se muestra). Las etiquetas se muestran en verde, los atributos de color morado, los valores de los atributos de color azul y los valores de los elementos en negro.

Hay que tener en cuenta que el documento contiene el elemento **DOCTYPE** **que se utiliza**. También contiene el elemento meta simplificado que se utiliza para especificar la codificación de caracteres UTF-8.

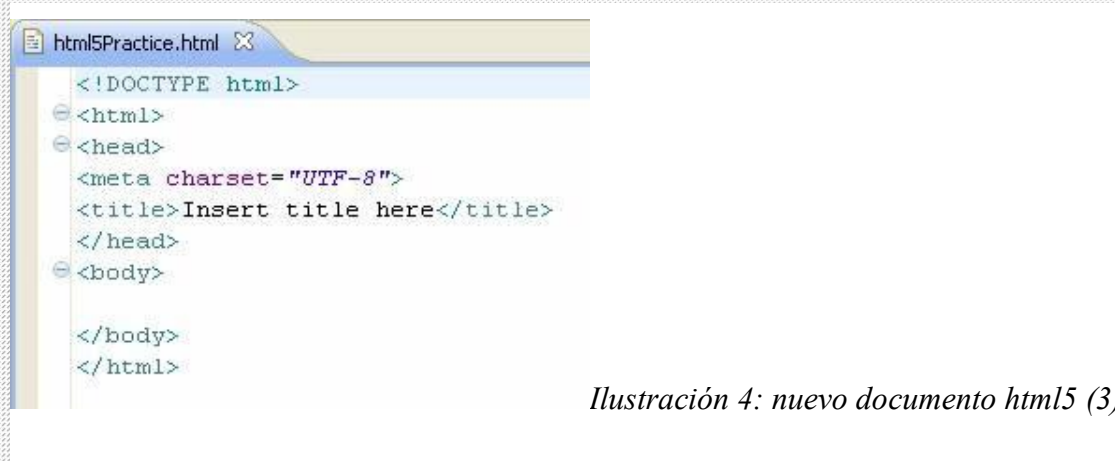
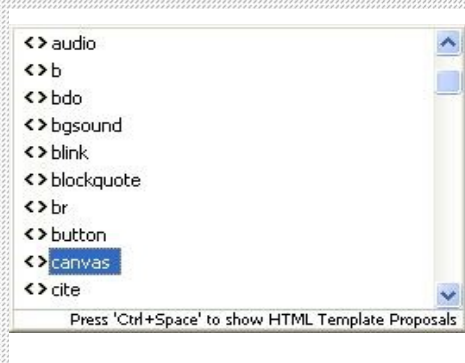


Ilustración 4: nuevo documento html5 (3)

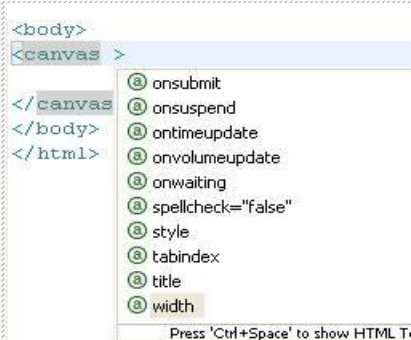
Herramientas de edición de Eclipse

Vamos a ver algunas de las herramientas que el plugin nos proporciona.

- La primera y más evidente herramienta que nos ofrece el plugin ya lo hemos comentado. Es el resaltado de la sintaxis. Las palabras reservadas y signos correspondientes al estándar aparecen resaltados en color. El resto del texto en negro.
- Autocompletado de etiquetas. Basta con pulsar ctrl-espacio y una lista de elementos HTML5 válidos aparecerán listados. Si ya hemos escrito parte de la etiqueta, nos lo autocompletará si solo hay una etiqueta que empieza por el texto escrito. Si hay más de una, aparecerá un listado únicamente con las coincidencias. Solo muestra las etiquetas permitidas en el lugar en el que estamos editando.

*Ilustración 5: autocompletado de etiquetas*

- Autocompletado de atributos: funciona exactamente igual que con las etiquetas, solo que debemos estar editando dentro de la declaración de una etiqueta.

*Ilustración 6: autocompletado de atributos*

Introducción a HTML

HTML (HiperText Markup Language). Su nombre se traduce literalmente como “Lenguaje de marcado de hipertexto”.

Se encuentra enmarcado en los llamados lenguajes de marcado. Los lenguajes de marcado son los utilizados para generar documentos en los que, junto con el texto que constituye el contenido, se incluyen etiquetas o marcas que contienen información adicional sobre la estructura del contenido.

En el caso de HTML, se trata concretamente de un lenguaje de marcado de presentación, en el que las etiquetas que acompañan al texto indican al navegador información de como se encuentra organizado.

Es un estándar desarrollado y regulado por la W3C, que es un organismo dedicado a la estandarización de las tecnologías utilizadas en la Web.

Una de las características principales del lenguaje es que las páginas generadas con HTML son de hipertexto. Esto quiere decir que una página no contiene un texto aislado, si no que están o pueden estar enlazadas a otras páginas y ser referenciadas desde otras páginas mediante enlaces o links. Esto nos permite navegar por ellas.

Es independiente de la plataforma ya que HTML solo genera documentos de texto y admite cualquier codificación de caracteres.

El proceso de alterar la información de una página es relativamente sencillo y ágil ya que es un lenguaje de fácil comprensión.

Basa su contenido en referenciación. Si en una página HTML queremos introducir algún elemento como una imagen, script, video, u otra página HTML (por ejemplo), no se incluye su contenido directamente en ella si no que se referencia (se indica donde está el contenido que queremos incluir). El navegador es el encargado de interpretar las referencias y cargarlas componiendo la visualización final.

Al ser un estándar, uno de sus objetivos es la compatibilidad hacia atrás (o retro-compatibilidad). Un navegador que es capaz de interpretar una versión de HTML, debe ser capaz de interpretar páginas escritas en cualquiera de sus versiones anteriores.

Marcado HTML

El marcado HTML se realiza mediante etiquetas de apertura y cierre. Todo lo que se encuentra dentro de una etiqueta de apertura y cierre se encuentra afectado por ella, incluyendo texto plano y cualquier otra etiqueta que pueda contener.

- Una etiqueta de apertura se define con un nombre de etiqueta encerrado entre los corchetes angulares < y > (signo "menor que" y signo "mayor que")
Ejemplo: <body>
- Una etiqueta de cierre se define con el corchete angular de apertura < seguido del carácter barra /, el nombre de etiqueta y finalmente el corchete angular de cierre.
Ejemplo: </body>

- Entre la etiqueta de apertura y la de cierre, se encuentra el contenido. El contenido puede ser texto u otras etiquetas. Una etiqueta con contenido tiene el siguiente formato:
`<etiqueta>contenido</etiqueta>`
- Si entre la etiqueta de apertura y la de cierre no se añade ningún contenido, se puede declarar con un corchete angular de apertura `<` seguido del nombre de la etiqueta, una barra `/` y finalmente el corchete angular de cierre.
Ejemplo: `
`
- Los atributos añaden información adicional a una etiqueta. Cada etiqueta admite una serie de atributos. Se añaden únicamente a las etiquetas de apertura, justo después del nombre. Se admite más de un atributo por etiqueta. Siguen el esquema clave / valor de la siguiente forma: *clave="valor"*.
Ejemplo: ``

Por ejemplo, supongamos que tenemos el siguiente código HTML:

```
<a href="http://httpd.apache.org">enlace a la página de apache</a>
```

El texto que se encuentra encerrado entre la etiqueta de apertura `<a>` y la etiqueta de cierre ``, será interpretado por el navegador como un enlace. El atributo `href` indica la página a la que referencia el enlace.

En un fichero que va a contener un documento HTML, debemos definir en la primera línea cual es la versión de HTML que vamos a utilizar. De esta forma, el navegador podrá interpretarlo correctamente. También es útil añadir el tipo de documento a la hora de desarrollar puesto que el editor de HTML que usemos, podrá validar si lo que escribimos es correcto.

A continuación, vemos dos ejemplos de página de versiones diferentes de HTML:


```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>login</title>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
</head>
<body>
```

Ilustración 7: Definición de documento en HTML 4.01

```
<!DOCTYPE html>
<html>
<head>
  <title>login</title>
  <meta charset="UTF-8">
</head>
<body>
```

Ilustración 8: Definición de documento en HTML 5

Etiquetas principales del estándar HTML

A continuación, veremos como podemos crear los elementos más comunes de una página web mediante etiquetas HTML:

Comentarios

Los comentarios son textos que no serán interpretados por el navegador. Únicamente se utilizan para añadir información útil para el desarrollador.

Los caracteres `<!--` se utilizan para la apertura del comentario y los caracteres `-->` se para el cierre.

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

Ilustración 9: comentario

Estructura principal del documento

```
<!DOCTYPE html>
<html>
<head>
  <title>login</title>
  <meta charset="UTF-8">
</head>
<body>

...

</body>
</html>
```

Texto 1: Estructura de página HTML

- **html:** etiqueta que define la raíz del documento. Todos los elementos de la página deben estar contenidos en la etiqueta HTML.
- **header:** contiene las cabeceras de la página HTML. Las cabeceras contienen información adicional para el navegador. Pueden ser, por ejemplo, el título de la página, enlaces a hojas de estilo, codificación de caracteres, etc.
- **body:** es el cuerpo de la página. Dentro de esta etiqueta se encuentra el contenido que vamos a mostrar al usuario.

Etiquetas de la cabecera

Ya hemos visto que es la cabecera de una página. Ahora veremos cuales son los principales elementos que podemos incluir en ella:

- **title:** título de la página. Se muestra en la barra de título del navegador (o en una pestaña si usamos un navegador con pestañas). Solo se puede añadir una vez por documento.



Ilustración 10: título de la página web

- meta (**cambia en HTML5**): añade información adicional a la página de naturaleza variada. Podemos indicar la codificación de caracteres del documento, información sobre el autor, palabras clave para los motores de búsqueda, etc.

Se pueden añadir los meta que sean necesarios pero sin repetirlos. Algunos ejemplos de meta que podemos añadir:

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

Texto 3: Codificación de caracteres con UTF-8

La forma en la que se especifica la codificación de caracteres **cambia en HTML 5**

```
<meta name="author" content="nombre del autor" />
```

Texto 2: Autor

```
<meta name="email" content="nombreautor@dominio.com" />
```

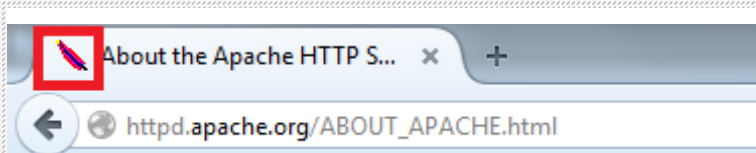
Texto 5: Email

- style: añade hojas de estilo explícitamente en el propio documento HTML.

```
<style type="text/css">
  h1.blue {
    color: blue;
  }
</style>
```

Texto 4: Estilos en documento HTML

- link (**cambia en HTML5**): referencia ficheros externos, por ejemplo, ficheros con hojas de estilo.
Es mejor práctica usar referencias a ficheros externos de hojas de estilo que definirlos en el propio documento HTML con la etiqueta *style* para separar el contenido de la presentación.
También podemos añadir el icono de favoritos.

*Ilustración 11: Favicon*

- script (**cambia en HTML5**): también se puede añadir dentro del cuerpo. Permite añadir un script en la propia página, o referenciar un fichero que contiene scripts y es específico para ello. Esta segunda forma, suele ser más adecuada puesto que el código del script está totalmente separado del documento HTML.

```
<script type="text/javascript">
  alert('hola mundo');
</script>
```

Texto 7: Script explícito en la página

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

Texto 6: Fichero de scripts referenciado

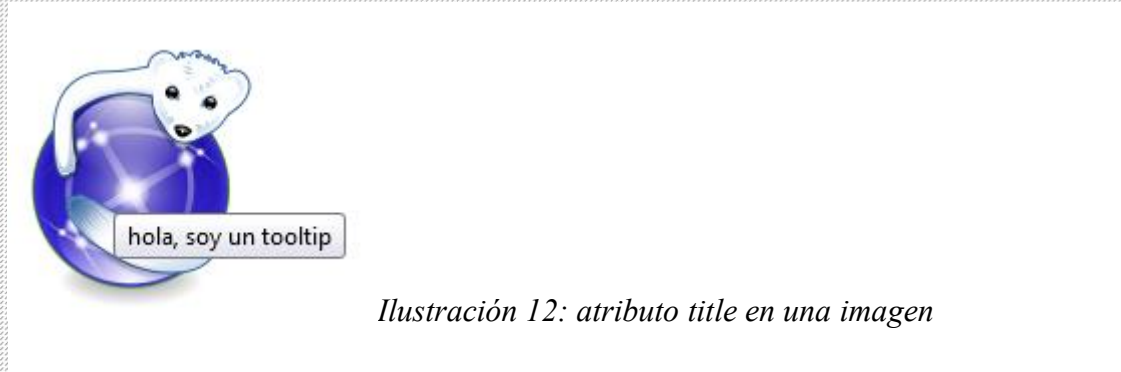
Etiquetas del cuerpo

El cuerpo y las etiquetas que incluye una página definen el contenido que se muestra al usuario.

