Máster en Full Stack Developer

HTML Y CSS (SASS Preprocesor)

Índice

Tema 1. Introducción al HTML

1.1. Introducción

En este tema aprenderemos conceptos del lenguaje HTML. HTML proviene de Hyper Text Markup Language y no es un lenguaje de programación, sino un lenguaje de marcado que indica a un navegador web el contenido y la estructura de este.

1.2. HTML como lenguaje estructurador de documentos

HTML contiene tanto el contenido como su estructura y jerarquía. Para indicar dicha información a un navegador: esto es una cabecera, esto es un párrafo, esto es un listado, etc. hace uso de etiquetas:

```
<h1>¡Hola Alumnos!</h1>
<p>Esto es un ejemplo de muestra</p>
```

En el anterior ejemplo, las etiquetas <h1> y <p> rodean al contenido e indican al navegador qué clase de contenido es aquello que están rodeando y el navegador u otros programas (como lectores de pantalla) actúan en consecuencia.

Anatomía de un elemento HTML

Al igual que en castellano se puede alterar una frase envolviéndola entre signos de exclamación (¡Hola Alumnos!), HTML funciona de la misma manera: envolviendo contenido con etiquetas de apertura y de cierre.

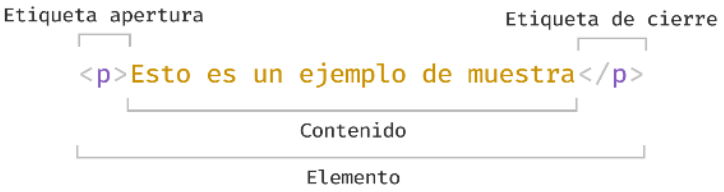

Figura 1. Marcado HTML en detalle. Adaptado de Mozilla.

Las principales partes de un elemento son:

- **Etiqueta de apertura.** Abre el elemento HTML e indica que tipo de elemento va a ser (dependiendo de la etiqueta que se utilice) y desde donde comienza a tener efecto.

- **Contenido.** El contenido que está envuelto entre etiquetas, en el anterior ejemplo: Esto es un ejemplo de muestra.

- **Etiqueta de cierre.** Determina donde acaba un elemento HTML y el contenido deja de estar envuelto, tiene una estructura similar que la etiqueta de apertura, pero se le añade una barra diagonal.

Algunos tipos de elementos HTML no necesitan **etiqueta de cierre** ya que no envuelven contenido como, por ejemplo:

```
  <!-- este elemento muestra una caja para que el usuario introduzca texto,
no necesita contenido -->
  <input>
```

Un elemento HTML puede contener otros elementos HTML, esto es conocido como anidación y es la manera en la que se construyen los documentos HTML:

```
<!-- hay dos párrafos dentro de una etiqueta div -->
<!-- y además, hay una etiqueta strong dentro de un párrafo -->
<div>
  <p>Esto es un párrafo que tiene <strong>un elemento importante</strong><p>
  <p>Esto es otro párrafo dentro de un div</p>
</div>
```

Atributos de un elemento HTML

Los elementos HTML pueden tener atributos y son utilizados para añadir información extra o cambiar su comportamiento.

```
<p class="text">Esto es un ejemplo de muestra</p>
```

El atributo class está añadiendo información extra al párrafo: Está identificándolo para aplicar sobre él estilos CSS. El texto class="text" no se verá en el navegador, permanece oculto y es por así decirlo información extra que el maquetador le da al navegador sobre ese párrafo.

Para escribir HTML es necesario respetar su sintaxis, en el caso de los atributos:

- Los atributos de una etiqueta (tanto clave como valor) tienen que estar separados por espacios.
- La clave y el valor de un atributo debe estar separado por el signo =.
- El valor de un atributo debe estar rodeado de comillas (simples o dobles es cuestión de estilo, lo habitual es comillas dobles).

Existe una excepción a esta sintaxis y es que cuando un atributo coincide con su valor se puede abreviar:

```
<!-- esta etiqueta muestra un vídeo muteado con la etiqueta muted -->
<video muted="muted">
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
</video>

<!-- esta etiqueta muestra exactamente el mismo vídeo muteado -->
<video muted>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
</video>
```

Estructura básica de un documento HTML

Un documento HTML empieza especificando su doctype. Este elemento indica qué tipo de HTML se va a utilizar con el fin de que navegadores o cualquier otro software sepan leer y tratar el marcado que viene a continuación.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Página de ejemplo</title>
  </head>
  <body>
    <h1>Hola mundo</h1>
  </body>
</html>
```

Tras el doctype se inserta la etiqueta de apertura <html>, dentro de la cual, estará todo el documento HTML. Un documento HTML siempre se cierra con una etiqueta de cierre </html> y es la última que debe aparecer en el documento.

Dentro del elemento <html> están las etiquetas head y body.

Etiqueta <head>

Contiene información acerca del documento HTML relacionada con información que no se va a ver en la caja del navegador dedicada a mostrar una página web (viewport) título de la página o metadatos.

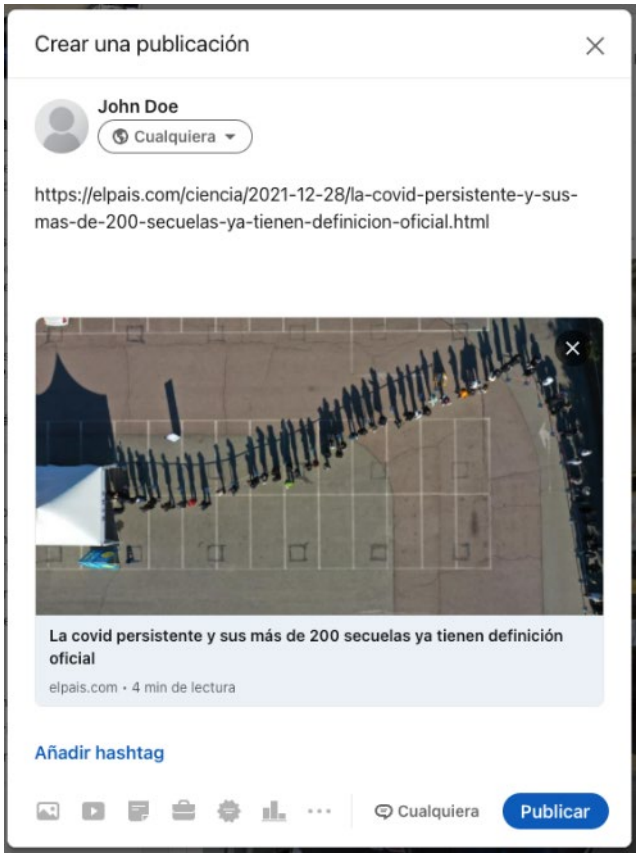

Figura 2. Metadatos que extrae el sitio LinkedIn.com de un documento HTML. Fuente: elaboración propia.

Por ejemplo, los datos que extrae LinkedIn y utiliza para mostrar una caja al compartir un enlace como el título de la noticia y la imagen destacada están especificados con metadatos en el <head>.

Etiqueta <body>

La etiqueta <body> contiene realmente la información que el usuario va a ver del documento HTML, aquella que se va a mostrar en el viewport del navegador.

Dentro de body es donde reside realmente el contenido de la página y donde se maqueta este, en el body estarán las fotografías, texto, tablas, etc. que componen la página.

En el vídeo *Escribiendo un fichero HTML* se muestra un ejemplo de cómo crear un fichero HTML, escribir las primeras etiquetas de contenido y mostrarlo en un navegador.

 Accede al vídeo

1.3. Validación de documentos HTML

Al crear documentos HTML es posible que ocurran errores de marcado o de otro tipo que hagan que el documento no siga la especificación HTML correctamente. Para ello, es útil utilizar un validador de html, como por ejemplo el disponible en: http://validator.w3.org/ que nos asegure que el documento está construido correctamente.

Los navegadores harán lo posible por mostrar un documento HTML, aunque este no sea válido, sin embargo, un documento HTML inválido puede provocar errores no esperados y es un problema a la hora de mantener un sitio web, además de que otros agentes (no solo navegadores de usuarios visitan un sitio web) es posible que no sepan entender un HTML inválido.

1.4. Tipos de documentos (DOCTYPE)

Un documento HTML no es más que un archivo de texto plano con extensión .html, pero a su vez hay versiones de HTML por lo que es difícil para un intérprete saber qué versión utilizar si no se indica de alguna manera.

Para ello está el **doctype** en los documentos HTML. Un documento HTML empieza especificando mediante el doctype el tipo de documento que va a utilizar para que así cualquier intérprete del documento sepa qué normas tiene que seguir.

Actualmente se utiliza un doctype muy sencillo que indica que el documento sigue las *reglas* de HTML5:

```
<!DOCTYPE html>
```

Antiguamente el doctype era más complicado y podía parecerse a esto:

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

La especificación de antiguos doctype se puede consultar en un artículo de Jeffrey Zeldman de 2002 que está disponible en la sección «A fondo».

1.5. Separación de estructura y diseño

Un documento HTML no deja de ser un contenido marcado de determinada manera para ordenarlo, jerarquizarlo y darle sentido. El HTML contiene el qué es lo que hay que mostrar que debe estar separado del cómo mostrarlo.

El cómo se debe mostrar un contenido: tipografías, colores, fondos, etc. forman parte de la presentación del contenido incluido en el HTML, pero **no debe pertenecer a él**.

Antiguamente esta separación no estaba tan clara, y en el mismo documento HTML donde se entrega el contenido (por ejemplo, una tabla) se escribía la manera en la que ese contenido debía representarse (colores, fondos, espaciados, etc.) lo cual acarrea problemas como la baja reutilización de esos estilos, dificultades para mantener la consistencia en el diseño del sitio web, etc.

Actualmente disponemos de las herramientas necesarias para hacer esta separación correctamente: **el contenido es responsabilidad del HTML y la forma de representar ese contenido es responsabilidad del CSS**.

1.6. Introducción a la estructura semántica

HTML no solo puede aportar el contenido de una página web, es capaz de jerarquizarlo y darle una semántica de manera que un intérprete pueda explorar el contenido y entenderlo de una manera sencilla.

Aunque existan muchos tipos de sitios web, todos tienden a compartir elementos comunes que cumplen el mismo cometido y que los usuarios ya reconocemos ya que está dentro de nuestro modelo mental sobre lo que conocemos como una página web.

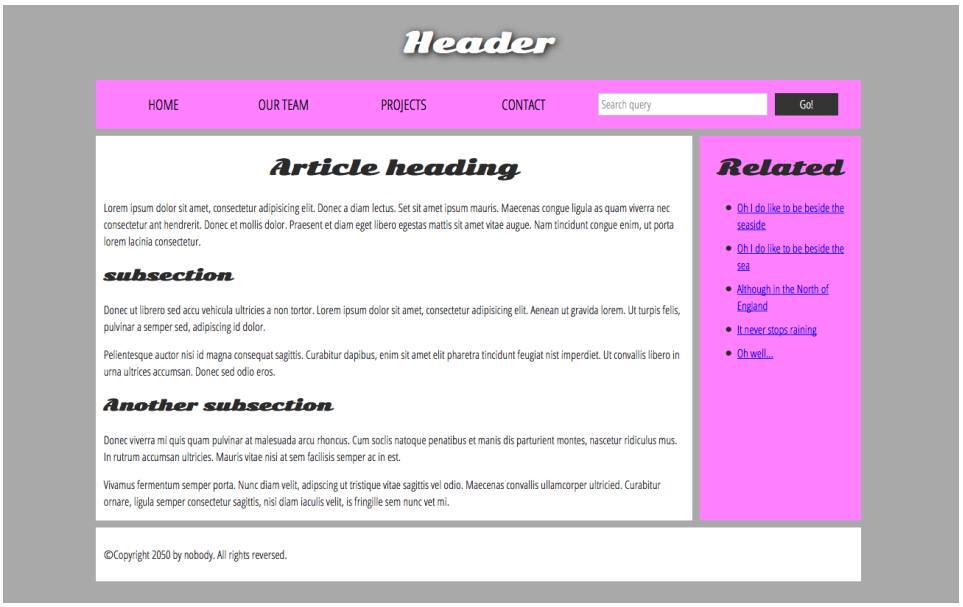

Figura 3. Ejemplo de sitio web. Fuente: Mozilla.

Una típica página web se puede parecer mucho (en concepto) a la que muestra esta imagen. Gracias a las etiquetas semánticas estructurales un intérprete puede conocer con exactitud que partes del contenido corresponden a esas partes ya estandarizadas de un sitio web como la cabecera, contenido principal o pie de página.

Encabezado <header>

Sección de una página que habitualmente aparece en primera posición cruzando de lado a lado con el cometido de ubicar al usuario en el sitio web dándole una introducción al contenido como un logotipo y/o un título de la página.

```
<body>
  <header>
    <h1>Empresa S.L.</h1>
  </header>
</body>
```

Menú de navegación <nav>

Representa un menú de navegación, es decir, dentro de él aparecerán enlaces mediante los cuales el usuario puede ir a partes de documento o hacia otras páginas web.

La etiqueta <nav> puede ser utilizada para menús de navegación independientemente de su importancia en el sitio, ya sea el menú principal u otro menú (o colección de enlaces).

```
  <nav>
    <!-- La etiqueta `ul` y `li` representa una lista desordenada de elementos. -->
    <ul>
      <li><a href="#">Inicio</a></li>
      <li><a href="#">Sobre nosotros</a></li>
      <li><a href="#">Contacto</a></li>
    </ul>
  </nav>
```

Contenido principal <main>

Dentro de la etiqueta <main> está todo el contenido propio de la página, es decir, **el contenido para el que la página ha sido creada**. Es por lo tanto que únicamente habrá una etiqueta <main> en un documento ya que solo puede haber un contenido para el que la página se ha creado, si hay dos, estos dos deben ser envueltos por una etiqueta <main>.

```
<!-- Se recomienda que la etiqueta main sea un hijo directo de body -->
<!-- El siguiente uso estaría recomendado -->
<body>
  <main>
    <p>Contenido principal</p>
  </main>
</body>

<!-- El siguiente uso no estaría recomendado -->
<body>
  <div>
    <main>
      <p>Contenido principal</p>
    </main>
  </div>
</body>
```

Pie de página <footer>

De la misma manera que el <header> de una página da comienzo a esta, el <footer> le pone final. Habitualmente se representa con una franja que recorre de lado a lado la página y muestra contenido como el autor de la página, fecha de publicación o enlaces a otros documentos relacionados.

Además, al igual que la etiqueta <header> es posible encontrar una etiqueta <footer> dentro de secciones de la página para hacer un pie de página para esos contenidos.

1.7. Comprensión y aplicación de las etiquetas semánticas

Las etiquetas semánticas no siempre han estado ahí, hubo un tiempo en el que el contenido se estructuraba únicamente utilizando etiquetas que simplemente

agrupan, pero no dan semántica como el , sin embargo, con la llegada de HTML5 llegaron una serie de etiquetas que además de contener, indican qué es el contenido que contienen. Estas etiquetas llegan para hacer más fácil el trabajo de cualquier máquina que tenga que leer un documento HTML que por la razón que sea no puede apreciar cómo el documento se renderiza en pantalla como lectores de pantalla para personas con deficiencias visuales o robots de motores de búsqueda como Google.

Realmente la semántica es una ayuda a este tipo de herramientas, pero no es ni mucho menos una guía escrita en piedra para que un robot entienda la página es por ello por lo que, será decisión última de dicho robot seguir o no la semántica que el programador ha indicado. Incluso algunas etiquetas han llegado a ser objeto de discusión sobre cómo han de utilizarse en determinados casos, por lo que finalmente se apela a la responsabilidad y sentido común de los desarrolladores al utilizarlas.

1.8. Herramientas de navegadores

Las herramientas para navegadores son utilidades que un desarrollador web utiliza cada día. Con estas herramientas es posible inspeccionar lo que el navegador ha interpretado del documento y qué información tiene en memoria en tiempo real la cual utiliza para mostrar la página.

Lo mejor de todo es que son gratuitas y están integradas en la mayoría de los navegadores por lo que simplemente hay que abrirlas para empezar a utilizarlas:

- **Chrome**: Menú -> Más herramientas -> Herramientas para desarrolladores.
- **Firefox:** Menú -> Más herramientas -> Herramientas para desarrolladores.
- **Safari:** Primero, hay que activarlas desde: Preferencias -> Avanzado -> Mostrar el menú Desarrollo en la barra de menús y una vez activado Desarrollo -> Mostrar inspector web.

Además, en los navegadores suele haber un acceso directo: pulsando botón derecho en cualquier parte de la pantalla y en el menú que aparece se debe seleccionar la opción «Inspeccionar».

Las siguientes son las funciones principales de las herramientas de navegadores.

Inspección de elementos

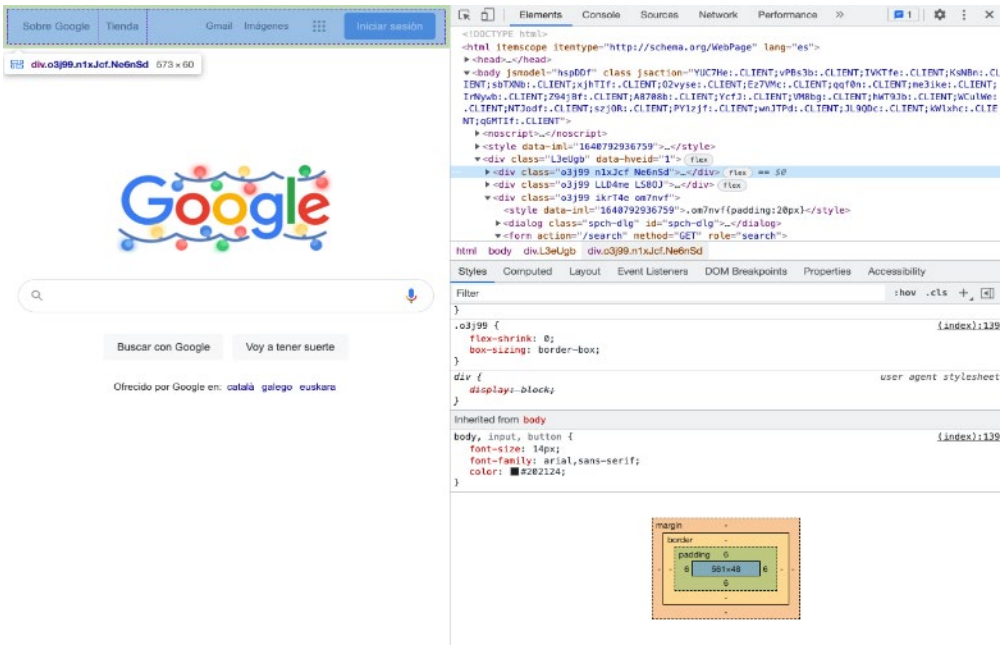

Figura 4. Inspección de elemento en Chrome. Fuente: elaboración propia.

En esta pestaña aparece una representación del documento HTML tal cual como lo ha interpretado el navegador: orden, estructura, atributos CSS y modelo de caja que el navegador ha entendido y realmente está renderizando.

Gracias a la inspección en tiempo real de elementos, se puede comprobar visualmente, gracias a una caja que resalta: el elemento, el espacio que ocupa y su padding/margin (separaciones del elemento respecto de otros elementos).

Consola

En la pestaña de consola se pueden ver mensajes provenientes de la ejecución de JavaScript. Además de HTML y CSS, las páginas web pueden disponer de JavaScript. JavaScript es un lenguaje de programación que se ejecuta en el navegador y como lenguaje de programación dispone de una consola de salida en la que el programador recibe feedback de qué está ocurriendo en la ejecución.

Por otro lado, además de recibir feedback, un usuario puede ejecutar directamente JavaScript escribiéndose en la consola. Este código JavaScript se ejecutará en el navegador para la página web que esté mostrando en ese momento.

Fuentes

Esta pestaña muestra absolutamente todos los ficheros que el navegador ha descargado para mostrar la página web. Es muy útil, por ejemplo, para buscar cadenas de texto en todos los ficheros y localizar así código JavaScript.

Además, posiblemente la función más utilizada de esta pestaña sea la de hacer debug de JavaScript estableciendo puntos de interrupción. Gracias a un punto de interrupción, la ejecución de JavaScript en el navegador se pausa y es posible controlar el flujo de ejecución paso a paso e ir viendo qué hace realmente el navegador: cómo ejecuta el código, qué variables tiene en memoria, cuál es el alcance de estas, etc.

Red

Todo lo que el navegador tenga que recibir/enviar para ver o interactuar con una página web se puede ver en esta pestaña.

Cada una de las peticiones ya sea de ficheros u otro tipo de peticiones como envío de datos crean una fila en el listado de la pestaña red. Pulsando en cada una de esas filas se puede acceder a una información detallada sobre la petición: cabeceras, respuesta del servidor, por qué se ha hecho esa petición, etc.

1.9. Referencias bibliográficas

Mozilla. (s.f.). *Empezar con HTML.* Obtenido de MDN Web Docs: https://developer.mozilla.org/es/docs/Learn/HTML/Introduction_to_HTML/Getting_started#anatom%C3%ADa_de_un_elemento_html

Mozilla. (s.f.). *MDN Web Docs.* Obtenido de Estructura web y documentación: https://developer.mozilla.org/es/docs/Learn/HTML/Introduction_to_HTML/Document_and_website_structure

Tema 2. Etiquetas básicas de estructura

2.1. Introducción

Las etiquetas HTML se pueden dividir en dos grupos: elementos de bloque y elementos de línea.

Esta forma de categorizar elementos HTML corresponde a la manera en la que el elemento se comporta dentro del flujo de HTML y dónde se coloca el elemento respecto de sus hermanos:

- **Elementos de bloque**. Los elementos de bloque generan un bloque dentro del flujo HTML. Se muestran en una línea nueva respecto de su elemento anterior y fuerzan a que el elemento siguiente se muestre en una línea nueva.

- **Elementos de línea**. A diferencia de los elementos de bloque, los elementos en línea ocupan únicamente lo que ocupe el contenido que albergan. Los elementos de línea están deben estar contenidos en elementos de bloque y delimitan piezas pequeñas del contenido del documento.