Todas las etiquetas del cuerpo del documento (excepto *script*) pueden llevar los siguientes atributos:

- id: este atributo debe tener un valor único en toda la página, no deben existir dos etiquetas con el mismo valor en su atributo *id*.
El valor de este atributo, identifica unívocamente la etiqueta en el que está contenido dentro de la página.
Asignar un identificador a un elemento, nos puede servir, por ejemplo, para crear un enlace interno dentro de la propia página. En posteriores módulos veremos como se pueden acceder a dichos elementos para manipularlos mediante javascript.
- style: añade los estilos CSS especificados en el valor del atributo.
- class: añade un estilo CSS definido previamente en una hoja al elemento cuyo nombre se especifica en el valor del atributo.

- **title:** añade un texto explicativo especificado en el valor del atributo. Se muestra como un *tool tip* (globo informativo). La información se muestra al pasar el cursor sobre el elemento generado por la etiqueta.



- h[x] (título): define un título donde x es sustituido por un número del 1 al 6. h1 el título con mayor tamaño de letra y peso semántico y h6 el de menor tamaño y peso semántico.

Si no aplicamos hojas de estilo, se ve de la siguiente forma:

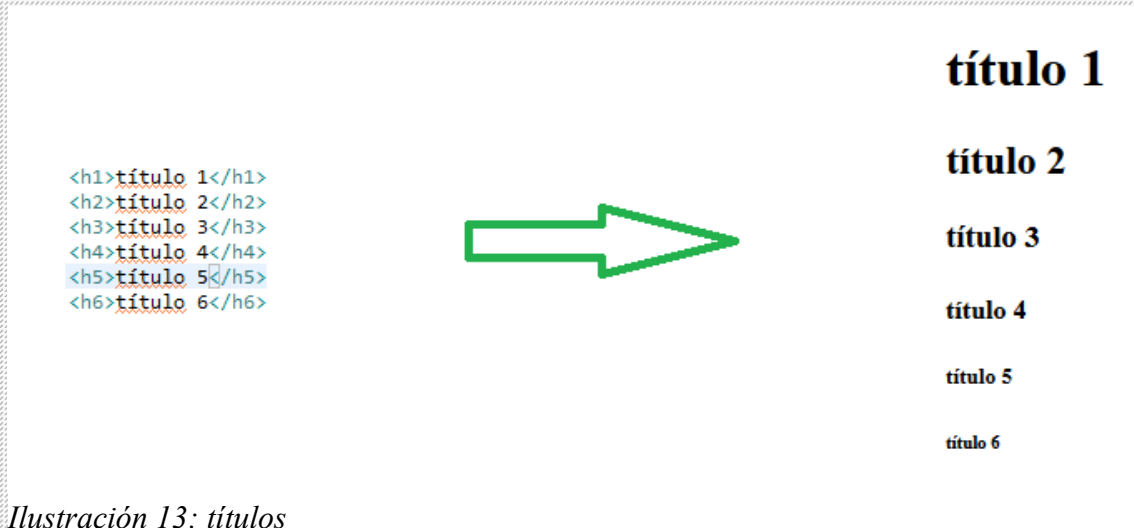


Ilustración 13: títulos

- p: define un párrafo. Entre cada párrafo añade un pequeño espaciado si no usamos un estilo adicional.

Si no aplicamos hojas de estilo, se ve de la siguiente forma:

```
<p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce auctor sollicitudin augue,
  nec sollicitudin urna euismod nec. Vivamus scelerisque ultricies justo, a finibus felis
  suscipit interdum.
</p>
<p>
  Etiam pulvinar ultricies lectus. Interdum et malesuada fames ac ante ipsum primis in faucibus.
</p>
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce auctor sollicitudin augue, nec sollicitudin urna euismod nec. Vivamus scelerisque ultricies justo, a finibus felis suscipit interdum.

Etiam pulvinar ultricies lectus. Interdum et malesuada fames ac ante ipsum primis in faucibus.

Ilustración 14: parrafo

Nótese que los saltos de línea existentes en el texto del documento HTML no se materializan como tal en el navegador. Tampoco cuando se usan múltiples espacios en blanco se ven reflejados como tal. Cualquier concatenación de tabulaciones, saltos de línea, y espacios en blanco o combinaciones de estos, se representan como un único espacio en blanco independientemente del número de ocurrencias existentes.

- pre: texto pre-formateado. El texto incluido como contenido de esta etiqueta se representa tal y como se escribe, manteniendo espacios en blanco, saltos de línea, tabulaciones, etc.

- `ol` / `li` (lista numerada / elemento lista): la etiqueta `ol` define una lista numerada, mientras que `li` contiene cada elemento de la lista. Si no aplicamos hojas de estilo, se ve de la siguiente forma:

```
<ol>
  <li>Fusce auctor sollicitudin augue, nec sollicitudin urna euismod nec</li>
  <li>Vivamus scelerisque ultricies justo, a finibus felis suscipit interdum</li>
</ol>
```



1. Fusce auctor sollicitudin augue, nec sollicitudin urna euismod nec
2. Vivamus scelerisque ultricies justo, a finibus felis suscipit interdum

Ilustración 15: lista numerada

- `ul` / `li` (lista no numerada / elemento lista): la etiqueta `ul` define una lista no numerada, mientras que `li` contiene cada elemento de la lista. Si no aplicamos hojas de estilo, se ve de la siguiente forma:

```
<ul>
  <li>Fusce auctor sollicitudin augue, nec sollicitudin urna euismod nec</li>
  <li>Vivamus scelerisque ultricies justo, a finibus felis suscipit interdum</li>
</ul>
```



- Fusce auctor sollicitudin augue, nec sollicitudin urna euismod nec
- Vivamus scelerisque ultricies justo, a finibus felis suscipit interdum

Ilustración 16: lista no numerada

- `table`, `caption`, `thead`, `tbody`, `tr`, `th`, `td` (tabla / cabecera tabla / cuerpo tabla / fila / celda cabecera / celda cuerpo): la etiqueta `table` define una tabla en HTML. Dentro de esta etiqueta, añadimos las etiquetas `thead` y `tbody`, que son la cabecera (títulos de la tabla) y el cuerpo de la tabla (datos contenidos en la tabla) respectivamente. Dentro de las etiquetas `thead` y `tbody`, podemos añadir filas mediante la etiqueta `tr`. En las filas (`tr`) de la cabecera, podemos añadir celdas de título mediante la etiqueta `th`. En las filas (`tr`) del cuerpo de la tabla, podemos añadir celdas de datos mediante la etiqueta `td`. El número de celdas debe coincidir en cada fila, independientemente de si son de la cabecera o del cuerpo. Si no es así, podemos hacer que una celda ocupe más de un hueco con el siguiente atributo de la etiqueta `td`:
 - ▲ `colspan`: define el número de huecos que va a ocupar una columna en una fila.

```

<table>
  <thead>
    <tr>
      <th>titulo 1</th>
      <th>titulo 2</th>
      <th>titulo 3</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>valor 1</td>
      <td>valor 2</td>
      <td>valor 3</td>
    </tr>
    <tr>
      <td colspan="2">valor 4</td>
      <td>valor 5</td>
    </tr>
  </tbody>
</table>

```

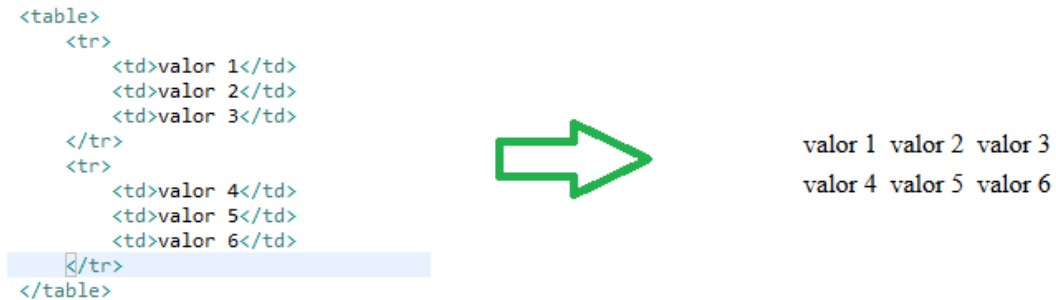


titulo 1	titulo 2	titulo 3
valor 1	valor 2	valor 3
valor 4	valor 5	

Ilustración 17: tabla completa con `colspan`

Si no aplicamos hojas de estilo, se ve de la siguiente forma:

También podemos omitir la cabecera de la tabla suprimiendo las etiquetas `thead` y `tbody`. Añadimos las filas directamente a la tabla de la siguiente forma:



```

<table>
  <tr>
    <td>valor 1</td>
    <td>valor 2</td>
    <td>valor 3</td>
  </tr>
  <tr>
    <td>valor 4</td>
    <td>valor 5</td>
    <td>valor 6</td>
  </tr>
</table>

```

valor 1 valor 2 valor 3
valor 4 valor 5 valor 6

Ilustración 18: tabla sin cabecera

Podemos añadir una leyenda a la tabla usando la etiqueta `caption` dentro de la tabla. El contenido de dicha etiqueta es el que se incluirá en la leyenda.

- a (enlace): nos permite crear un enlace a un fichero o página de nuestro sitio web, externo a nuestro sitio web o a elemento interno de la propia página que tenga un atributo *id* declarado.

El contenido de la etiqueta será convertido en un enlace.

Atributos principales:

- href: ruta del elemento referenciado.
Si el elemento es un fichero o página externa, su valor será una ruta (absoluta o relativa), aunque también puede ser una URL.
Si el elemento es un fichero o página externa, su valor será una URL.
Si el elemento es interno de la propia página (referencia a una etiqueta del propio documento), su valor tendrá el siguiente formato: *#id* donde *id* es el identificador de la etiqueta HTML que queremos enlazar.
- target: especifica donde queremos abrir el enlace. Algunos valores interesantes para este atributo son:
 - ⤴ *_self* (valor por defecto si no se especifica el atributo *target*): abre el enlace en la misma ventana.
 - ⤴ *_blank*: abre el enlace en una ventana o pestaña nueva.

Si no aplicamos hojas de estilo, se ve de la siguiente forma:

```
<a href="http://httpd.apache.org" target="_blank">enlace a apache</a>
```



[enlace a apache](http://httpd.apache.org)

Ilustración 19: enlace externo en ventana nueva


```
<a href="#titulo">volver al inicio del tema</a>
```



[volver al inicio del tema](#)

Ilustración 20: enlace

interno de la página

img (imagen): inserta una imagen en el documento.

Atributos:

- src: URL o ruta donde se encuentra la imagen.
- alt: texto alternativo. Debe ser un texto descriptivo de la imagen. Se utiliza en lectores para invidentes.
- height: altura de la imagen en píxeles.
- width: anchura de la imagen en píxeles.

Si no aplicamos hojas de estilo, se ve de la siguiente forma:

- div (divisor o capa): define una división o sección en un documento HTML agrupando elementos en un bloque (salto de línea antes y después del contenido). Su misión es permitir aplicar estilos sobre la sección o bloque completo.

Ejemplo:

```
<h1>insertamos una imagen:</h1>  

```



insertamos una imagen:



Ilustración 21: imagen

```
<div class="rojo"> esto es una capa</div>  
<a href="#titulo">volver al inicio</a>
```



esto es una capa
[volver al inicio](#)

Ilustración 22: capa

- span (divisor de elementos inline): agrupa los elementos que contiene, pero no produce ningún nivel visual. Simplemente crea un grupo de contenidos para poder aplicar estilos sobre el conjunto

completo. A diferencia de *div*, no crea un bloque, si no que puede ser aplicado, por ejemplo, en una línea de texto.

Ejemplo:

```
<p>dentro de este texto resaltamos lo que es <span class="rojo">importante</span></p>
```



dentro de este texto resaltamos lo que es importante

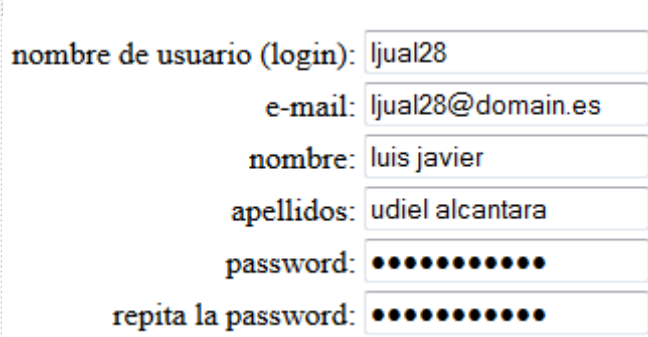
- br (salto de línea): crea un salto de línea.
- hr (línea o regla horizontal): elemento visual utilizado para separar secciones.
- applet / object (**desaprobado en HTML5**): etiqueta que permite insertar un objeto externo a HTML en una página web. Podríamos insertar un plugin de Adobe Flash o un Applet de Java.

Formularios

Los formularios HTML son elementos que permiten recoger datos del cliente desde una página web (en el lado del cliente) y enviarlos al servidor. Dentro del formulario, podemos introducir componentes HTML de distintas naturalezas para este fin. Por ejemplo, campos y cajas de texto, selectores, checkbox, radio, etc.

Desde un formulario podemos, incluso, mandar ficheros al servidor.

Usando formularios, tenemos un mecanismo para que el usuario pueda interactuar con nuestra aplicación.



nombre de usuario (login): ljual28

e-mail: ljual28@domain.es

nombre: luis javier

apellidos: udiel alcantara

password: ●●●●●●●●

repita la password: ●●●●●●●●

Ilustración 23: ejemplo de formulario

Un formulario también puede ser utilizado para mostrar datos al cliente.

Características

Un formulario permite enviar al servidor la información recogida por los diferentes elementos que lo componen.

Los datos se envían como una lista de parámetros. Cada parámetro se corresponde con un elemento del formulario.

La lista de parámetros se envía en formato clave / valor, donde la clave es el nombre del elemento del formulario al que corresponde y el valor es la información introducida o seleccionada por el usuario.

La información de un formulario se envía en una petición HTTP, por lo que debemos indicar en el formulario cual va a ser la URL del servidor que va a atender nuestra petición.

Una petición HTTP admite varios métodos o verbos como vimos en el módulo uno. Podemos indicar en el formulario si la información se va a enviar usando el método GET o POST.

Si no se especifica el método, por defecto se envía por GET.

- Método GET: la información recogida del formulario se envía en la propia URL de la petición con el siguiente formato:

url?param_1=valor_1¶m_2=valor_2&.....param_n=valor_n

Al enviarse en la propia URL, los parámetros serían visibles para el usuario.

- Método POST: los datos del formulario se envían dentro del cuerpo del mensaje HTTP.

valor_1¶m_2=valor_2&.....param_n=valor_n

Aunque los parámetros también son accesibles por el usuario, no se muestran directamente en el navegador. Es necesario utilizar alguna herramienta o plug-in del navegador para poder acceder a ellos.

Creación de formularios con HTML

La etiqueta *form* es el elemento principal en la creación de un formulario. Se encarga de agrupar los elementos que recogerán los valores que serán enviados en la petición HTTP.

Estos son algunos de sus atributos:

- **action** (obligatorio): URL que va a ser llamada cuando se envíen los datos del formulario.
- **method**: método HTTP utilizado en la llamada al servidor. Están permitidos los valores *get* y *post*.
- **target**: define donde se va a mostrar la respuesta del servidor. Puede tener los mismos valores que el atributo *target* de la etiqueta *a*: *_self*, *_blank*, *_parent*, etc.

Para añadir un formulario a una página web necesitamos conocer las etiquetas con las que definimos los campos de entrada de datos del formulario.

Los elementos de entrada de datos de un formulario pueden poseer los siguientes atributos comunes:

- **name**: este atributo identifica el elemento dentro del formulario. Pueden existir elementos con el mismo atributo *name* puesto que es relativo al formulario en el que están incluidos.
- **disabled**: el elemento aparece como desactivado. No es modificable por el usuario y visualmente, se presenta de diferente forma para que el usuario identifique claramente que el elemento está inactivo. Puede ser útil en casos en los que, dependiendo del perfil del usuario, queramos que no pueda introducir un valor, y queramos hacérselo saber expresamente. Un ejemplo puede ser un formulario en el que deben darse de alta los datos de una empresa. Si se sabe que el usuario es autónomo, no podemos permitirle seleccionar un tipo de empresa.

seleccione el tipo de empresa:

Ilustración 24: campo

desactivado

Elementos de un formulario

A continuación, veremos que etiquetas podemos incluir dentro de un formulario (etiqueta *form*).

- **input**: elemento genérico de entrada de datos. Se puede concretar con el atributo *type*.

Algunos atributos interesantes son:

- **type:** define el tipo de entrada de datos que queremos utilizar. Dependiendo del valor de este atributo, el componente interactuará de una manera concreta con el usuario y tendrá un aspecto u otro. Más adelante, detallaremos los tipos que podemos utilizar.
- **value:** valor por defecto del elemento.
- **readonly:** si aparece este atributo, el elemento es de solo lectura y no sería modificable por el usuario.

Algunos tipos de elemento *input*

Ya hemos visto que un elemento de tipo *input* nos permite añadir campos de entrada de datos a un formulario. Ahora veremos que tipos podemos incluir.

- **reset:** carga los valores por defecto de todos los campos del formulario.
Se presenta al usuario como un botón.
El valor de su atributo *value* define el texto que contiene el botón.

```
<input type="reset" value="cargar valores por defecto" />
```



cargar valores por defecto

Ilustración 25: *reset*

- **submit:** realiza la petición HTTP usando la URL definida en el atributo *action* del formulario, y añadiendo los parámetros correspondientes a los elementos contenidos en el formulario.
Se presenta al usuario como un botón.
El valor de su atributo *value* define el texto que contiene el botón.

- text: entrada de texto estándar. Solo admite una línea.

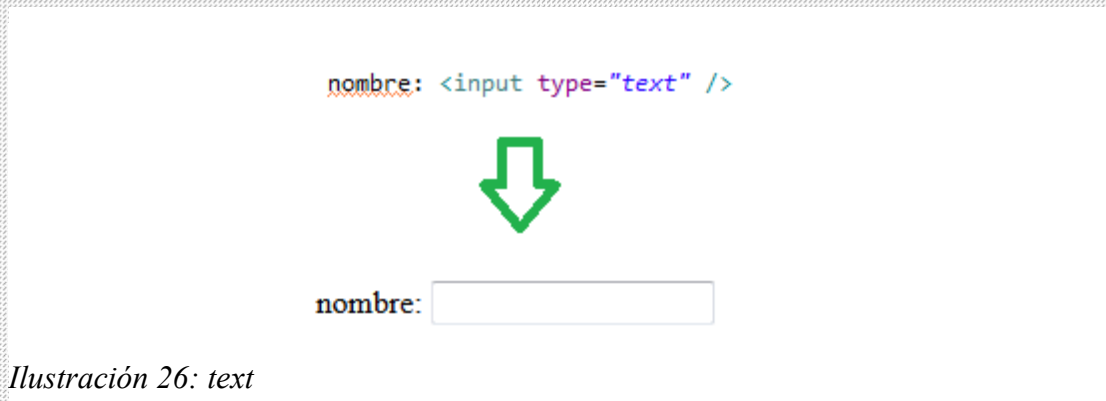


Ilustración 26: text

- password: es igual que un input de tipo text solo que cuando se escribe texto en el campo, se oculta con un asterisco o cualquier otro símbolo de ocultación.

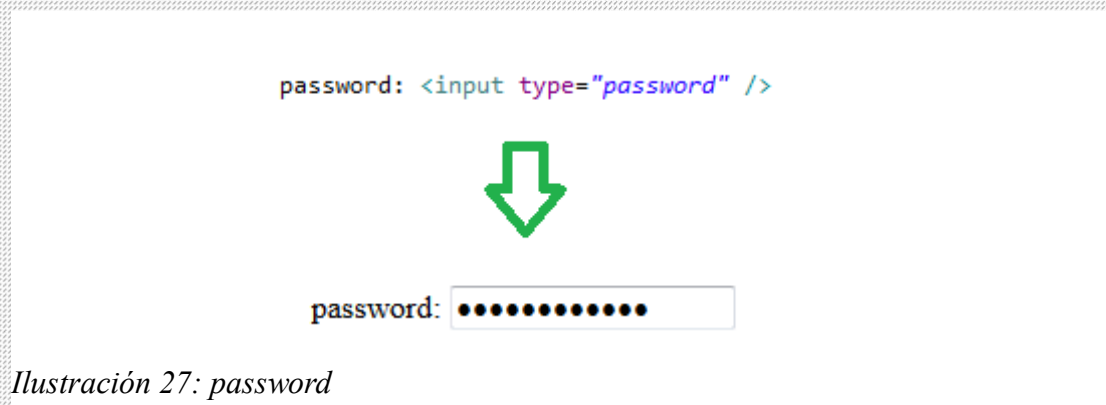


Ilustración 27: password

- hidden: campo invisible para el usuario. Se suele utilizar para enviar datos al servidor cuyo valor no debe ser conocido ni editado por el usuario.

- **radio:** componente que permite seleccionar una opción entre las presentadas al usuario. Tiene la misma función que el *select* simple (solo una opción) solo que siempre se muestran todas las opciones en el formulario.

Para que funcione correctamente, se debe añadir un *input* por cada opción seleccionable con un valor (atributo *value*) distinto, pero con el mismo nombre (atributo *name*).

Atributos propios del tipo *radio*:

- **checked:** si se presenta este atributo en alguno de los *input*, este será el que aparezca seleccionado por defecto.

```
<div>seleccione una de las siguientes opciones:</div>
<div><input type="radio" value="valor 1" /> opción 1</div>
<div><input type="radio" value="valor 2" checked /> opción 2</div>
<div><input type="radio" value="valor 3" /> opción 3</div>
```



seleccione una de las siguientes opciones:

- ☐ opción 1
- ☒ opción 2
- ☐ opción 3

Ilustración 28: radio

- checkbox: elemento para marcar y desmarcar. Si se desmarca, no se envía el atributo al cliente. Si se marca, se envía el parámetro, y como valor se añade el atributo *value*.

Para que aparezca marcado por defecto, se debe añadir el atributo *checked*.

```
<div><input type="checkbox" value="ok" /> acepto las condiciones</div>
```



☐ acepto las condiciones

Ilustración 29: checkbox

Textarea: caja de texto que admite múltiples líneas. Se utiliza para entradas de texto de gran tamaño y admite saltos de línea.

Si se añade contenido a la etiqueta, este aparecerá como valor por defecto de la misma.

Si no se indica lo contrario mediante estilos, el área de texto permitirá ser redimensionada por el usuario.

Atributos de *textarea*:

- rows: número de filas.
- cols: ancho del campo.
- readonly: al igual que en el elemento input, si aparece este atributo, el elemento es de solo lectura y no sería modificable por el usuario.

```
Lorem ipsum dolor sit amet, consectetur  
adipiscing elit. Fusce auctor  
sollicitudin augue, nec sollicitudin urna  
euismod nec. Vivamus scelerisque  
ultrices justo, a finibus felis suscipit  
interdum.
```

Ilustración 30: textarea

- **select / option:** elemento que permite elegir entre una lista de opciones. El elemento principal es la etiqueta *select* que contendrá la lista de posibles opciones.

Atributos de la etiqueta *select*:

- **multiple:** si aparece este atributo, el componente permite seleccionar más de un elemento de la lista. Manteniendo pulsada la tecla *ctrl* y haciendo click en cada elemento de los que queremos seleccionar.
- **Size:** número de elementos que son visibles simultáneamente. Por defecto es uno.

Como contenido de la etiqueta *select*, podemos tener una o más etiquetas *option*. Por cada etiqueta *option* que añadimos a *select*, se añade un elemento a la lista desplegable que se muestra al usuario.

El contenido de la etiqueta *option* es el texto que se mostrara al usuario.

Atributos de la etiqueta *option*:

- **value:** valor que se enviará al servidor si se selecciona l opción.
- **disabled:** no solo podemos desactivar el componente completo añadiendo este atributo a la etiqueta *select*, si no que también podemos desactivar algunas de las opciones concretas.
- **selected:** si se presenta este atributo en una opción, aparecerá como seleccionada. Si no, se selecciona por defecto la primera opción.