```
<span>span es un elemento de línea</span>
<strong>strong también</strong>
<p>un párrafo es un elemento de bloque</p>
<span>otro span con un <strong>strong dentro</strong></span>
```

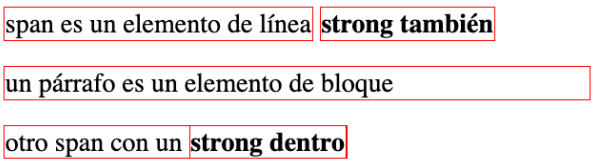

Figura 5. Elementos de bloque y línea. Fuente: elaboración propia.

En este ejemplo se ha resaltado con un borde rojo cada uno de los elementos HTML. Los dos primeros elementos con etiquetas y son elementos de línea, es decir, ocupan lo que tengan que ocupar en función de su contenido y dejan que el siguiente elemento se posicione junto a ellos siguiendo el flujo HTML.

Sin embargo, el párrafo es un elemento de bloque, es decir, aunque su elemento anterior sea de línea, saltará de línea y tenderá a ocupar todo el espacio posible haciendo que su siguiente hermano tenga que situarse en la siguiente línea.

2.2. Elementos de la cabecera <head>

La cabecera de un documento HTML <head> contiene información extra a cerca del documento que no se ve a simple vista por el usuario y que no es realmente el contenido que se puede ver desde un navegador web.

```
<head>
    <meta charset="utf-8">
    <title>Sitio web personal de John Doe</title>
    <meta name="author" content="John Doe">
    <meta name="description" content="En este sitio web encontrarás información sobre mí, mis intereses y una lista de enlaces que me parecen interesantes">
    <link href="favicon.ico" rel="icon" type="image/x-icon" />
  </head>
```

Título del documento

La etiqueta <title> es un metadato que indica el título del documento, no del contenido del documento y para ello se debe utilizar la etiqueta <h1>.

Este título aparece como etiqueta para la página web en la pestaña del navegador y debe proporcionar la información necesaria para que el usuario pueda distinguir un sitio web dentro de otros sitios web abiertos en otras pestañas.

Metadatos

La etiqueta <meta> funciona como un cajón de sastre para cualquier tipo de metadato que se quiera añadir al documento.

Algunos de ellos metadatos están estandarizados como author o description pero cualquier intérprete de páginas web puede crear los suyos propios, por ejemplo, Open Graph protocol (más información en la sección A fondo) es un protocolo creado por Meta que define meta etiquetas propias para obtener información del documento HTML. Es utilizado al compartir un enlace en Facebook (u otras plataformas), de manera que esa información incluida pudiese ser extraída para elaborar un elemento de interfaz adecuada a ese enlace.

```
<head>
    <!-- extracto de la cabecera de una noticia
         en el sitio web elpais.com -->
    <title>Subida del IPC: La inflación se dispara en diciembre al 6,7%, el mayor nivel en casi tres décadas | Economía | EL PAÍS</title>
    <meta name="date" scheme="W3CDTF" content="2021-12-30T08:53:14.221Z">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="lang" content="es">
    <meta name="author" content="José Luis Aranda">
    <meta name="description" content="La escalada de precios no encuentra de momento techo, según el indicador adelantado del INE, que destaca el encarecimiento de la electricidad ">
```

```
    <meta name="organization" content="Ediciones EL PAÍS S.L.">
    <meta property="article:publisher"
content="https://www.facebook.com/elpaiseconomia/">
    <meta property="og:title" content="La inflación se dispara en diciembre
al 6,7%, el mayor nivel en casi tres décadas">
    <meta property="og:description" content="La escalada de precios no
encuentra de momento techo, según el indicador adelantado del INE, que
destaca el encarecimiento de la electricidad ">
    <meta property="og:article:publisher" content="EL PAÍS">
    <meta property="og:article:author" content="José Luis Aranda">
    <meta property="og:updated_time" content="2021-12-30T08:53:14.221Z">
    ...
  </head>
```

Favicon

Esta etiqueta sirve para añadir a un documento HTML un icono, este es el icono que aparece al guardar un sitio web en favoritos o que aparece junto al título en una pestaña del navegador se define con una meta etiqueta.

```
<link href="favicon.ico" rel="icon" type="image/x-icon" />
```

Gracias a esta etiqueta, se le proporciona al intérprete del documento HTML una ruta a seguir (atributo href) donde puede encontrar ese fichero de imagen para mostrar el icono.

2.3. Etiquetas de contenido

Encabezados y párrafos

La separación en bloques del contenido en lugar de que todo un contenido sea texto corrido ayuda tanto al autor en su redacción como al lector al tener una separación clara en piezas de contenido sabiendo dónde empieza una y termina otra de manera

sencilla. Una ayuda extra es que además estas piezas de contenido estén introducidas por titulares o encabezados que permitan al lector anticiparse y tener una idea de qué se va a encontrar en ese contenido.

Cualquier texto, ya sean estos contenidos, artículos de periódico, manuales de instrucciones, etc., siguen este criterio.

Encabezados

HTML dispone de varias etiquetas para maquetar encabezados dependiendo de cuál sea su jerarquía, un documento debe tener un único encabezado principal y de él cuelgan otros encabezados de nivel 2, si un contenido con un encabezado 2 necesita otro encabezado, ese encabezado será de nivel 3, y así hasta llegar al encabezado de nivel 6. Si bien es cierto que en la práctica es muy complicado ver encabezados de nivel 5 y 6.

```
<h1>Título del contenido</h1>
<h2>Subtítulo de nivel 2</h2>
<h3>Subtítulo de nivel 2</h3>
<h4>Subtítulo de nivel 4</h4>
<h5>Subtítulo de nivel 5</h5>
<h6>Subtítulo de nivel 6</h6>
```

Párrafos

Una vez un contenido tiene un encabezado, es interesante separarlo en bloques que estén separados entre sí con el fin de facilitar la lectura y comprensión del contenido. HTML dispone de la etiqueta <p> para ello:

```
<!-- Extracto del libro Don Quijote de la Mancha -->
<p>La del alba sería cuando don Quijote salió de la venta, tan contento, tan gallardo, tan alborozado por verse ya armado caballero, que el gozo le reventaba por las cinchas del caballo. Mas, [...] que parecía que no ponía los pies en el suelo.</p>
<p>No había andado mucho, cuando le pareció que a su diestra mano, de la espesura de un bosque que allí estaba, salían unas voces delicadas, como de persona que se quejaba; y apenas las hubo oído, cuando dijo:</p>
```

Énfasis e importancia

Para dar énfasis en una frase acentuamos determinadas palabras para que el receptor de la comunicación entienda que esa palabra es diferente al resto por el motivo que sea.

HTML tiene un marcado para ello: que la mayoría de los navegadores muestran con la tipografía en itálica, pero esto es una decisión del navegador y no se debe utilizar esta etiqueta para mostrar textos en itálica.

```
No hay nada como las rebajas para comprar ropa a <em>buen precio</em>
```

De la misma manera, pero cuando la importancia de lo que se desea acentuar es mayor, se utiliza la etiqueta que indica que esa parte del texto realmente es más importante que lo que tiene alrededor. Los navegadores suelen mostrar estos elementos con letra negrita.

```
No me gusta comprarme ropa, pero por las rebajas, <strong>Enero es el único mes</strong> en el que gasto dinero en ropa.
```

Listas

Las listas se utilizan para organizar información que habitualmente es de la misma naturaleza por lo que tienen algún tipo de conexión entre sí. El orden en el que

aparecen los elementos de una lista puede o no tener relevancia por lo que existen: **listas desordenadas y listas ordenadas.**

```
<!-- Lista desordenada, el orden de los elementos no es relevante -->
<p>Comunidades autónomas de España</p>
<ul>
  <li>Andalucía</li>
  <li>Aragón</li>
  <li>Principado de Asturias</li>
  ...
</ul>

<!-- Lista ordenada, el orden de los elementos es relevante -->
<p>Cómo cocer un huevo</p>
<ol>
  <li>Llevar a ebullición agua.</li>
  <li>Una vez esté hirviendo el agua, introducir el huevo y esperar entre 11 y 12 minutos.</li>
  <li>Añade una cucharada de sal para que luego se pele más fácilmente.</li>
  <li>Pasado ese tiempo, saca los huevos de la cazuela y déjalos enfriar o refréscalos bajo el chorro de agua fría.</li>
</ol>
```

Tanto las listas desordenadas <ul> como las listas ordenadas <ol> tienen en su interior un listado de elementos recogidos por etiquetas <li>. Habitualmente los navegadores añaden un punto al inicio de cada elemento para las listas desordenadas y un número consecutivo para las listas ordenadas.

También es posible tener listas anidadas, una lista que esté anidada debe estar dentro de un elemento <li> de otra lista, entre elementos <li> no puede haber ningún otro elemento.

```
  <!-- Este índice de contenidos es correcto -->
  <ul>
    <li>Inicio</li>
    <li>Sobre nosotros</li>
    <li>Servicios
      <ul>
        <li>Fontanería</li>
        <li>Pintura</li>
        <li>Mantenimiento de piscinas</li>
      </ul>
    </li>
    <li>Contacto</li>
  </ul>
  <!-- Este índice de contenidos es incorrecto, una etiqueta ul solo puede tener como hijos elementos html con etiquetas li -->
  <ul>
    <li>Inicio</li>
    <li>Sobre nosotros</li>
    <li>Servicios</li>
    <ul>
      <li>Fontanería</li>
      <li>Pintura</li>
      <li>Mantenimiento de piscinas</li>
    </ul>
    <li>Contacto</li>
  </ul>
```

Tablas

Una tabla es un conjunto de datos estructurados en filas y columnas. Precisamente esa estructura da sentido a los datos y ayuda a un usuario a entender y consumir dichos datos.

Antiguamente fueron muy populares a la hora de maquetar sitios web enteros (y no solo para mostrar datos) ya que la maquetación de otra manera era muy difícil. Actualmente existen numerosas técnicas CSS para maquetar correctamente un sitio

por lo que es desaconsejado utilizar tablas para maquetar y solo se deben utilizar para mostrar datos y ayudar al usuario a entenderlos y consumirlos.

ID	Name	Email	Address
1	Dora Stevens	billie.fox@msn.com	1218 Easy Alley
2	Anne Wright	albert.campbell@yahoo.com	807 Umber Dale
3	Cathy Payne	daryl.dean@live.com	850 Cinder Berry Turnabout

Figura 6. Tabla HTML. Fuente: elaboración propia.