```
seleccione una divisa:  
<select>  
  <option value="dolar">$</option>  
  <option value="libra">£</option>  
  <option value="euro">€</option>  
  <option value="yen" disabled>¥</option>  
</select>
```

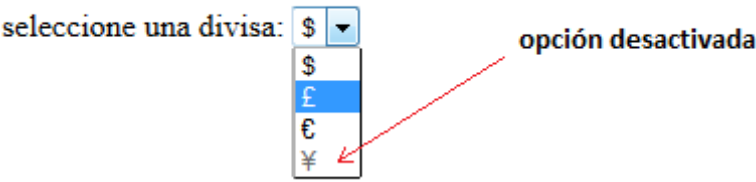


Ilustración 32: select

```
seleccione las posibles divisas:  
<select multiple>  
  <option value="dolar">$</option>  
  <option value="libra">£</option>  
  <option value="euro">€</option>  
  <option value="yen" disabled>¥</option>  
</select>
```

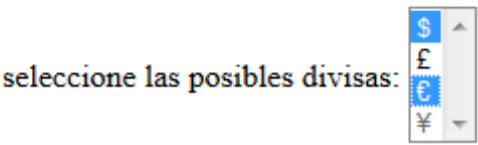


Ilustración 31: select múltiple

Cambios en HTML5

Los cambios introducidos en HTML5, en general pretenden simplificar ciertas declaraciones que en versiones previas, eran innecesariamente complejas. A continuación, veremos algunas de ellas.

El primer cambio que encontramos en HTML5 se encuentra en la definición del tipo de documento (primera línea del documento HTML). Quedaría de la siguiente manera:

```
<!DOCTYPE html>
```

en HTML5

Ilustración 33: definición del tipo de documento

También se ha simplificado la declaración de las siguientes etiquetas:

1. meta (específico de codificación de caracteres): ahora se ha simplificado quedando de la siguiente manera:

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

HTML5

Ilustración 34: codificación de caracteres en

2. link:

```
<link rel="stylesheet" href="styles.css" />
```

Ilustración 35: link en html5

3. script:

```
<script src="scripts.js"></script>
```

Ilustración 36: script en html5

Elementos desaprobadados en HTML5

Etiquetas eliminadas:

1. acronym: explica acrónimos. Mejor usar *abbr*.

2. **applet**: inserta un applet. Mejor usar la etiqueta *object*.
3. **basefont**: tamaño de fuente predeterminado. Mejor usar CSS.
4. **big**: resalta un texto. Mejor usar CSS.
5. **center**: centra un texto. Mejor usar CSS.
6. **dir**: lista directorios. Mejor usar la etiqueta *ul*.
7. **font**: estilo fuente. Mejor usar CSS.
8. **frame**, **frameset**, **noframes**: etiquetas para marcos. Mejor usar CSS.
9. **strikes**: muestra tachado en el texto. Mejor usar CSS.
10. **tt**: muestra el texto como espaciado simple. Mejor usar CSS.
11. **u**: subraya el texto. Mejor usar CSS.
12. **xmp**: para texto preformateado. Usar etiqueta *pre*.
13. **marquee**: desplazamiento de texto. Obsoleto por falta de usabilidad.

Atributos eliminados:

- **aling**: La alineación horizontal se debe utilizar con CSS.
- **link**, **vlink**, **alink**: en enlaces. Mejor usar CSS.
- **bgcolor**: el color de nuestro componente debe ir con CSS.
- **height**, **width**: altura y anchura. Mejor usar CSS.
- **scrolling** (en iframes): obsoleto debido a su falta de usabilidad.
- **valign**: para la alineación vertical se debe utilizar con CSS.
- **hspace**, **vspace**: espacios en blanco se consideran mala práctica.
- **cellpadding**, **cellspacing**, **border** (en tablas): mejor con CSS.
- **target**: Solo se debería usar el valor *_blank*.
- **font**: el tipo de letra debe definirse con CSS.

Etiquetas incorporadas a HTML5

Con la aparición del estándar HTML, se añaden una serie de nuevas etiquetas y atributos al lenguaje:

Etiquetas semánticas

El HTML 5 introduce una serie de elementos estructurales que facilitarán tanto el desarrollo de las páginas como también el análisis de las mismas por buscadores.

Son elementos que pretenden concretar la funcionalidad del elemento *div* (usado como divisor en bloque) y darle a su contenido un significado semántico.

Los elementos de HTML 5 que ayudan a estructurar la parte semántica de la página son:

- **main:** esta etiqueta contiene el cuerpo de la aplicación o del documento. El contenido principal consiste en aquel que está directamente relacionado con el tema central del documento o con la funcionalidad principal de la aplicación.
- **header:** el elemento *header* debe utilizarse para marcar la cabecera de una página (contiene el logotipo del sitio, una imagen, un cuadro de búsqueda etc).
Puede estar anidado en otras secciones de la página (es decir que no solo se utiliza para la cabecera de la página).
- **footer:** se utiliza para indicar el pie de la página o de una sección. Un pie de página contiene información general acerca de su sección el autor, enlaces a documentos relacionados, datos de derechos de autor etc.
- **nav:** representa una parte de una página que enlaza a otras páginas o partes dentro de la página. Es una sección con enlaces de navegación. No todos los grupos de enlaces en una página deben ser agrupados en un elemento *nav*.
Únicamente las secciones que consisten en bloques de navegación más importantes son adecuadas para el elemento de navegación.
- **article:** el elemento *article* representa una entrada independiente en un blog, revista, periódico etc.

Cuando se anidan los elementos `article`, los artículos internos están relacionados con el contenido del artículo exterior.

Por ejemplo: en una entrada de blog en un sitio que acepta comentarios, el elemento *article* principal agrupa el artículo propiamente dicho y otro bloque *article* anidado con los comentarios de los usuarios.

- `section`: Elemento de contenido con valor semántico dentro de una unidad mayor como pudiera ser 'article'.
- `aside`: el elemento *aside* representa una nota, consejo, una explicación, etc. Estas áreas son representadas a menudo como barras laterales en la revistas impresas.
El elemento puede ser utilizado para efectos de atracción, como las comillas tipográficas o barras laterales, para la publicidad, por grupos de elementos de navegación, y por otro contenido que se considera por separado del contenido principal de la página.
- `figure`: representa una unidad de contenido que es autosuficiente y que puede ser movido fuera del flujo principal del documento sin que esto afecte a su significado.
Por lo general contiene una figura o imagen.
- `figcaption`: está anidada dentro de `figure` y representa una nota o leyenda de la figura.



Ilustración 37

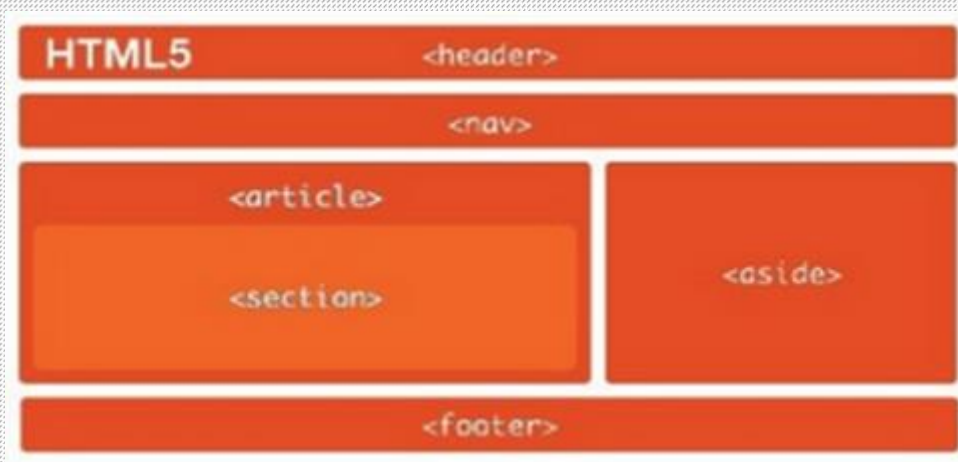


Ilustración 38

Etiquetas en formularios

En esencia, la estructura de los formularios permanece igual que en versiones anteriores, pero se han introducido elementos y atributos nuevos que enriquecen la manera en la que trabajamos con formularios.

- label: representa una etiqueta de un campo (texto explicativo de un campo de formulario).

Atributos:

- for: indica el identificador del elemento de formulario al que referencia. De esta forma, podemos pinchar en la etiqueta y el navegador pondrá el foco sobre el elemento referenciado.

```
<label for="nombre">introduzca su nombre: </label>  
<input id="nombre" type="text" name="nombre" />
```



introduzca su nombre:

Ilustración 39: label

Se ha ampliado el número de tipos para la etiqueta input:

- number: específico para números.
- search: específico para input de buscadores.

- url: específico para direcciones con formato URL.
- tel: campo específico para teléfono.
- email: campo de texto específico para e-mails.
- color: muestra una paleta de colores para elegir.

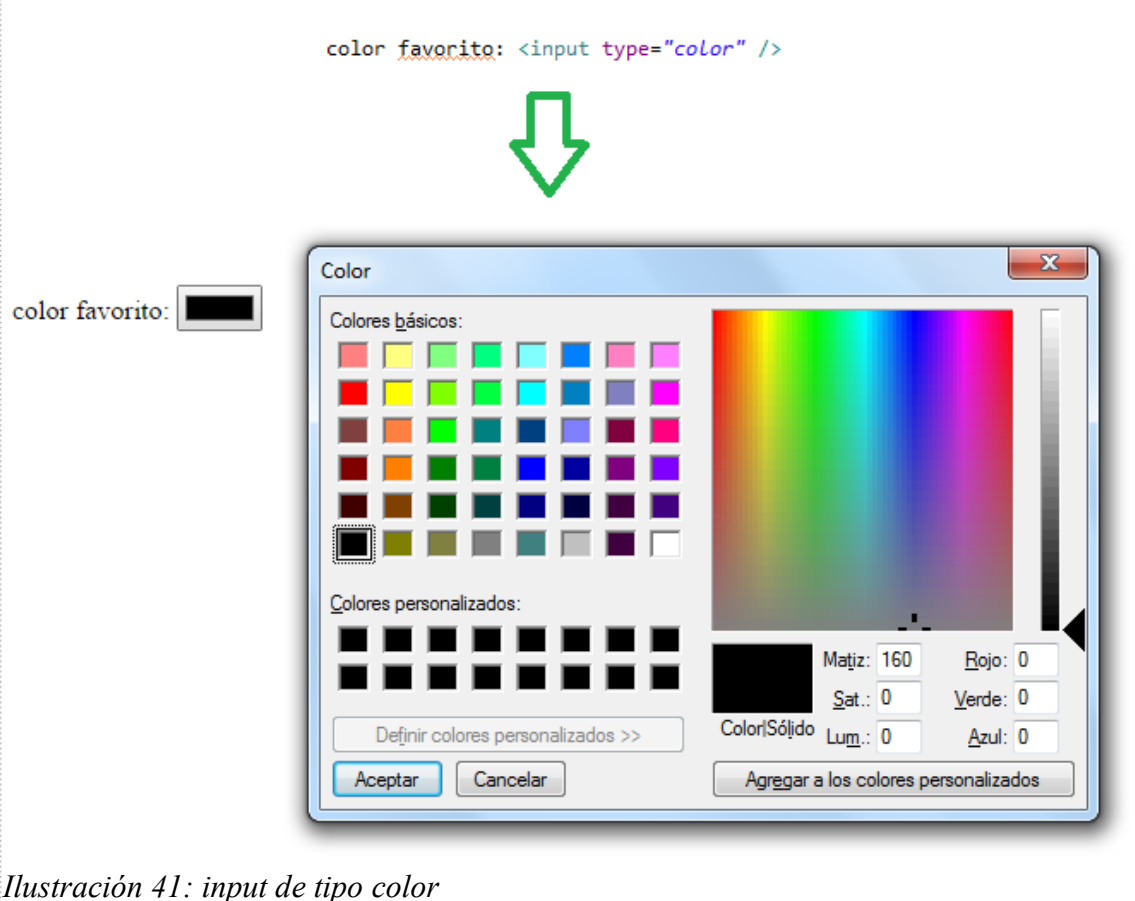
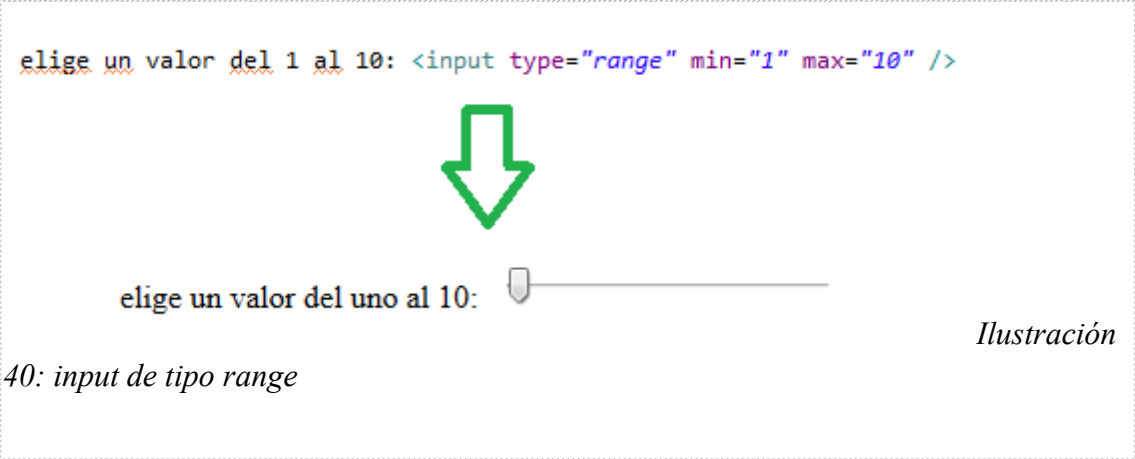


Ilustración 41: input de tipo color

- range: muestra un componente de tipo slider (barra de desplazamiento). Permite elegir un valor dentro de un rango.



Ilustración

40: input de tipo range

- `datetime`, `date`, `month`, `week`, `time` y `datetimelocal`: específicos para manejar fechas.

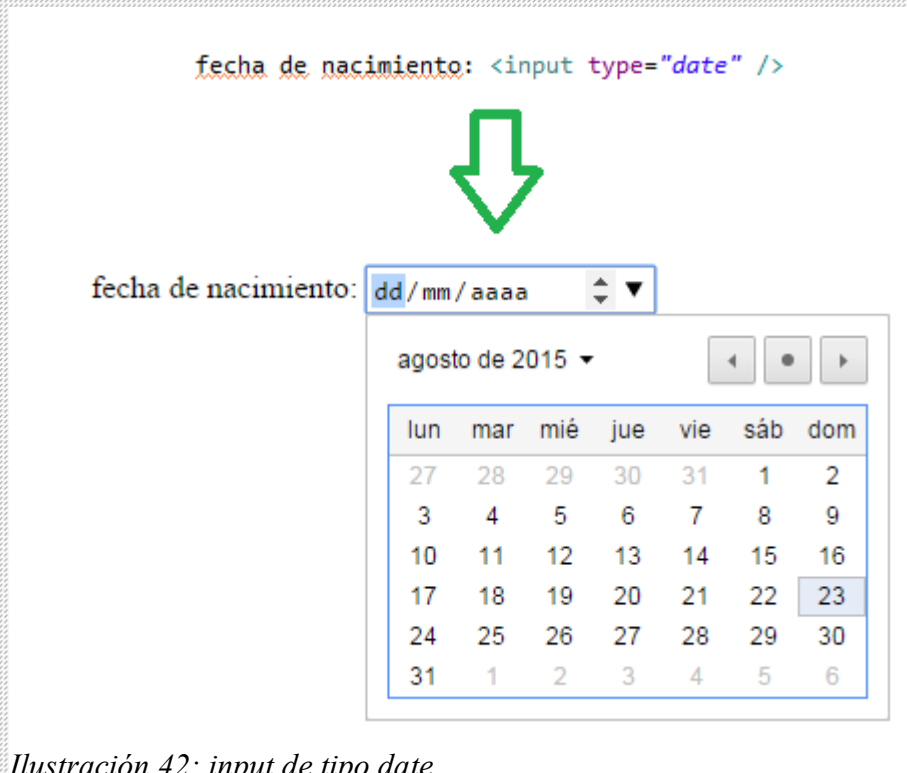


Ilustración 42: input de tipo date

Además, se añaden nuevos atributos a los formularios:

- `placeholder`: rellena el campo con un valor ilustrativo. Cuando pinchamos en el campo, este valor desaparece permitiéndonos editarlo.
- `required`: el campo es obligatorio, es necesario darle algún valor.
- `pattern`: permite definir una expresión regular para validar el valor del campo.
- `title`: permite añadir información extra al mensaje de error que genera la validación del campo.
- `novalidate`: añadiendo este atributo sobre el `input` de tipo `submit`, el formulario no se valida al pulsar el botón y se realiza la petición, aunque existan campos en rojo.

- `oninvalid`: evento que se dispara cuando la validación del formulario no se cumple. En el valor, se incluye el código JavaScript que va a manejar este evento (lo veremos más adelante).

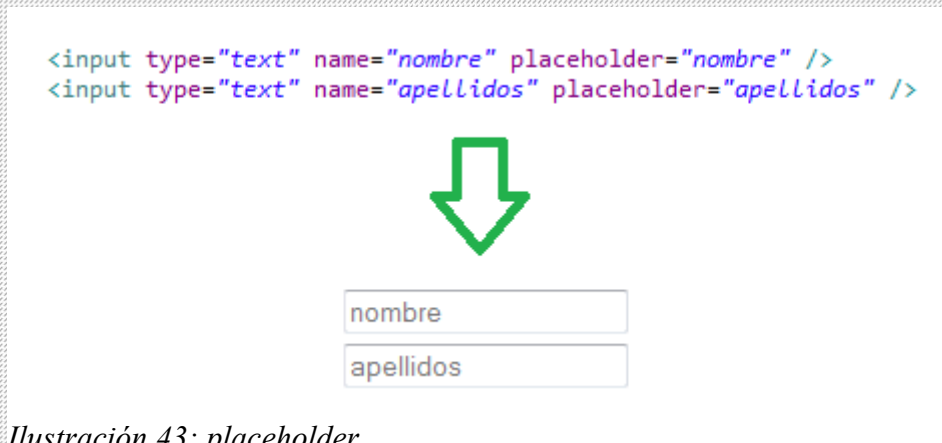


Ilustración 43: placeholder

Componentes multimedia HTML5

Además de los cambios que hemos visto anteriormente, se han añadido a HTML5 elementos que permiten contenido multimedia sin necesidad de plugins externos como Adobe Flash, Microsoft Silverlight o los Applets de Java.

A continuación, se detallan algunas de las nuevas etiquetas de HTML5 creadas como componentes multimedia:

- `audio`: permite la carga de ficheros de audio. Cada navegador puede soportar los formatos que se crea oportunos, es decir, el estándar HTML5 no regula que estándares de audio son permitidos. Los más habituales son mp3, ogg y wav.

La etiqueta *audio* cuenta con los siguientes atributos:

- `src`: URL donde se encuentra el fichero de audio que queremos que se reproduzca.
- `autoplay`: indica si el archivo se reproduce automáticamente al cargar la página.
- `loop`: el fichero se reproduce en bucle.

- controls: la presencia de este atributo indica que se debe mostrar el interfaz visual (reproducir, pausa, parar, volumen, etc.).
- autobuffer: su presencia indica que el fichero debe descargarse completamente antes de comenzar a ejecutarse.

```
<audio src="sonido.ogg" autoplay controls loop></audio>
```



Ilustración 44: visualización

Como no hay un formato de audio estándar adoptado por todos los navegadores, podemos agregar distintas fuentes eliminando el atributo `src` y añadiendo como contenido una etiqueta `source` por cada formato de audio que tengamos.

```
<audio autoplay controls loop>  
  <source src="sonido.ogg">  
  <source src="sonido.mp3">  
</audio>
```

Ilustración 45: etiqueta audio con múltiples fuentes

El elemento `source` indica, a través de la propiedad `src`, la ubicación del archivo de audio. El orden en el que añadimos estas fuentes es importante. Primero el navegador busca la primera fuente y verifica que puede reproducirla en ese formato. En caso afirmativo, reproduce ese fichero y, en caso negativo, pasa a la siguiente fuente.

- Video: este elemento nos permite mostrar un video sin necesidad de un plugin externo. En este momento, los navegadores permiten mostrar una cantidad de formatos de video muy limitados.

Atributos:

- src: URL donde se encuentra el fichero de video que queremos reproducir.
- controls: visualiza los controles de reproducción del video (reproducir, pausa, parada, volumen, etc.).
- autoplay: el video se reproduce inmediatamente.
- width: anchura del video en píxeles
- height: altura del video en píxeles.

Al igual que con la etiqueta video, podemos utilizar múltiples fuentes para un mismo video, suprimiendo el atributo *src* y añadiendo tantos elementos *source* como fuentes tengamos. El elemento *source* también acepta el atributo *type*.

```
<video controls>  
  <source src="http://www.quirksmode.org/html5/videos/big_buck_bunny.mp4" type="video/mp4">  
</video>
```

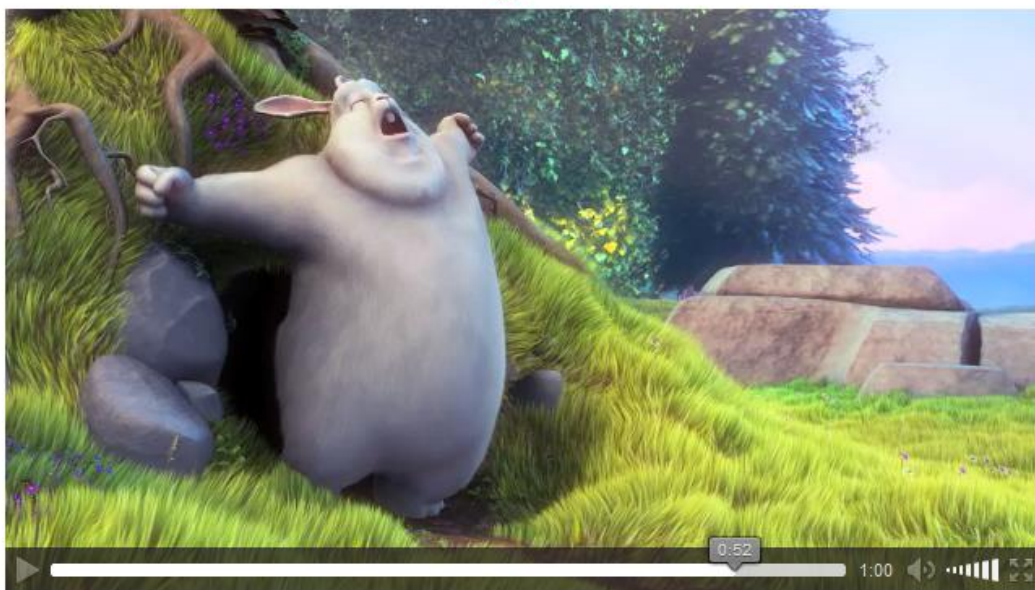


Ilustración 46: video

- Canvas: con esta etiqueta podemos crear en un documento un área en la que es posible dibujar mediante JavaScript.

El objetivo de este componente es generar gráficos en el cliente.

Para poder hacer un uso correcto de este componente, debemos manejar el lenguaje JavaScript que veremos más adelante.

De momento nos limitaremos a ver ejemplos para ver como se comporta.

Atributos de la etiqueta:

- width: anchura en píxeles del canvas.
- height: altura en píxeles del canvas.

Es conveniente definir el atributo *id* para poder referenciar cómodamente el componente desde JavaScript.

El elemento *canvas* nos permite crear gráficos en 2d y 3d.

A continuación veremos un ejemplo de *canvas*. Veremos que es un ejemplo más complejo que los anteriores. El ejemplo cuenta con los siguientes elementos:

1. elemento canvas en un documento HTML. Se declara en donde queramos tener el lienzo, y le damos un tamaño. También hemos añadido el atributo *id*.

2. Código Javascript. Vemos que se declara la función *window.onload*, cuyo código se ejecuta cuando la página ha cargado por completo en el navegador.

Dentro de dicha función, se recupera el elemento *canvas* de HTML5 mediante la función *getElementById* del objeto *document* (más adelante entraremos en detalles) y se guarda en la variable "lienzo".

Después, creamos el contexto del lienzo en el que vamos a dibujar. En este caso, esta tarea la realiza la llamada a la función *crearContexto2d* que crea un contexto para gráficos en 2d se guarda en la variable *contexto2d*.

Por último, se llama a la función *pintar rectángulo* que recibe como parámetro el contexto, y realiza el dibujo sobre el.

3. Representación del *canvas* en el navegador una vez pintado nuestro rectángulo.