```
<table>
  <tr>
    <th>ID</th>
    <th>Name</th>
    <th>Email</th>
    <th>Address</th>
  </tr>
  <tr>
    <td>1</td>
    <td>Dora Stevens</td>
    <td>billie.fox@msn.com</td>
    <td>1218 Easy Alley</td>
  </tr>
  ...
</table>
```

Etiquetas utilizadas que son comunes en tablas:

- <table>: Contiene la tabla.
- <tr>: Alberga cada una de las filas de una tabla.
- <td>: Alberga una celda de una tabla.
- <th>: Caso especial de celda que alberga una cabecera.

Imágenes

Gracias a la etiqueta <img> se pueden incrustar imágenes en un documento HTML. Es una etiqueta que no necesita cierre ya que no necesita contenido para que funcione, simplemente necesita atributos:

```
<img src="abeja.jpg">
```

Esta etiqueta muestra una imagen, mediante el atributo src se enlaza desde el documento HTML y debe estar disponible desde la misma carpeta que el documento para que pueda visualizarse.

HTML cuida la semántica de su contenido (y la etiqueta <img> no iba a ser una excepción) por lo que al añadir una imagen al documento HTML se pierde parte de ese contenido para un intérprete de código, ya que no es posible saber qué contenido muestra la imagen (suponiendo que no se usa inteligencia artificial), por eso existe y **es recomendable el atributo** alt. Este atributo añade a la imagen una breve descripción para que intérpretes puedan actuar en consecuencia (por ejemplo, un lector de pantalla para personas invidentes puede dar una descripción de la imagen).

```
<img src="abeja.jpg" alt="La abeja de la miel poliniza la flor de la rabina">
```

Si la imagen es ornamental, es decir, incluir la imagen responde a cuestiones de diseño del sitio y no al contenido, la etiqueta alt debe ser incluida pero vacía:

```
<img src="fondo-degradado.jpg" alt="">
```

Enlaces: relativos, absolutos y anclas

La etiqueta <a> sirve para crear hipervínculos, es decir, enlaces hacia un punto específico de un documento o hacia otro documento. El contenido del hipervínculo (aquello que esté entre la etiqueta de apertura y la de cierre) se convertirá en un elemento con el que el usuario puede interactuar para navegar.

```
<!-- Enlace dentro de texto -->
<p>
  <!--
  el texto  World Wide web tendrá un enlace hacia la página de wikipedia donde se amplia información
  -->
  Los hipervínculos son parte fundamental de la arquitectura de la <a href="https://es.wikipedia.org/wiki/World_Wide_Web">World Wide Web</a>
</p>

<!-- Enlace en una imagen -->
<a href="https://empresa.com">
  <img src="logotipo.png" alt="Empresa S.L.">
</a>
```

Un hipervínculo acepta los siguientes atributos:

- **href:** Gracias a este atributo se especifica la ruta a la que debe llevar el hipervínculo.
- **title:** Similar al atributo alt en imágenes, pero utilizado en enlaces, gracias al atributo title se añade información adicional sobre lo que se está enlazando.
- **target:** Indica el contexto de navegación que se va a utilizar para mostrar el enlace, actualmente prácticamente en desuso exceptuando el target="blank" que indica que el contexto de navegación es otra ventana, por lo que al pulsar el enlace se abrirá otra pestaña del navegador con la dirección que esté indicada en el atributo href.

Enlaces relativos y absolutos

Un enlace puede apuntar (atributo href) hacia un documento del mismo sitio web o hacia otro documento alojado en otro sitio.

- **Enlace absoluto**: Un enlace absoluto hace que la navegación se dirija hacia otro sitio, por ejemplo, con un atributo que empiece por **http://** la navegación salta de un sitio web a otro sitio web.
- **Enlace relativo**: Un enlace relativo hace referencia a un documento relativo al sitio web que se está visitando, por así decirlo, la navegación se produce, pero dentro del mismo sitio web.

```
  <!-- Enlace relativo, la navegación se mantiene dentro del mismo sitio --
>
  <a href="contacto.html">Contacto</a>

  <!-- Enlace absoluto, la navegación salta hacia otro lugar -->
  <a href="https://google.com">Google</a>

  <!--
    Tipo de enlace especial para enviar correos electrónicos,
    también es un enlace absoluto
  -->
  <a href="mailto://info@correo.com">info@correo.com</a>
```

Anclas

Un ancla es un enlace que se produce dentro del mismo documento, es decir, es un enlace que salta a una parte específica del contenido.

Algunos sitios web implementan la función «volver al inicio», es decir, tienen un enlace de tipo ancla apuntando hacia el inicio del documento, de manera que el usuario al pulsar sobre él devuelve la navegación (siempre dentro del mismo documento) al inicio.

Un enlace de ancla tiene dos partes, el enlace que utiliza la etiqueta <a> y el elemento hacia el que apuntan (cualquier etiqueta HTML con un identificador único o atributo id):

```
<!-- botón con un ancla -->
<a href="#contenido">Saltar al contenido</a>
<header>
  <h1>Empresa S.L.</h1>
</header>
<!-- etiqueta específica a la que apunta el ancla -->
<main id="contenido">
  <p>Bienvenidos al sitio web de Empresa S.L. ...</p>
</main>
```

Marcado de contenido

Las siguientes etiquetas no añaden funcionalidad ni semántica al documento HTML, sirven para marcar determinado contenido con el fin de agruparlo y que varios elementos pertenezcan a un mismo padre (con diferentes propósitos).

La etiqueta sirve para agrupar contenidos siendo esta un elemento de bloque, mientras que la etiqueta cumple el mismo cometido, pero es un elemento de línea.

```
<div>
  <h2>Marcado de contenido</h2>
  <p>Las siguientes etiquetas no añaden funcionalidad ni semántica al
<span>documento HTML</span> ...</p>
</div>
```

2.4. Etiquetado semántico: etiquetas de contenido y su importancia

<section>

La etiqueta <section> se utiliza para agrupar un contenido con un tema común, es decir, cada una de las partes que conforma un documento HTML.

Por ejemplo, en una portada de un sitio de noticias, un elemento con etiqueta <section> podría ser la agrupación de noticias internacionales, otro <section> correspondería a la agrupación de noticias de economía y otra <section> podría ser la parte en la que habla sobre la suscripción al periódico.

<article>

La etiqueta <acticle> contiene una unidad de contenido, es decir, contenido que tiene entidad propia por sí mismo y responde a un tema concreto.

Por ejemplo, en un sitio de noticias se utiliza un elemento <article> para cada una de las noticias, es decir, una noticia tiene entidad propia y responde a un tema en concreto que es precisamente la noticia.

Es tal la entidad propia que puede tener un <article> que puede ser considerado por así decirlo como un documento en sí, de manera que puede contener etiqueta <header> y etiqueta <footer>.

```
<article>
  <header>
    <h1>La infracción se dispara en diciembre...</h1>
    <p>Publicado por: John Doe</p>
    <p>Madrid</p>
```

```
    </header>
    <p>En un año que se recordará por el retorno a tasas de inflación que
no se veían en mucho tiempo, diciembre no podía...</p>
    <footer>
      <time>09:09, 30 diciembre 2021</time>
      <p>123 comentarios</p>
    </footer>
  </article>
```

En el vídeo *Introducción a etiquetado semántico HTML5* se recrea utilizando este tipo de marcado una página web real que contiene principalmente un <article> y además se utilizan otro tipo de etiquetas semánticas.

 Accede al vídeo

<aside>

La etiqueta aside es utilizada para contener todo aquel contenido que debe estar en el sitio, pero es menos importante o incluso puede no estar relacionado con el contenido principal. Su nombre proviene de la posición que suele ocupar (a un lado del contenido principal) pero no tiene por qué ocupar esa posición.

Por ejemplo, en la página de una noticia, la etiqueta <aside> recoge el sidebar que acompaña a la noticia donde se ubica publicidad, una caja de suscripción a newsletter, etc.

<figure>

Etiqueta que muestra una unidad de contenido, pero de manera visual, generalmente una imagen, diagrama, ilustración etc. Se parece a la etiqueta <article> en el sentido de que es auto-contenido. Lo habitual es que ese contenido vaya acompañado de un título que lo describa dentro de la etiqueta <figcaption>.

```
<figure>
  <img src="abeja.jpg"
       alt="La abeja de la miel poliniza la flor de la rabina">
  <figcaption>Abeja polinizando una flor</figcaption>
</figure>
```

<address>

Gracias a <address> se puede dar semántica a un conjunto de datos que indiquen información de contacto. El tipo de datos que contiene la etiqueta <address> puede ser una dirección, un número de teléfono, enlaces a redes sociales, etc.

```
<address>
  Email de contacto: <a href="mailto:contacto@example.com">Empresa S.L.</a>
  Teléfono: 915 123 456
  Nos puedes encontrar en Calle de Almansa, 101, 28040 Madrid
</address>
```

<audio> y <video>

Las etiquetas <audio> y <video> no solo proveen la semántica de que lo que contienen es una pieza audiovisual, sino que también, gracias a los navegadores, hacen que el desarrollador no tenga que preocuparse por programar un reproductor, si no que directamente utilizando esta etiqueta ya aparece un reproductor de manera nativa (es el navegador quien se encarga de ello).

```
<video width="320" height="240" autoplay>
    <source src="movie.mp4" type="video/mp4">
    <source src="movie.ogg" type="video/ogg">
</video>
```

Se pueden utilizar los siguientes atributos:

- **controls:** Muestra los controles para que el usuario pueda iniciar el vídeo, pausarlo, controlar el volumen, etc.
- **autoplay:** Añadiendo este atributo el audio o vídeo se reproducirá automáticamente.
- **muted:** Si la etiqueta tiene este atributo por defecto se muteará el sonido (el usuario siempre puede cambiarlo desde los controles).

Tema 3. Formularios

3.1. Introducción

Gracias a los formularios un usuario de un sitio web ya no solo consume información si no que la puede enviar al servidor.

Un formulario HTML sirve para recolectar información que el usuario introduce en un sitio web (como texto, ficheros, fechas, etc.) validarla, procesarla y enviarla a un servidor web con el fin de guardarla o realizar acciones en función de los datos y una vez realizado eso, que el usuario obtenga un feedback.

Figura 7. Ejemplo de formulario. Fuente: elaboración propia.

Por ejemplo, la sección de contacto de un sitio web contiene un formulario que funciona de la siguiente forma:

- **El documento HTML recoge datos** que el usuario introduce como una dirección de correo electrónico y un campo de texto libre.
- **El formulario valida los campos** (controles de formulario) antes de enviar, la dirección de correo electrónico debe ser válida y el campo de texto libre debe contener al menos un carácter.
- **El usuario pulsa el botón de enviar** y el navegador envía los campos recogidos por el usuario al servidor.
- **El servidor recibe los datos**, guardándolos en una base de datos para su posterior consulta y enviando un correo electrónico al usuario y al responsable del sitio web avisando de que se ha enviado un mensaje nuevo.
- **El servidor responde al navegador** y esta muestra un mensaje de que se ha enviado correctamente el formulario de contacto.

3.2. Estructura básica de los formularios