```
<canvas id="lienzo1" width="300" height="200" class="borde-gris"></canvas>
```



```
<script>
  var crearContexto2d = function(lienzo) {
    if(!lienzo) return null;
    if(!lienzo.getContext) return null;

    return lienzo.getContext('2d');
  }

  var pintarRectangulo = function(contexto2d) {
    if(!contexto2d) return;

    contexto2d.fillStyle = 'rgb(200,0,0)';
    contexto2d.fillRect (10, 10, 100, 100);
  }

  window.onload = function() {
    var lienzo = document.getElementById('lienzo1');
    var contexto2d = crearContexto2d(lienzo);
    pintarRectangulo(contexto2d);
  }
</script>
```

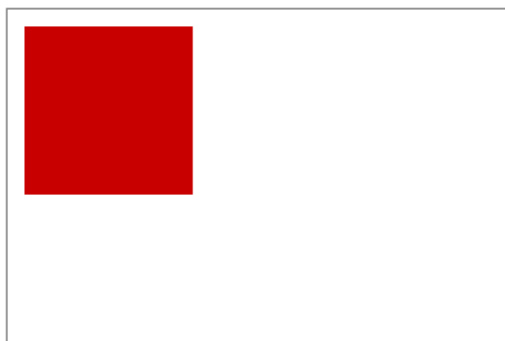


Ilustración 47: canvas

Compatibilidad entre navegadores

El motor de renderizado es el encargado de leer los datos recibidos de una web (HTML, XML, CSS, XSL...) y representarlos en pantalla. **Cada navegador tiene su motor de renderizado**, como Webkit, el motor de Safari, Opera y hasta hace poco de Chrome (ahora utiliza Blink); Gecko, el motor de Firefox; o Trident, el motor de Internet Explorer.

Pues bien, aunque en gran medida todos cumplen los estándares y representan las etiquetas de la misma forma, **hay elementos que no todos interpretan igual**, o incluso debido a su novedad, **algunos no llegan ni tan siquiera a soportar**. Por ejemplo, hasta hace poco Trident, el motor de Internet Explorer, era incapaz de interpretar la gran mayoría de etiquetas HTML5.

Si quieres utilizar, por ejemplo, el elemento canvas es posible que tengas más de un problema al probar con los distintos navegadores.

Pero si las etiquetas pueden ser un problema, **los estilos CSS lo son aún más**. Cosas como las sombras en los textos funcionan en algunos navegadores sí y en otros no, el tratamiento de los bordes no es igual en Firefox que en Explorer, lo que puede causar que se descuadre algún elemento, hacer bordes redondeados es diferente según el navegador, e incluso algo tan básico como los márgenes (margin) difiere dependiendo del navegador y su versión.

Consulta tablas de compatibilidad

Una de las primeras cosas a tener en cuenta son las tablas de compatibilidades, o lo que es lo mismo, **tablas que te indican si el recurso que quieres utilizar es compatible con todos los navegadores o no**. Te puedo recomendar las siguientes:

- quirksmode.org: sin duda la página más completa sobre el tema. Hay disponibles tablas de compatibilidad tanto de CSS, el DOM y test de elementos HTML5. A día de hoy está completamente actualizado con las últimas versiones de los navegadores (como IE11).
- sitepoint: sitepoint tiene un apartado llamado reference en el que no sólo encontraremos tablas de compatibilidades sino también explicaciones sobre cada atributo, posibles valores, etc... El gran

- problema es que no está tan actualizado como quirksmode y hay tablas algo antiguas.
- [w3cschools](#): aunque w3cschools es más útil para conocer tags, atributos y otros elementos del estándar, junto a cada uno de ellos nos encontraremos los iconos con los navegadores compatibles. Por desgracia únicamente tiene en cuenta las últimas versiones de los mismos, por lo que no nos ayuda a hacer compatible nuestra página con navegadores antiguos.

Usar librerías

No reinventar la rueda. Puedes **utilizar librerías que hacen lo que quieres. Se trata de un conjunto de elementos de código reutilizables que no conforman una aplicación en sí misma, si no que están orientadas a ser utilizadas en otros programas. Suelen estar enfocadas a un campo concreto (por ejemplo, librerías para manejar fechas, operaciones matemáticas, o en este caso, a uso de componentes HTML).**

Usar librerías en el desarrollo de páginas web es una forma de evitarte los problemas de compatibilidad entre navegadores. Por lo general estarán bien probadas y los problemas serán mucho menores.

Por ejemplo, si queremos incluir un reloj en nuestra página, podemos acudir a bastantes librerías existentes. Por ejemplo, la librería FlipClock nos permite incluir un reloj, temporizador o cronómetro con efecto moneda. A su vez, FlipClock usa otra librería llamada JQuery, por lo que también la necesitaremos para que funcione FlipClock.

Para usarla, necesitaríamos importar las librerías JavaScript y sus estilos css y usarla tal y como dice su documentación. A continuación, vemos como se usa:


```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="/css/flipclock.css">
  </head>
  <body>
    <div class="my-clock"></div>
    <script src="/js/jquery/jquery.js"></script>
    <script src="/js/flipclock/flipclock.min.js"></script>
    <script>
      var clock = new FlipClock($('.my-clock'), {});
    </script>
  </body>
```

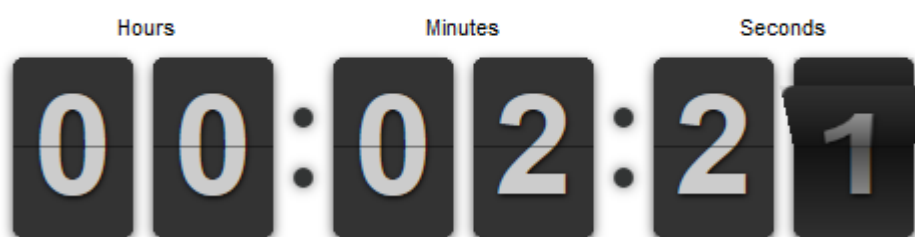


Ilustración 48: ejemplo de uso de librerías

Creación de hojas de estilo

CSS (Cascading Style Sheets) es un lenguaje usado para definir la presentación de un documento HTML.

La idea que subyace detrás del CSS es separar la estructura de un documento de su presentación.

Cuando usamos HTML, todos los elementos mostrados tienen un estilo por defecto definido, CSS redefine estos atributos para mejorar la presentación de una página o, incluso permite visualizar un mismo HTML de varias formas diferentes.

Tipos de hojas de estilo

- Inline → Usando CSS directamente sobre el elemento HTML que queremos modificar. Se realiza usando la etiqueta `style` de los elementos HTML, y como valor, se añaden todos los estilos CSS que

queramos aplicar separados por el carácter punto y coma. Evidentemente usar CSS inline no es lo más recomendable, debe usarse para cosas muy puntuales, ya que su mantenimiento es complicado (está sobre la propia página, hay que buscarlo). Sin embargo, para hacer pruebas o ejemplos pequeños es muy útil.

```
<h1 style="color: red;">esto es un título</h1>
```

Ilustración 49: estilo inline

- Internal → Usando la etiqueta *style* en el área *head* del documento HTML para redefinir el estilo de los elementos HTML. Parece algo más adecuado que *inline* puesto que unifica todos los estilos de una página en un mismo sitio. Sin embargo, sigue sin permitir reutilizar estilos entre distintas páginas de nuestro sitio web.

```
<style>
  h1.rojo {
    color: red;
  }
</style>
```

...

```
<h1 class="rojo">esto es un título</h1>
```

Ilustración 50: estilo internal

- External → Usando ficheros externos que posean la configuración CSS que queramos aplicar. Es el sistema más recomendado ya que fomenta la reutilización y el mantenimiento de las hojas de estilo en diferentes páginas HTML.


```
<link rel="stylesheet" href="styles.css" />
```

...

```
<h1 class="rojo">esto es un título</h1>
```

Ilustración 51: estilo external

Todos los ejemplos anteriores son equivalentes. En todos los casos se muestra un título de color rojo como el siguiente:

esto es un título

Ilustración 52: estilos aplicados de forma equivalente

Como se define un estilo

En primer lugar, veremos como se definen las propiedades CSS. El formato para definir un estilo sería el siguiente:

```
selector {  
    propiedad_1: valor_1;  
    propiedad_2: valor_2;  
  
    ...  
  
    propiedad_n: valor_n;  
}
```

Texto 8: definición de un estilo

Existen las siguientes maneras para definir el elemento al que aplica un estilo.

Selectores

- *: selector universal. Se aplica a todas las etiquetas.


```
* {...}
```

Texto 9: selector universal

- *etiqueta*: selector de tipo o etiqueta. Se aplica a todas las etiquetas con ese nombre.

```
div {...}
```

Texto 10: selector de etiqueta

- *etiqueta_1,etiqueta_2, ... ,etiqueta_n*: agrupación de selectores. Se aplica a todas las etiquetas que coincidan con dichos nombres de etiqueta.

```
h1, h2, h3 {...}
```

Texto 11: agrupación de selectores

- *etiqueta subetiqueta*: selector descendiente. Se aplica a todas las subetiquetas descendientes directas o no de una etiqueta.

```
div p {...}
```

Texto 12: selector descendiente

- *.nombre_estilo*: selector de clase. Se aplica a las etiquetas que tengan como valor de atributo *class* el valor *nombre_clase*.

```
.nombre_clase {...}
```

Texto 13: selector de clase

- *#id*: selector de identificador. Se aplica a la etiqueta cuyo atributo *id* coincida con *identificador*.


```
#identificador {...}
```

Texto 14: selector de identificador

- *etiqueta>subetiqueta*: selector de hijos. Se aplica a todas las subetiquetas (derecha) que sean hijos directos de la etiqueta (izquierda).

```
div>p {...}
```

Texto 15: selector de hijos

- *etiqueta_1+etiqueta_2*: selector adyacente. Se aplican a todas las etiquetas de tipo *etiqueta_2* que estén justo al lado de una etiqueta *etiqueta_1*.

```
div+p {...}
```

Texto 16: selector adyacente

- *etiqueta[atributo]*: Selector de atributos por nombre. Selecciona todas las etiquetas que contengan dicho atributo.

```
input[selected] {...}
```

Texto 18: selector de atributos por nombre

- *etiqueta[atributo="valor"]*: selector de atributos por valor. Selecciona todas las etiquetas que contengan dicho atributo con ese determinado valor.

```
input[name="dni"] {...}
```

Texto 17: selector de atributos por valor

Pseudo-clases CSS3

Las pseudoclasas se aplican a etiquetas que se encuentran en un estado concreto.

De esta forma, podemos aplicar estilos distintos, por ejemplo, a un estilo visitado y a otro no visitado.