```
<form action="/contacto.php" method="post">
  <div>
    <label for="name">Nombre y apellidos</label>
    <input type="text" id="name" name="name">
  </div>
  <div>
    <label for="email">Email</label>
    <input type="email" id="email" name="email">
  </div>
  <div>
    <label for="refer">¿Cómo nos has conocido?</label>
    <select id="refer" name="refer">
      <option value="email">Email</option>
      <option value="instagram">Instagram</option>
    </select>
```

```
    </div>
    <div>
      <label for="msg">Message:</label>
      <textarea id="msg" name="user_message"></textarea>
    </div>
    <button type="submit">
      Enviar
    </button>
  </form>
```

Todo formulario debe comenzar y terminar con la etiqueta <form>. Dentro de ella se añaden los controles de formulario para que el usuario introduzca la información, además, puede contener textos, imágenes, etc. siempre que sean relevantes para la introducción de texto del usuario.

En el vídeo *Maqueta de un formulario en HTML* se escribe el código HTML necesario para mostrar en pantalla y enviar datos a un servidor, utilizando para ello la etiqueta <form>, además de muchas otras.

3.3. Controles de formulario

<label>

La etiqueta <label> da título a un control de formulario de manera que el usuario pueda entender qué información debe introducir.

Los controles y los <label> deben estar estrechamente ligados, de manera que no debe haber 2 controles con el mismo <label> y cada control debe tener un <label>.

Esta asociación se realiza poniendo el mismo valor para el atributo for en el <label> y el atributo id en el control de formulario:

```
<!--
    el label tiene el valor "name" en el atributo for
    y hace que esté relacionado con el control de formulario id="name"
  -->
  <label for="name">Nombre y apellidos</label>
  <input type="text" id="name" name="name">
  <!-- otro ejemplo para campo de email ligado con su label -->
  <label for="email">Email</label>
  <input type="email" id="email" name="email">
```

<input>

La etiqueta <input> es el control de formulario más común, hay numerosos tipos y HTML5 introdujo más todavía.

Text

Muestra una caja para introducir texto en una única línea, útil para por ejemplo recoger el nombre de un usuario, su dirección o DNI.

Es el input por defecto, por lo que quitando el atributo type="text" también se mostrará un control de tipo texto, pero es recomendable incluirlo.

```
  <input type="text">
```

Password

Un input de tipo password es muy similar a un input de tipo text pero diseñado para introducir contraseñas. Cada carácter que introduzca el usuario en lugar de mostrarse se mostrará oculto para evitar que otros usuarios puedan ver la contraseña que se ha introducido.

```
<input type="password">
```

Hidden

Un campo de tipo hidden no aparece visualmente en el formulario y por lo tanto el usuario no puede interactuar con él, sin embargo, este campo llegará al servidor como un campo más.

Dejar campos ocultos en el formulario es útil ya que es información que va incluida en el envío del formulario donde un maquetador puede dejar información, pero que el usuario no ve, no controla y no puede cambiar.

```
<!--
el campo "campaign" llegará al servidor sin que el usuario haya podido interactuar con él. En este ejemplo gracias a este campo podemos saber que ese formulario enviado se ha realizado a través de una campaña de instagram (id de la campaña instagram_102021)
-->
<input type="hidden" name="campaing" value="instagram_102021">
```

Checkbox/radio

Un input de tipo checkbox o radio mostrará un control para seleccionar, con solo dos estados posibles: si/no. Es posible dejar marcado por defecto este control añadiendo el atributo checked.

- **Checkbox:** Los navegadores muestran este control con una caja vacía si no hay selección y una caja con una marca de verificación ✓ si se ha realizado la selección. La selección de checkbox es independiente del resto de checkboxes que haya en el formulario.
- **Radio:** Los navegadores muestran este control con una circunferencia si no hay selección y un círculo dentro de otro círculo si se ha realizado una selección. Habitualmente los inputs de tipo radio se utilizan para selecciones únicas dentro de un grupo, es decir, dentro de un listado de inputs de tipo radio solo puede

haber un seleccionado. Este comportamiento se consigue utilizando un identificador único para el atributo name.

Es posible dejar marcado por defecto cualquiera de los dos controles añadiendo el atributo checked.

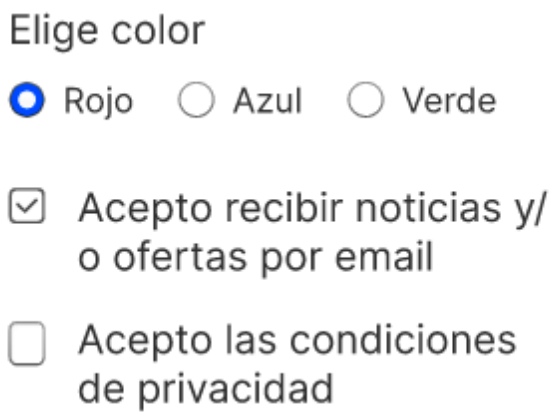

Figura 8. Ejemplo de campos input checkbox y radio. Fuente: elaboración propia.

```
<!--
  gracias al atributo name="color" en todos los radios solo podrá haber uno seleccionado al mismo tiempo
  -->
  <fieldset>
    <legend>Elige color</legend>
    <ul>
      <li>
        <label for="color-red">Rojo</label>
        <input type="radio" id="color-red" name="color" value="red" checked>
      </li>
      <li>
        <label for="color-blue">Azul</label>
        <input type="radio" id="peas" name="color" value="blue">
      </li>
      <li>
        <label for="color-green">Verde</label>
        <input type="radio" id="color-green" name="color" value="green">
      </li>
    </ul>
```

```
  </fieldset>
  <!-- el primer checkbox aparecerá seleccionado por defecto -->
  <label  for="privacy-email">Acepto  recibir  noticias  y/o  ofertas  por
email</label>
  <input type="checkbox" id="privacy-email" checked name="privacy-email">
  <label for="privacy">Acepto las condiciones legales</label>
  <input type="checkbox" id="privacy" name="privacy">
```

File

El input de tipo *file* muestra un control para que el usuario pueda adjuntar ficheros al formulario y poder enviarlos así al servidor.

Es posible usar los siguientes atributos:

- **accept**: Sirve para indicar que tipo de archivo acepta el control, por ejemplo, se podría indicar que solo se permite la subida de imágenes.
- **multiple**: Atributo booleano que sirve para indicar que se pueden subir más de un archivo a la vez.

```
  <!-- gracias al atributo accept se limita la extensión del fichero --
  <div>
    <label for="cv">Añade tu currículum</label>
    <input type="file" id="cv" name="cv" accept=".pdf,.doc">
  </div>
  <!-- gracias al atributo multiple se pueden subir una o varias fotos -->
  <div>
    <label for="images">Añade una o varias fotos tuyas</label>
    <input type="file" id="images" name="images" accept="image/*" multiple>
  </div>
```

Textarea

Un textarea es un control de formulario pensado para que el usuario introduzca textos de más de una línea, es decir, textos largos como un comentario en una red social.

```
<label for="comment">Añade un comentario:</label>
  <textarea id="comment" name="comment"></textarea>
```

Select

El control de formulario select muestra un desplegable con una lista acotada de opciones para que el usuario elija entre ellas, es decir, el usuario no introduce información, si no que selecciona una opción de entre las que se le presentan.

Además de la etiqueta select que engloba a todo el control, dentro de este se debe escribir cada una de las opciones con la etiqueta option.

```
  <label for="referred">¿Cómo nos conociste?</label>
  <select name="referred" id="referred">
    <option value="social">Redes sociales</option>
    <option value="friend">Un amigo / conocido</option>
    <option value="search">Motores de búsqueda</option>
    <option value="forum">A través de un foro</option>
    <option value="unknown">Otro</option>
  </select>
```

Button

El control button puede ser utilizado en cualquier parte del documento HTML (no solo en formularios) y sirve como control para que se ejecute una acción cuando el usuario pulsa sobre él.

Su uso dentro de formularios viene dado al utilizar el atributo type el cual acepta los siguientes valores:

- **submit:** Esto convierte al botón en el control que sirve para enviar el formulario al servidor, es decir, una vez ha rellenado el usuario el formulario, pulsando un botón de tipo submit toda esa información se mandará al servidor.
- **reset:** Un botón de tipo reset borrará toda la información introducida por el usuario en un formulario.

Fieldset y legend

Fieldset *y* legend no son controles como tal, están más relacionados con la semántica de los formularios y sirven como agrupadores de controles.

Todos los controles de formulario que estén dentro de una etiqueta fieldset deben tener el mismo propósito y la etiqueta legend describe ese propósito común.

```
<fieldset>
  <legend>Datos de envío</legend>
  <div>
    <label for="shipping-dir1">Dirección</label>
    <input type="text" name="shipping-dir1" id="shipping-dir1">
  </div>
  <div>
    <label for="shipping-phone">Teléfono</label>
    <input type="text" name="shipping-phone" id="shipping-phone">
  </div>
</fieldset>
<fieldset>
  <legend>Datos de facturación</legend>
  <div>
    <label for="billing-dir1">Dirección</label>
    <input type="text" name="billing-dir1" id="billing-dir1">
  </div>
  <div>
```

```
    <label for="shipping-phone">Teléfono</label>
    <input type="text" name="billing-phone" id="billing-phone">
  </div>
</fieldset>
```

3.4. Nuevos inputs y funcionalidades HTML5

Nuevos tipos de input

HTML5 trajo consigo la simplificación de muchas tareas comunes de los programadores web. Gracias a estos nuevos tipos de inputs, el trabajo de desarrollo de controles avanzados como la introducción de fechas o validación de campos desaparece y es labor del navegador adaptarse a la naturaleza del campo que el usuario está introduciendo.

- **search:** Campo de texto para hacer búsquedas.
- **url:** Campo de texto para introducir urls.
- **tel:** Campo de texto para introducir números de teléfono.
- **email:** Campo de texto que solo acepta direcciones de correo válidas.
- **number:** Campo de texto donde solo se pueden introducir números. Algunos navegadores añaden dos controles en la parte de la derecha para aumentar o disminuir en 1 el número introducido.
- **tel:** Campos de texto para introducir números de teléfono.
- **date:** Campo de texto en el que al pulsar sobre él aparece un selector de fecha (año, mes y día).
- **datetime:** Campos de texto en el que al pulsar sobre él aparece un selector de fecha (año, mes y día) y hora.
- **range:** Control de formulario para seleccionar un dato dentro de un rango (desde el punto de vista de usabilidad, al usuario le va a resultar difícil dar con el elemento exacto por lo que hacerlo, no debe ser relevante). Gracias a los atributos min y max se puede definir ese rango y al atributo step la diferencia entre un elemento

y otro de la selección: Por ejemplo, para elegir un número del 1 al 10: min = 1 max=10 step=1

- **datalist:** Un datalist no aparece en el documento como tal, sino que se utiliza junto con un input para dar una lista de opciones sugeridas. Es similar a la etiqueta select pero no limita la selección de entre las opciones, si no que el usuario puede seguir introduciendo la opción que crea conveniente.