```
selector:estado {  
    propiedad_1: valor_1;  
    propiedad_2: valor_2;  
  
    ...  
  
    propiedad_n: valor_n;  
}
```

Texto 19: pseudoclasas css

Estos estilos se declaran de la siguiente forma:

- hover: selecciona el elemento sobre el cual pasa el raton.
- active: selecciona el elemento que activa el usuario a traves de un click.
- focus: selecciona el elemento que tiene el foco del navegador. También hay algunos específicos para enlaces:
- link: selecciona los enlaces que no hayan sido visitados por el usuario.

- visited: selecciona los enlaces que hayan sido visitados por el usuario.

Estudio de las propiedades de estilo CSS3



Ilustración 53: pseudo-classes css

Después de ver como se define un estilo, vamos a ver las propiedades que podemos configurar en el:

- Tipo de letra (fuente)
 - font-family: tipo de letra
 - font-size: tamaño de letra
 - font-style: inclinación (cursiva). Posibles valores:
normal | italic | oblique
 - font-weight: grosor de la fuente. Posibles valores:
normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900
- Texto
 - color: color del texto.
 - direction: dirección del texto. Posibles valores
ltr | rtl
 - letter-spacing: espacio entre caracteres.

- `line-height`: espaciado entre líneas.
- `text-align`: alineación de texto. Posibles valores:
center | justify | left | right
- `vertical-align`: alineación vertical. Se puede añadir un valor numérico o uno de los siguientes valores:
baseline | bottom | middle | sub | super | text-bottom | text-top | top
- `text-decoration`: decoración del texto. Posibles valores:
none | blink | line-through | overline | underline
- `text-indent`: tamaño de sangría de la primera línea de un párrafo.
- `white-space`: comportamiento de saltos de línea y espacios en blanco. Posibles valores:
normal | nowrap | pre | pre-line | pre-wrap
- `word-spacing`: espacio entre palabras.
- `Text-shadow`: sombras del texto.

Bordes

- `border-style`: estilo del borde. Puede contener los siguientes valores:
none | hidden | dashed | dotted | double | groove | inset | outset | ridge | solid

También existe propiedades que permiten definirlo por separado:

border-[left | right | top | bottom]-style

- `border-color`: color del borde. Su valor puede ser un color o el valor *transparent* si es transparente.

También existe propiedades que permiten definirlo por separado:

border-[left | right | top | bottom]-color

- `border-width`: anchura del borde. Puede ser un valor en píxeles o uno de los siguientes valores:
medium | thick | thin

También existe propiedades que permiten definirlo por separado:

border-[left | right | top | bottom]-width

- `border`: configuración completa de un borde. La sintaxis es la siguiente: `border-style border-color border-width`

También existe propiedades que permiten definirlo por separado:

`border-[left | right | top | bottom]`

- `border-radius`: borde con esquinas redondeadas.
- `box-shadow`: borde con sombra.

```
p.dotted {
  border: dotted gray 1px;
}

...

<p class="dotted">
  nota: esto es un elemento div con borde
</p>
```



nota: esto es un elemento div con borde

Ilustración 54: border

- Márgenes
 - `margin`: márgenes exteriores (los 4 márgenes). Admite de uno a 4 valores (uno por cada margen). Los valores pueden ser números o el valor `auto`.

También existe propiedades que permiten definirlo por separado:

`margin-[left | right | top | bottom]`

- Márgenes interiores
 - `padding`: márgenes interiores (los 4 márgenes).

También existe propiedades que permiten definirlo por separado:

`padding-[left | right | top | bottom]`

- Listas

- `list-style-image`: imagen del marcador. Como valor puede contener el valor *none* o una URL con el formato `url('x')` donde x es la URL de la imagen.
- `list-style-position`: posición del marcador. Posibles valores: *inside* | *outside*
- `list-style-type`: tipo de marcador. Puede ser uno de lo siguientes valores:
none | *circle* | *disc* | *square* | *decimal* | *decimal-leading-zero* | *lower-alpha* | *upper-alpha* | *lower-greek* | *lower-latin* | *upper-latin* | *lower-roman* | *upper-roman* | *armenian* | *georgian* | *hebrew*⁽⁻⁾ | *cjk-ideographic* | *hiragana* | *katakana* | *hiragana-iroha* | *katakana-iroha*
- `list-style`: propiedad compuesta. Tiene la siguiente sintaxis:
list-style-image list-style-position list-style-type

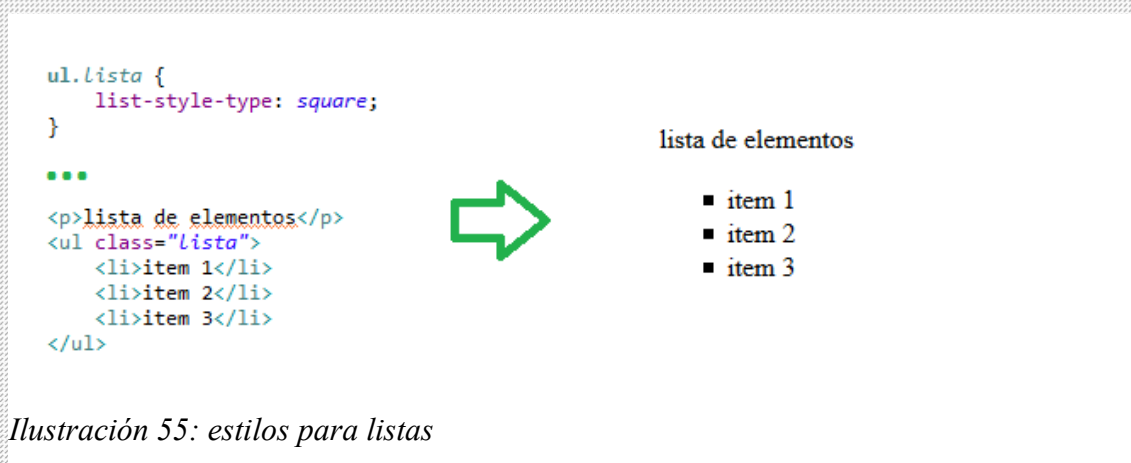


Ilustración 55: estilos para listas

Fondos

- `background-color`: color de fondo.
- `background-image`: imagen de fondo. Como valor puede contener el valor *none* o una URL con el formato `url('x')` donde x es la URL de la imagen.
- `background-position`: posición de la imagen de fondo.
- `background-repeat`: repetición de la imagen de fondo. Puede contener los siguientes valores:
no-repeat | *repeat* | *repeat-x* | *repeat-y*

- background-origin: origen de la imagen de fondo. Puede tener los siguientes valores:
border-box | padding-box | content-box
- background-clip: límite de la imagen de fondo.
- background-size: tamaño de la imagen de fondo.
- background: propiedad combinada. Tiene la siguiente sintaxis:
*background-color background-image background-position
background-repeat background-origin background-clip background-size*



Ilustración 56: background

Tablas

- border-collapse: tipo de bordes. Valores permitidos:
collapse | separate
- border-spacing: separación entre celdas.
- caption-size posición de la leyenda de la tabla. Valores permitidos:
top | bottom | left | right
- empty-cells: borde en celdas vacías. Valores permitidos:
hide | show
- table-layout: tipo de distribución de la tabla. Valores permitidos:
auto | fixed


```
table {
  border-collapse: collapse;
  caption-side: top;
}
table td {
  border: solid gray 1px;
}
```

...

```
<table>
  <tr>
    <td>elemento 1</td>
    <td>elemento 2</td>
  </tr>
  <tr>
    <td>elemento 3</td>
    <td>elemento 4</td>
  </tr>
  <caption>leyenda</caption>
</table>
```



leyenda	
elemento 1	elemento 2
elemento 3	elemento 4

Ilustración 57: estilos tabla

- Tamaño de elementos
 - width: anchura del elemento. Admite como valor un tamaño customizado o el valor *auto*.
 - min-width: anchura mínima del elemento.
 - max-width: anchura máxima del elemento.
 - height: altura del elemento. Admite como valor un tamaño customizado o el valor *auto*.
 - min-height: altura mínima del elemento.
 - max-height: altura máxima del elemento.
 - overflow: comportamiento si se desborda el elemento debido a su contenido. Puede tener los siguientes valores:
auto | hidden | scroll | visible

Posicionamiento de elementos

- float: modo de posicionamiento flotante. Indica sobre que elemento flota. Admite los siguientes valores:
none | left | right
- clear: indica el lado del elemento en el que no puede haber elementos flotantes. Puede tener los siguientes valores:
none | both | left | right
- position: tipo de posicionamiento del elemento. Admite los siguientes valores:
absolute | fixed | relative | static
- top: posición del borde superior del elemento. Admite una distancia personalizada o el valor *auto*.
- right: posición del borde derecho del elemento. Admite una distancia personalizada o el valor *auto*.
- bottom: posición del borde inferior del elemento. Admite una distancia personalizada o el valor *auto*.
- left: posición del borde izquierdo del elemento. Admite una distancia personalizada o el valor *auto*.

- z-index: prioridad de visibilidad en elementos apilados (eje z). Admite como valor un número entero que define su prioridad o el valor *auto*.
- Interfaz de usuario
 - cursor: tipo de cursor. Admite una URL con el icono del cursor *url('x')* o los valores *auto* | *crosshair* | *default* | *help* | *move* | *pointer* | *progress* | *n-resize* | *ne-resize* | *e-resize* | *se-resize* | *s-resize* | *sw-resize* | *w-resize* | *nw-resize* | *text* | *wait*
 - outline-color: color de los bordes. Mismos valores que *color*.
 - outline-width: anchura de los bordes. Mismos valores que *border-width*
 - outline-style: estilo de los bordes. Mismos valores que *border-style*.
 - outline: estilo combinado

Otras características de CSS3

Con CSS3 también podemos realizar modificaciones de las siguientes características:

- Color
 - Colores HSL
 - Colores HSLA
 - Colores RGBA
 - Opacidad (elementos translúcidos)
- Degradado CSS3
 - Degradados lineales
 - Degradados lineales de repetición

- Degradados radiales de repetición

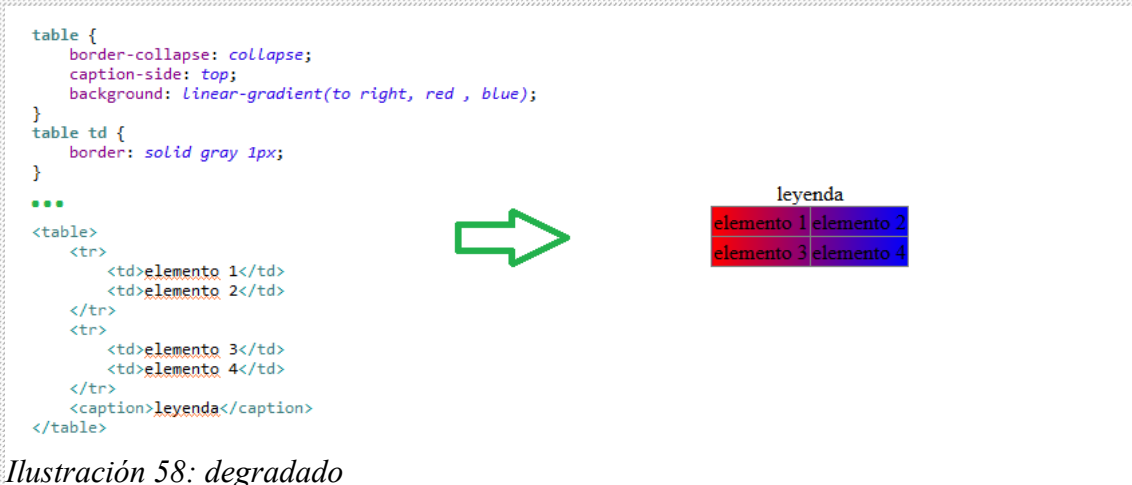


Ilustración 58: degradado

- Degradados radiales
- Otros
- Media queries
- Múltiples columnas de texto
- Propiedades orientadas a discurso o lectura automática de páginas web
- Animaciones CSS3

Posicionamiento SEO

SEO son las siglas de Search Engine Optimization, o lo que es lo mismo, optimización de motores de búsqueda. Una vez ya sabemos a que hace referencia esas siglas, será mucho más fácil enfocar para que nos va a servir esto. Hay que tener en cuenta que las técnicas de SEO se utilizan para posicionar en resultados naturales u orgánicos.

¿En qué consiste? Consiste en un conjunto de técnicas que aplicadas basándose en una buena estrategia pueden conseguir el objetivo de optimizar la web para que los buscadores la muestren en los primeros resultados. Hay que tener en cuenta que no todas las técnicas son buenas, muchas de ellas son penalizables y por lo tanto aplicar las malas artes en materia SEO puede hacer que tu web desaparezca de los buscadores. A este tipo de técnicas se les llama "Black hat", y

su finalidad es alterar los resultados de las búsquedas naturales. Los buscadores pueden penalizar a las webs que usan "Black Hat".

Algunas de las técnicas penalizadas son:

- keyword stuffing: repetición indiscriminada de palabras clave en la web de tal forma que el texto queda totalmente ilegible o sin sentido para los usuarios.
- El texto oculto: redactar texto con el mismo color del fondo para que el usuario no lo vea y si lo haga el buscador
- doorway page: páginas que redirigen a otra sin usar una redirección tipo 301.
- cloaking: trata de mostrar cosas diferentes al usuario y al buscador.

Concluyendo, hay varias técnicas que más vale no usar y ser conocedores de ellas para no caer en la trampa. Una técnica mal aplicada puede hacerte perder a medio-largo plazo unas posiciones muy valiosas en los buscadores y eso conllevaría una pérdida mayor de tiempo.

A continuación, veremos algunas técnicas de posicionamiento:

Correcta redacción de los contenidos

Hay que tener en cuenta que nuestra página no será vista únicamente por buscadores o por seres humanos. Debe poder ser correctamente interpretada por ambas. Por eso es importante cuidar el contenido y mantener una densidad de palabras y frases clave manteniendo un equilibrio entre visitantes y buscadores.

Es muy importante ofrecer un contenido de buena calidad, evitando repeticiones sobre dicho contenido.

También es importante hacer un correcto uso de la estructura de la página y las etiquetas HTML. Las etiquetas HTML tienen un alto contenido semántico. Un buscador no dará la misma importancia a una etiqueta *h1* o *a* que a una etiqueta *p*.

Metatags

Hoy en día hay 3 etiquetas metas que son las más importantes:

- **title:** en primer lugar, deberíamos apoyarnos en el título y la metatag "title". Aunque ya tenemos la etiqueta <title> (la hemos visto con anterioridad), podemos hacer uso de esta para enfatizarla.
- **description:** su única finalidad a día de hoy es mostrar ese texto en los resultados de los buscadores. Se aconseja no pasar de los 160 caracteres, ya que de lo contrario no se mostrará completo y por lo tanto no habremos optimizado esta metatag.