```
<!-- ejemplo de datalist -->
<label for="country">Selecciona un país:</label>
<input list="browsers" name="country" id="country" />
<datalist id="browsers">
  <option value="España">
  <option value="Portugal">
  <option value="Francia">
  <option value="Alemania">
</datalist>
```

Validación de formularios

La validación de formularios en el lado del cliente (es decir, en el caso de formularios HTML en el navegador del usuario) es una buena práctica para mejorar la experiencia del usuario. Es el propio navegador quien le puede alertar de inmediato al usuario de si el campo está introducido es correcto y/o es lo que debe rellenar.

HTML5 trajo consigo herramientas sencillas de implementar con el fin de validar formularios del lado del cliente, por ejemplo, en un formulario de contacto donde el campo de email es requerido (si no, es imposible ponerse en contacto de vuelta con el usuario) añadiendo unos simples atributos el navegador no enviará el formulario hasta que no se haya introducido una dirección de email correcta (una dirección que tenga el carácter @ en medio y el dominio sea correcto).

Los atributos que se pueden añadir a un control de formulario relacionados con su validación son los siguientes:

- **required**: Indica si el campo es requerido o no. No se podrá enviar el formulario si un campo marcado con este atributo está vacío.
- **pattern**: El maquetador puede especificar una expresión regular (más información en la sección A Fondo) y el campo introducido debe cumplirla para considerarse válido. Por ejemplo, un número de DNI debe tener una letra al final.
- **minlength**, **maxlength**: Longitud mínima/máxima de cadenas de texto.
- **min**, **max**: Mínimo y máximo para números.

```
  <form>
    <!-- este campo solo será válido si contiene un email correcto y no está vacío -->
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>
    <!-- este campo solo será válido si contiene un dni con formato español correcto y no está vacío -->
    <label for="dni">DNI:</label>
    <input type="text" id="dni" name="dni" pattern="/^[0-9]{8,8}[A-Za-z]$/" required>
  </form>
```

3.5. El proceso de envío y recepción de datos

Un formulario en un documento HTML no sirve de mucho si esos datos no se envían a un servidor. HTML no es un lenguaje de programación si no un lenguaje de marcado por lo que solo se encarga de recoger los datos, en enviarlos y procesarlos se encarga el navegador y el servidor.

Figura 9. Esquema Servidor / Cliente. Fuente: elaboración propia.

Tanto pedir el documento HTML inicial, las imágenes, hojas de estilo, etc. como enviar un formulario, todo se hace con esta arquitectura: el cliente solicita algo al servidor y este le responde con aquello que se le ha pedido.

Recogida de datos en HTML: atributo name

Todos los controles de formulario en HTML aceptan el atributo name, gracias a ese atributo podemos asignarle una clave a ese control para poder recoger posteriormente ese dato dentro del formulario, por ejemplo, desde el lado del servidor.

```
<form>
  <input name="address">
  <input name="vat-number">
  <select name="country">
    <option value="spain">España</option>
    <option value="unknown">Otros</option>
  </select>
  <input type="checkbox" name="privacy">
</form>
```

Suponiendo que el usuario ha rellenado el anterior formulario con sus datos, ha seleccionado España en el desplegable y ha seleccionado el checkbox, la información del formulario se puede convertir a:

```
{
  "name": "John Doe",
  "vat-number": "12345678A",
  "country": "spain",
  "privacy": "on"
}
```

Es decir, para albergar la información de un formulario se utiliza una estructura de *clave - valor* donde la clave es la que define el maquetador mediante el campo *name* en cada uno de los controles del formulario y el valor es el valor introducido por el usuario.

Los checkbox y radio funcionan de manera especial, si no se selecciona no llegará esa clave/valor al servidor (se ignora directamente) y si se seleccionan y no tienen atributo value el valor será on:

```
<input type="checkbox" name="check1">
<input type="checkbox" name="check2" checked>
<input type="checkbox" name="check3" checked value="accept">
```

Sin intervención del usuario, estos campos de formulario se traducirán a:

```
{
  "check2": "on",
  "check3": "accept"
}
```

Petición al servidor: atributos method y action

La etiqueta HTML form acepta los atributos method y action, ambos están relacionados con la petición que realizará el navegador al servidor para enviar los datos del formulario.

- **Action**: El atributo action define la url a la que el formulario enviará los datos, por así decirlo, es la dirección relativa al servidor que es está esperando los datos del formulario y actuar en consecuencia.
- **Method**: Este atributo define la manera en que se envían los datos al servidor, el protocolo HTTP define diferentes tipos de peticiones: GET, POST, PUT o DELETE (más información en la sección «A Fondo» siendo las dos primeras las más utilizadas y sencillas de entender:

  - GET: La petición es de consulta, es decir, no modifica nada en el servidor y simplemente pide datos. Los datos pueden viajar en la URL o por cabeceras.
  - POST: La petición envía información con el fin de que el servidor la almacene de alguna manera y actúe en consecuencia. Los datos solo viajan a través de cabeceras.

```
<!--
  Formulario de búsqueda, utiliza method get ya que consulta elementos al servidor que cumplen un determinado criterio de búsqueda pero no escribe nada en él
-->
<form action="/search.php" method="get">
  <label for="search-field">Campo de búsqueda:</label>
  <input type="search" name="q" id="search-field">
  <button type="submit">Buscar</button>
</form>

<!--
  Formulario de contacto, utiliza el method post ya que la petición guardará en el servidor la información que recibe y enviará un email al administrador avisando de un nuevo contacto
-->
<form action="contact.php" method="post">
  <label for="name-field">Nombre:</label>
  <input type="text" name="name" id="name-field">
  <label for="email-field">Email:</label>
  <input type="email" name="email" id="email-field">
  <label for="text-field">Comentarios:</label>
```

```
    <textarea id="text-field" name="comment"></textarea>
    <button type="submit">Enviar</button>
  </form>
```

3.6. Buenas prácticas de creación de formularios

Aunque HTML es solo una maqueta, realizarla correctamente influye enormemente en cómo será el código del servidor que reciba esa petición. Por ello, un formulario HTML bien maquetado supone un código más robusto, mejor experiencia de usuario y menos incidencias en un futuro.

Validación de lado del cliente, pero también desde el servidor

La validación de los datos no debe existir únicamente en el cliente, en el caso de formularios HTML: en el documento HTML que interpretará un navegador. Un atacante puede hacer peticiones directamente al servidor sin pasar por el documento HTML y si no hay validación en el servidor los datos no estarían validados de ninguna manera.

Por ejemplo, un formulario de venta de entradas en el que en el lado del cliente se valida que el asiento esté disponible, si un atacante hace la petición directamente al servidor sin pasar por el formulario HTML podría reservar una entrada para un asiento que ya está reservado.

Potencia de nuevos campos HTML5

En la mayoría de los casos de uso de formularios los campos de recogida de datos se parecen entre sí: textos, fechas, ficheros, etc. es muy posible que para cada campo que se necesite recoger haya un campo especializado en la nueva especificación de campos HTML5.

Gracias a utilizar campos específicos para cada tipo de dato, las validaciones, las interfaces de usuario para añadir los datos, etc. se simplifica enormemente ya que se convierte en responsabilidad del navegador y no del programador.

Por ejemplo, para recoger una fecha, el maquetador tiene dos opciones: primera opción utilizar un input de tipo texto y desarrollar mediante JavaScript todo el componente que recoge la fecha en un calendario, que lo valida, etc. o simplemente utilizar: <input type="date"> y dejar que el navegador decida cuál es la mejor forma de hacer que el usuario seleccione una fecha.

Utilización de label

Un control de formulario es simplemente eso, una herramienta para que el usuario introduzca información. Como tal, le falta semántica y le falta que el usuario entienda qué debe introducir en dicho campo, para ello existe la etiqueta label: Para dar título al control.

Es habitual utilizar incorrectamente el atributo placeholder como label de manera que se utiliza un campo que debe servir como ayuda al usuario como título:

Figura 10. Uso correcto de label / input. Fuente: elaboración propia.

```
<!-- uso incorrecto, se utiliza atributo placeholder como label -->
<form>
  <input type="text" placeholder="Nombre y apellidos">
  <input type="email" placeholder="Email">
</form>
```

```
<!-- uso correcto -->
<form>
  <label for="name">Nombre y apellidos</label>
  <input type="text" placeholder="Nombre y apellidos" id="name">
  <label for="email">Email</label>
  <input type="email" placeholder="johndoe@example.com" id="email">
</form>
```

Tema 4. Introducción a CSS

4.1. Introducción

Mientras que el HTML es utilizado para definir el contenido de una página web, su jerarquía y semántica, las hojas de estilo CSS (Cascading Style Sheets) permiten estilar con precisión ese contenido definido en el HTML: Colores, espaciados, tipografías, posición, etc.

4.2. Niveles de aplicación de CSS un documento HTML

Al igual que el marcado HTML, el código CSS es texto plano que llega al navegador de una manera o de otra y que este interpreta. Hay varias maneras de escribir y agregar al documento código CSS:

Archivo CSS

De la misma manera que el documento HTML está escrito en un archivo con extensión .html la mejor manera de escribir código css es hacer lo mismo: en un archivo separado del HTML y con extensión .css. De esta manera no solo es conceptual la disociación entre el contenido y el estilo, sino que además es literal: se encuentran en dos ficheros separados uno de otro.

Para relacionar un documento HTML con una hoja de estilo externa se utiliza la etiqueta <link> dentro de la etiqueta head del documento HTML. El atributo href sirve para hacer referencia al documento archivo CSS.

```
<html>
  <head>
    <!-- añade el archivo styles.css que se encuentra en la carpeta styles
al mismo nivel que el documento HTML -->
    <link rel="stylesheet" href="styles/styles.css">
  </head>
  <body>
    <h1>¡Hola, mundo!</h1>
  </body>
</html>
```

Etiqueta style

Es posible tener CSS sin utilizar un archivo aparte, por así decirlo el archivo CSS está incrustado en el documento HTML. CSS y HTML siguen estando separados conceptualmente, pero se encuentran en el mismo fichero lo cual no es la manera recomendada ya que el código CSS no es reutilizable: en caso de sufrir cambios se debe actualizar en todos los documentos HTML en lugar de en un único lugar (un archivo CSS).

```
<html>
  <head>
    <style>
      h1 {
        color: green;
        font-family: Arial, Helvetica, sans-serif;
      }
    </style>
  </head>
  <body>
    <h1>¡Hola, mundo!</h1>
  </body>
</html>
```

Estilos en línea

Es la manera menos recomendable de escribir CSS, se debe evitar a no ser que realmente haya una razón por la que cualquiera de las otras opciones no es viable. Maqueta HTML y estilo CSS se mezclan gracias al atributo style que puede ser aplicado en cualquier elemento HTML.