```
<meta name="description" content="encuentra tu viaje en internet" />
```

Texto 20: metas seo

- **keyword:** se trata de una lista de palabras clave que hacen referencia al contenido de nuestra página. A pesar de no ser valorado por muchos buscadores, otros si lo hacen y hay que dejar esta meta bien optimizada.
No vale de nada poner 50 palabras clave (ni 20) si la elección de las "keywords" no es minuciosa. Deben ser palabras naturales, altamente descriptivas de nuestro contenido y coherentes con la temática de nuestra página.

Enlaces hacia nuestra web.

La construcción de enlaces hacia nuestra web desde otras páginas, mejora el posicionamiento de nuestra web. Es el medidor de importancia de webs que utilizan los buscadores como Google.

Usar el atributo alt de las imágenes

Algunos buscadores tienen en cuenta la etiqueta *alt* de las imágenes (etiqueta *img*). Se tienen en cuenta, sobre todo, para las búsquedas de imágenes.

Herramientas

Existen herramientas para medir la calidad SEO de nuestra página. Un ejemplo sería <http://page-rank.es>, donde nosotros introducimos la ruta de nuestra página y evalúa la calidad de su contenido SEO dandonos una puntuación.



Ilustración 59: page-rank.es

Trabajar en equipo

Existen herramientas para trabajar en equipo, que además nos permiten versionar nuestras aplicaciones. Son los controles de versiones. Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación. Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (por ejemplo, para algún cliente específico). Ejemplos de este tipo de herramientas son entre otros Subversion y GIT. Para colaborar en un proyecto usando un sistema de control de versiones lo primero que hay que hacer es crearse una **copia local** obteniendo información del repositorio. A continuación el usuario puede modificar la copia. Existen dos esquemas básicos de funcionamiento para que los usuarios puedan ir aportando sus modificaciones:

- De forma **exclusiva**: En este esquema para poder realizar un cambio es necesario comunicar al repositorio el elemento que se desea modificar y el sistema se encargará de impedir que otro usuario pueda modificar dicho elemento. Una vez hecha la modificación, ésta se comparte con el resto de colaboradores. Si se ha terminado de modificar un elemento entonces se libera ese elemento para que otros lo puedan modificar. Subversion, por ejemplo, permite este esquema de funcionamiento.
- De forma **colaborativa**: En este esquema cada usuario modifica la copia local y cuando el usuario decide compartir los cambios el sistema automáticamente intenta combinar las diversas modificaciones. El principal problema es la posible aparición de conflictos que deban ser solucionados manualmente o las posibles inconsistencias que surjan al modificar el mismo fichero por varias personas no coordinadas. Subversion o GIT implementan este esquema.

Arquitecturas de almacenamiento

Podemos clasificar los sistemas de control de versiones atendiendo a la arquitectura utilizada para el almacenamiento del código:

- **Centralizados**: existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos). Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones fuertes (como crear una nueva rama) necesitan la aprobación del responsable. Subversión implementa esta arquitectura de almacenamiento.

Las principales ventajas de este sistema son:

- En los sistemas distribuidos hay menos control a la hora de trabajar en equipo ya que no se tiene una versión centralizada de todo lo que se está haciendo en el proyecto.
- En los sistemas centralizados las versiones vienen identificadas por un número de versión. Sin embargo, en los sistemas de control de versiones distribuidos no hay números de versión, ya que cada repositorio tendría sus propios números de revisión dependiendo

- de los cambios. En lugar de eso cada versión tiene un identificador al que se le puede asociar una etiqueta (tag).
- **Distribuidos:** Cada usuario tiene su propio repositorio. Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos. Es frecuente el uso de un repositorio, que está normalmente disponible, que sirve de punto de sincronización de los distintos repositorios locales. GIT implementa este sistema.

Ventajas de sistemas distribuidos:

- Necesita menos veces estar conectado a la red para hacer operaciones. Esto produce una mayor autonomía y una mayor rapidez.
- Aunque se caiga el repositorio remoto la gente puede seguir trabajando.
- Al hacer los distintos repositorios una réplica local de la información de los repositorios remotos a los que se conectan, la información está muy replicada y por tanto el sistema tiene menos problemas en recuperarse si por ejemplo se quema la máquina que tiene el repositorio remoto. Por tanto hay menos necesidad de copias de seguridad (backups). Sin embargo, los backups siguen siendo necesarios para resolver situaciones en las que cierta información todavía no haya sido replicada.
- Permite mantener repositorios centrales más limpios en el sentido de que un usuario puede decidir que ciertos cambios realizados por él en el repositorio local, no son relevantes para el resto de usuarios y por tanto no permite que esa información sea accesible de forma pública. Por ejemplo, es muy útil se pueden tener versiones inestables o en proceso de codificación o también tags propios del usuario.
- El servidor remoto requiere menos recursos que los que necesitaría un servidor centralizado ya que gran parte del trabajo lo realizan los repositorios locales.
- Al ser los sistemas distribuidos más recientes que los sistemas centralizados y, al tener más flexibilidad por tener un repositorio local y otro/s remotos. Estos sistemas han sido diseñados para hacer fácil el uso de ramas (creación, evolución y fusión) y poder

aprovechar al máximo su potencial. Por ejemplo, se pueden crear Mramas en el repositorio remoto para corregir errores o crear funcionalidades nuevas. Pero también se pueden crear ramas en los repositorios locales para que los usuarios puedan hacer pruebas y dependiendo de los resultados fusionarlos con el desarrollo principal o no. Las ramas dan una gran flexibilidad en la forma de trabajo.

Algunos de los programas para controles de versiones más extendidos son:

- CVS (Concurrent Versions System): sistema de libre distribución bajo licencia GNU y, también desarrollado por GNU que funciona bajo el esquema de cliente-servidor. Al igual que el resto de sistemas de control de versiones, su finalidad es proveer a un equipo de desarrollo de una herramienta que permita gestionar los distintos cambios realizados por el equipo de desarrollo y generar un único código final en la medida de lo posible.
- Subversión: sistema de libre distribución bajo licencia Apache y desarrollada por Apache Software Foundation. Está basado en CVS, pero lo mejora en muchos aspectos como la gestión de ficheros, envíos atómicos de los cambios (todos los cambios son tratados como un único cambio), etc.
- Clear Case: solución propietaria desarrollada por IBM. Se trata de una herramienta principalmente enfocada a trabajar dentro del ecosistema de aplicaciones de desarrollo de IBM.
- VSS (Visual SourceSafe): sistema de control de versiones de Microsoft (desarrollado inicialmente por la empresa One Tree Software). Su sistema de transferencia de archivos se basa en SMB (el sistema de compartición de recursos de Microsoft). Se integra con el entorno de desarrollo Visual Studio.
- StarTeam: sistema de control de versiones propietario de la empresa Borland (previamente desarrollado por Starbase Corporation y posteriormente adquirido por Borland). Es una aplicación cliente / servidor cuya parte servidora se implementa mediante una base de datos relacional (Oracle o SQL Server) y un cliente pesado desarrollado en Java.

También existen otros clientes como un cliente de línea de comandos o el cliente implementado en el entorno de desarrollo Jbuilder.

- GIT, Bazaar y Mercurial: a diferencia de los anteriores, no siguen el esquema cliente-servidor, si no que son sistemas de control de versiones distribuidos.

Construcción de una web completa

Cuando empezamos la construcción de una página web completa debemos tener dos cosas claras: cual va a ser el contenido de dicha página y cual queremos se sea su diseño.

Para desarrollar el contenido, debemos tener en cuenta la calidad de sus textos y las etiquetas que escogemos para organizarlo en nuestro documento.

Recordemos la existencia de las etiquetas semánticas y que importancia tiene todo esto en cuanto a posicionamiento en los buscadores (ya hemos visto en otra sección que el contenido y su organización dentro de la página influye y mucho en como los buscadores califican nuestras páginas).

A continuación, vemos como quedaría un documento de ejemplo:


```
<!DOCTYPE html>
<html>
<head>
  <title>My first styled page</title>
</head>
<body>
  <nav>
    <ul>
      <li><a href="index.html">Home page</a>
      <li><a href="musings.html">Musings</a>
      <li><a href="town.html">My town</a>
      <li><a href="links.html">Links</a>
    </ul>
  </nav>
  <section>
    <h1>My first styled page</h1>
    <p>Welcome to my styled page!</p>
    <p>
      It lacks images, but at least it has style.
      And it has links, even if they don't go anywhere&hellip;
    </p>
    <p>There should be more here, but I don't know what yet.</p>
  </section>
  <footer>
    <p>
      Made 2 September 2015<br>
      by myself.
    </p>
  </footer>
</body>
</html>
```



- [Home page](#)
- [Musings](#)
- [My town](#)
- [Links](#)

My first styled page

Welcome to my styled page!

It lacks images, but at least it has style. And it has links, even if they don't go anywhere...

There should be more here, but I don't know what yet.

Made 2 September 2015
by myself.

Ilustración 60: contenido de página HTML5

Vemos que se han utilizado etiquetas semánticas para organizar el contenido. Con estas etiquetas se ve claramente que parte corresponde al menú de navegación, contenido y pie de la página. Aunque posteriormente, demos estilos a la página de forma que, visualmente no se identifiquen dichas partes, un buscador como Google si sería capaz de reconocer dichos elementos sabiendo identificando que sección representa cada uno

A continuación, creamos una hoja de estilo vacía y la añadimos a nuestra página HTML. Entonces empezamos a implementar las hojas de estilo para darle el diseño que ya habíamos previsto.

1. Empezamos por lo más general: tipo, tamaño y color de las fuentes por defecto de la página, color o imagen de fondo, márgenes, etc.
2. Continuamos por los elementos genéricos como los títulos, párrafos, enlaces, tablas, etc.
3. Y finalmente los elementos concretos como la barra de navegación, el footer, etiquetas para resaltar un fragmento de texto, etc.

Ahora veremos como ha quedado nuestra hoja de estilo CSS.


```
<link rel="stylesheet" href="style.css">
```



```
body {  
  padding-left: 11em;  
  font-family: Arial, sans-serif;  
  color: purple;  
  background-color: #d8da3d  
}  
  
nav ul {  
  padding: 0;  
  margin: 0;  
  position: absolute;  
  top: 2em;  
  left: 1em;  
  width: 9em  
}  
  
nav ul {  
  list-style-type: none;  
  padding: 0;  
  margin: 0;  
}  
  
nav ul li {  
  background: white;  
  margin: 0.5em 0;  
  padding: 0.3em;  
  border-right: 1em solid black  
}  
  
nav ul a {  
  text-decoration: none  
}  
  
footer p {  
  font-style: italic;  
  padding-top: 1em;  
  border-top: thin dotted  
}
```

Ilustración 61: añadimos el diseño a nuestra página

Y vemos el resultado final:



Ilustración 62: HTML5 + CSS3

Aunque no lo parezca, ¡Es el mismo documento!

Finalmente añadimos los metas para optimizar el posicionamiento SEO, aunque ya tenemos mucho terreno ganado ya que hemos estructurado correctamente el contenido.