```
<html>
  <head>
  </head>
  <body>
    <h1 style="color: green; font-family: Arial, Helvetica, sans-serif">
      ¡Hola, mundo!
    </h1>
  </body>
</html>
```

En el vídeo *Cómo añadir código CSS a un documento HTML* se ponen en práctica estos diferentes métodos para comenzar a escribir código CSS en un documento y poder verlo en un navegador.

 Accede al vídeo

4.3. Sintaxis de CSS

El código CSS es en resumen un conjunto de reglas y dichas reglas definen el estilo de los elementos a los que hacen referencia. Dentro de la regla CSS, hay declaraciones que van definiendo propiedades y a cada se establece un valor.

Figura 11. Esquema de una regla CSS. Fuente: elaboración propia.

4.4. Propiedades y valores

No es habitual que un maquetador (tenga la experiencia que tenga) conozca absolutamente todas las propiedades CSS, por un lado, porque la lista es muy grande y por otro porque evoluciona de manera muy rápida: nuevas propiedades que los navegadores soportan o a las que dejan de dar soporte.

Sin embargo, hay un conjunto de estas propiedades inmensamente utilizadas que conviene conocer:

Texto

Color

La propiedad color establece el color de letra utilizado, por defecto los navegadores muestran el texto de color negro y con esta propiedad se puede modificar ese color:

```
/* Ahora el color de texto de todos los párrafos será rojo */
p {
  color: #e60000
}
```

Font-family

La propiedad font-family define la tipografía que debe utilizarse para renderizar un texto. Se puede definir de dos formas:

- Utilizando el nombre de la familia tipográfica: Arial, Helvetica, etc.
- Utilizando el nombre genérico de una familia tipográfica, es decir no referenciando a ninguna familia en concreto si no a un tipo genérico de ellas:
  - **serif:** Para utilizar una familia tipográfica con serifa.
  - **sans-serif:** Para utilizar una familia tipográfica sin serifa.

De alguna o de otra manera el navegador necesita los archivos fuente de las tipografías para poder renderizar el texto, puede ser que el sistema no disponga de esa tipografía y haya que buscar una alternativa, por ello es común definir un listado de tipografías a modo de fallback*:*

```
/*
Los encabezados h1 se deben mostrar utilizando la familia "Times New Roman", en caso de no poder utilizarse, se debe utilizar la familia Times, y en último caso se debe utilizar una tipografía con serifa
*/
h1 {
  font-family: "Times New Roman", Times, serif;
}
p {
  font-family: Arial, Helvetica, sans-serif;
}
```

Font-size

Define el tamaño de letra que debe utilizar un texto, se pueden utilizar medidas absolutas o medidas relativas:

```
/*
Los encabezados h1 tendrán un tamaño de letra de 30px independientemente de dónde se utilice = medida absoluta
*/
h1 {
  font-size: 30px;
}

/*
Los párrafos tendrán un tamaño de letra 1,5 veces mayor al que tuviesen si no existiese esta regla, como el tamaño final de letra depende de otro elemento es una medida relativa
*/
p {
  font-size: 150%;
}
```

Line-height

Define el alto de la línea de texto, es decir, si por ejemplo un párrafo se muestra a dos líneas, el tamaño vertical total de ese párrafo será su line-height multiplicado por 2. Por legibilidad, es recomendable que el alto de línea sea un poco mayor que el tamaño de fuente.

```
/*
El alto de línea del encabezado h1 será 1.6 veces mayor que
el tamaño de letra
*/
h1 {
  line-height: 1.6;
}
```

```
p {
  line-height: 20px;
}
```

Font-weight

Maneja el peso dentro de la familia tipográfica al que se renderiza un texto. El peso de una tipografía se establece en el momento de definirla.

Los pesos que puede tener una tipografía son: 100, 200, 300, 400, 500, 600, 700, 800, 900 y, además, se puede escribir valores equivalentes: *normal* equivale al peso 400 y *bold* al 700.

```
/*
Los encabezados utilizarán el peso 500 de la familia tipografía que tuviesen definida
*/
h1 {
  font-weight: 500;
}

/*
Los encabezados utilizarán el peso 400 de la familia tipografía que tuviesen definida
*/
p {
  font-weight: normal;
}
```

Text-align

La propiedad align maneja la manera en la que el texto se alinea dentro de la caja que lo contiene, los posibles valores son: left, right, center, justify. Por defecto, los navegadores utilizan left.

```
h1 {
  text-align: center
}

p {
  text-align: justify
}
```

Fondos

Gracias a las propiedades de fondo se puede alterar el fondo de un elemento HTML: se pueden utilizar como fondo colores, imágenes o degradados.

```
/* Color plano */
h1 {
  background-color: green;
}

/* Imagen de fondo */
p {
  background-image: url(fondo.jpg);
}

/* Degradado */
span {
  background-image: linear-gradient(#e60000, #9c0000);
}
```

Además de establecer el fondo, es posible manejar su posición o si se repite o no para cubrir todo el espacio disponible:

```
div {
  /* Se utilizará la imagen fondo.jpg como fondo */
  background-image: url(fondo.jpg);
  /* La imagen no se repetira para cubrir todo el espacio */
  background-repeat: no-repeat;
  /* El fondo empezará a verse a partir de la coordenada 100,100 */
```

```
    background-position: 100px 100px;
  }
  /* El anterior CSS se puede escribir de manera abreviada */
  div {
    background: url(fondo.jpg) no-repeat 100px 100px;
  }
```

Bordes

Todo elemento HTML tiene una caja que lo contiene, a dicha caja se puede añadir un borde.

```
  p {
    border-color: green;
    border-width: 1px;
    border-style: solid;
  }

  /* El anterior CSS se puede escribir de manera abreviada */
  p {
    border: 1px solid green;
  }
```

Tamaños

Por defecto, los elementos en línea ocupan lo que ocupe aquello que contienen (tamaño intrínseco) y los elementos de bloque tienden a ocupar todo el espacio disponible. Es posible forzar este comportamiento y que un elemento tenga un tamaño (alto o ancho) específico.

```
  /* a un elemento en línea lo forzamos para que ocupe todo el ancho
disponible */
  .img {
    width: 100%
  }
```

```
/* a un elemento de bloque se fuerza un ancho y alto específico */
p {
  width: 300px;
  height: 300px;
}
```

width y height establecen el tamaño exacto que debe tener un elemento, hay un paso intermedio a esto en el que el flujo de HTML se mantiene y es estableciendo un tamaño mínimo o máximo con las propiedades min-width, max-width, min-height o max-height.

```
/* Propiedad muy habitual para hacer imágenes flexibles, la imagen cogerá todo el ancho posible pero nunca excediendo su tamaño real (el 100%), es decir, que la imagen será flexible (varía su ancho) pero solo hasta su tamaño real. */
img {
  max-width: 100%
}

/* Este párrafo ocupará como mínimo 200px de alto, si tiene más contenido puede ocupar más */
p {
  min-height: 200px;
}
```

Funciones

Una función es un trozo de código aislado que recibe unos valores de entrada, los procesa y devuelve unos valores de salida.

CSS incluye funciones que realizan determinadas operaciones, por ejemplo:

```
div {
  color: rgb(12, 231, 2) /* #0CE702 */
}
```

rbg() es una función CSS que recibe unos valores de entrada, en este caso está recibiendo 3 valores de entrada: 12, 231 y 2. Para estos valores de entrada revuelve un código HEX de color, en este caso: #0CE702.

Las funciones en CSS se utilizan siempre como valores para propiedades (es decir, detrás de los dos puntos de la definición.

Algunas funciones:

- **calc():** Hace operaciones matemáticas en el navegador, por ejemplo: calc(100% - 20px)
- **rbg():** Genera un color desde rbg.
- **rgba():** Similar a rgb() pero con otro parámetro para la transparencia.
- **max():** A nivel de navegador, devuelve el número más grande de entre la lista de parámetros que reciba.
- **min():** Similar a max() pero para devolver el número más pequeño.

Abreviaturas

Algunas propiedades pueden abreviarse consiguiendo un código CSS más corto.

```
  .caja {
    margin: 10px 20px 30px 40px;
    /*
    La anterior declaración abrevia las siguientes (orden sentido agujas del reloj empezando por top)
    margin-top: 10px;
    margin-right: 20px;
    margin-bottom: 30px;
    margin-left: 40px;
    */
  }
```

Otras propiedades que permiten abreviaturas en su declaración son, por ejemplo: padding, background, border.

4.5. Unidad básica de medida absoluta: píxel

Las medidas absolutas en CSS son utilizadas con unidades fijas y el elemento al que apliquen aparecerá exactamente a ese tamaño.

Hay otras medidas absolutas, pero la mayormente utilizada es el píxel. El píxel al que referencia CSS no debe ser confundido con el píxel del hardware:

- **Píxel hardware**: Unidad mínima que una pantalla puede físicamente mostrar, es decir, una pantalla está compuesta por un número concreto de pixeles.
- **Píxel de referencia en CSS**: Es un tamaño concreto definido por la W3C que tiene que ver con la percepción de tamaño y no con la cantidad de pixeles que se utiliza para ello. Gracias a esto, un píxel CSS en diferentes tamaños de pantalla o diferentes densidades mantiene su tamaño.

Es debido a este cálculo de píxel de referencia que hace CSS por el que no se puede asegurar al 100% el tamaño exacto de elementos HTML en una misma pantalla y ni mucho menos en todo tipo de pantallas en las que podemos ver páginas web: smartphones, tabletas, pantallas de portátil, etc. Por lo que el pixel perfect no es posible.

4.6. Unidades avanzadas de medida relativa: %, em, rem, vw, vh

Las unidades de medida relativas en CSS no están completamente definidas, es decir, necesitan otra medida para definirlas exactamente.

```
/* Medida absoluta al body */
  body {
    font-size: 18px;
  }

  /*
  El párrafo por defecto hereda el font-size: 18px del body, Al aplicar esta regla sobre párrafos, el tamaño final de letra será:
  18 * 1.5 = 27px
  La unidad % es relativa por que necesita otra unidad (el 18px del body) para definirse
  */
  p {
    font-size: 150%;
```

Medidas relativas en CSS:

- **%** Unidad relativa al elemento padre del elemento
- **em** Unidad relativa al tamaño de fuente del elemento. Si un elemento tiene un tamaño de fuente 1.7em su tamaño final de fuente será el tamaño de fuente que tuviese multiplicado por 1.7.
- **rem** Unidad relativa al tamaño de fuente del root (elemento con etiqueta html).
- **vw** Unidad relativa al ancho del viewport. 100vw === ancho del viewport.

vh Similar al vw pero relativa al alto del viewport.

Tema 5. Selectores CSS

5.1. Introducción

Un archivo CSS es un conjunto de reglas que están basadas en dos partes: selector y declaración. El selector indica a quién se le aplica lo que la declaración especifica.

La correcta utilización de selectores es algo que se debe dominar al desarrollar sitios web avanzados, ya que los archivos CSS se tornarán grandes y es complicado gobernar todas esas reglas CSS y un buen uso de selectores puede ahorrar muchos quebraderos de cabeza.

5.2. Selectores

Selector universal

El selector universal selecciona cualquier elemento HTML. Es poco habitual que se necesite seleccionar absolutamente todos los elementos HTML.

```
/*
Ahora todos los elementos tendrán un borde azul
*/
* {
  border: 1px solid blue;
}
```

Selector de elemento

Un selector de elemento se refiere a etiquetas HTML directamente.

```
h1 {
  background-color: green;
}
p {
  color: blue;
}
```

Selectores de clase

A medida que un sitio web crece, los selectores de elemento HTML no son útiles, es poco común que absolutamente todas las etiquetas tengan el mismo estilo en todo el sitio, por ejemplo, solo teniendo selector de elemento nos obliga a que todos los párrafos del sitio tengan el mismo estilo.

Para ello existen las clases, los elementos HTML aceptan el atributo class y mediante el valor de ese atributo se pueden estilar elementos HTML específicos (aquellos que dispongan de ese nombre de clase).

```
  .big {
  font-size: 20px;
}

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>

<!-- este párrafo que incluye la clase big tiene un tamaño de letra de 20px -->
<p class="big">Praesent varius arcu rutrum, malesuada mauris id, ultricies lacus. Vestibulum eu urna libero.</p>
<p>Suspendisse fermentum volutpat dui, vitae molestie magna fringilla ornare.</p>
```

```
  <!-- este también, puede haber varios párrafos con la misma clase -->
  <p class="big">Praesent varius arcu rutrum, malesuada mauris id, ultricies lacus. Vestibulum eu urna libero.</p>

  <!-- incluso otro tipo de etiquetas, también tendrán un tamaño de letra de 20px -->
  <div class="big">Lorem ipsum dolo</div>
```

La utilización de clases en elementos HTML es la manera más habitual de estilar elementos HTML ya que su selección es muy rápida por parte del navegador (al navegador le cuesta poco encontrar elementos que dispongan de determinada clase) y permite un control muy preciso por parte del maquetador ya que simplemente tiene que controlar el atributo class de la etiqueta que quiera estilar.

Identificadores únicos

Los identificadores únicos funcionan de una manera similar al atributo class pero con una excepción, solo puede haber un elemento con ese identificador en el documento.

```
  #header {
    background-color: blue;
  }
```

El maquetador debe asegurar que va a haber un único elemento con ese identificador único en el documento, de no ser así, el documento HTML será inválido. Los identificadores únicos son la manera más rápida de acceder a elementos por parte del navegador, pero deben utilizarse con precaución ya que tienen la limitación de solo un mismo identificador por documento.

```
  <!-- dos elementos HTML con el mismo identificador único no es correcto -->
  <p id="parrafo"></p>
  <p id="parrafo"></p>
```

Selectores de atributo

Los atributos son pedazos de información extra que pueden tener las etiquetas HTML, es posible diferenciar un elemento de otro a nivel CSS mediante el uso de selector de atributo.

```
/* Solamente el input disabled se estilará */
input[disabled] {
  background-color: black;
}

/* La label para el input de email se estilará */
label[for="email"] {
  background-color: red;
}

<div>
  <label for="name">Nombre y apellidos</label>
  <input type="text" id="name" name="name" disabled>
</div>
<div>
  <label for="email">Email</label>
  <input type="email" id="email" name="email">
</div>
```

Añadiendo el nombre del atributo o atributo + valor entre corchetes: [] es posible seleccionar elementos HTML que dispongan de ese atributo.

Selectores descendentes

El selector descendente es la manera de seleccionar elementos que están dentro de otros elementos, simplemente añadiendo un espacio entre selectores se indica que dicho selector debe anidado al anterior para así ser válido.

```
li strong {
  color: red
}

<p>Revisa la <strong>siguiente lista</strong>:</p>
<ul>
  <li><strong>Opción 1</strong>: Macarrones boloñesa</li>
  <li><strong>Opción 2</strong>: Spaghetti pesto</li>
  <li><strong>Opción 2</strong>: Pizza margarita</li>
</ul>
```

Selector de hermandad

El selector de hermandad necesita dos selectores y selecciona elementos con respecto de otro solo si comparten el mismo padre directo:

```
/*
Todos los elementos con clase item que se encuentren después de un párrafo, tendrán el fondo rojo
*/
p ~ .item {
  background: red;
}

<p>Párrafo</p>
<div>Etiqueta div</div>
<div class="item">Este item tendrá el fondo rojo</div>
<div class="item">Este item también tendrá el fondo rojo</div>
```

Selector de adyacente

Similar al selector de hermandad, pero solo selecciona un hermano y si este se encuentra inmediatamente después del primer selector.

```
p + .item {
  background: green;
}

<p>Párrafo</p>
<div class="item">Este item tendrá el fondo verde</div>
<div class="item">Este item no tendrá el fondo verde</div>
```

Pseudoclases

Una pseudoclase es un selector que especifica cuando un elemento está en un estado específico, por ejemplo, cuando el usuario tiene el ratón por encima. Para utilizarlas, simplemente hay que ponerlas detrás de un selector:

```
/*
Un enlace va a tener un color verde hasta que el usuario ponga el cursor encima que se convertirá en rojo
*/
a {
  color: green;
}

/* El selector se refiere al estado hover (cursor del usuario encima) de un enlace */
a:hover {
  color: red;
}
```

- **:first-child:** Selecciona un elemento cuando es el primer hijo respecto de sus hermanos.
- **:last-child:** Similar a :first-child pero para seleccionar el últim elemento.
- **:nth-child(n):** Selecciona el hijo n. Acepta también como parámetro odd o even para hijos pares o impares
- **:hover:** Selecciona un elemento cuando el cursor está encima de él.

- **:active:** Selecciona un elemento cuando el usuario lo ha activado (como por ejemplo al pulsar un botón).
- **:focus:** Selecciona un elemento cuando un elemento interactuable por el usuario ha recibido foco.

5.3. Estilos de cascada

Como su nombre indica: CSS (Cascading Style Sheets) son hojas de estilo en cascada, es decir, un archivo CSS no deja de ser un conjunto de reglas una detrás de otra en el que el orden de estas importa. Si dos reglas tienen la misma especificidad, terminará aplicándose la que aparezca en último lugar: la última regla que aparezca gana.

```
p {
  color: blue;
}

p {
  color: red;
}
```

Para el anterior código, un párrafo <p> finalmente tendrá el color de texto rojo, ya que ambos selectores tienen la misma especificidad y, por lo tanto, se aplica el último: color rojo.

5.4. Especificidad

La especificidad es el método por el cual el navegador acaba decidiendo entre diversas reglas CSS aplicadas al mismo elemento. Tras comprobar la especificidad, solo si hay un empate de esta se aplicará la regla de la cascada (el último, gana).

Los selectores especifican el elemento al que se aplica una declaración CSS, cuanto más específico sea ese selector, mayor puntuación de especificidad tendrá y más probable que termine aplicando esa declaración al elemento.

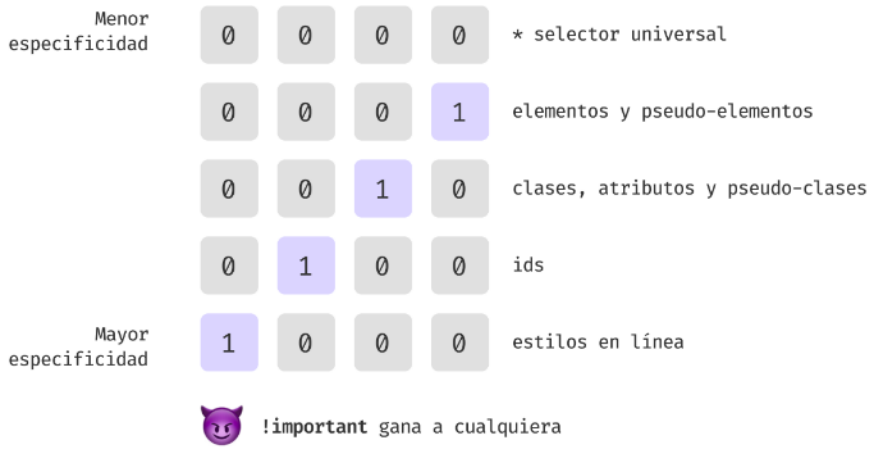

Figura 12. Cuadro de puntuación de especificidad. Fuente: Elaboración propia.

Cada una de las partes de las que se compone un selector va sumando y finalmente gana quien más puntuación tenga en esa suma:

```
/*
Ordenado de mayor a menor especificidad
Selectores inspirados en: https://specificity.keegan.st/
*/

/*
1 ID
1 clase, atributo y pseudo-clase
3 elementos y pseudo-elementos
*/
ul#primary-nav:hover li.active {
}

/*
1 ID
1 clase, atributo y pseudo-clase
2 elementos y pseudo-elementos
```

```
*/
ul#primary-nav li.active {
}

/*
1 ID
0 clase, atributo y pseudo-clase
0 elementos y pseudo-elementos
*/
#primary-nav {
}

/*
0 IDs
1 clase
3 elementos y pseudo-elementos
*/
nav > a:hover::before {
}

/*
0 Ids
1 clase
2 elementos y pseudo-elementos
*/
nav > a:hover {
}
```

Tema 6. Características avanzadas de CSS

6.1. Introducción

La maqueta de sitios web ha sido tradicionalmente un quebradero de cabeza para los maquetadores/programadores. Las tendencias de diseño y las capacidades del lenguaje CSS no siempre fueron a la misma velocidad.

Afortunadamente, en la actualidad, CSS es una herramienta potente y fácil de utilizar que conociendo sus capacidades permite crear sitios web robustos para cualquier tipo de tamaño de pantalla, acorde a las tendencias de diseño y capaz de ahorrar mucho tiempo a la programación.

6.2. Herencia en CSS

Algunas propiedades CSS se heredan de padres a hijos, de tal manera que, si un padre tiene una propiedad, todos sus hijos también van a tenerla.

```
body {
  color: green;
}
```

La propiedad color es una de esas propiedades que se heredan, de tal manera que al haber establecido que el color de texto para la etiqueta body es verde, todo el documento heredará ese color (ya que todo el documento va a ser hijo de body) a no ser que se vuelva a especificar de nuevo la propiedad color:

```
body {
  color: green;
}

.caja {
  color: red;
}
```

En este ejemplo, todo el documento tendrá un color de texto verde excepto los elementos con la clase caja y todos los hijos de estos.

¿Cómo saber si una propiedad es heredable o no? Realmente son pocas las propiedades que se heredan, pero y sobre todo están relacionadas con el texto, pero para saber a ciencia cierta si una propiedad es heredable conviene consultar la referencia de CSS disponible en el sitio web de MDN Web Docs: https://developer.mozilla.org/es/docs/Web/CSS/Reference

6.3. Modelo de caja

Todos los elementos HTML que se visualizan tienen una caja que los contiene, es necesario entender el comportamiento de esa caja para poder maquetar correctamente gracias a CSS.

Figura 13. Modelo de caja CSS. Fuente: elaboración propia.

Una caja tiene 4 zonas:

- **Content box:** Es la parte de la caja propia del contenido, es el espacio intrínseco al contenido que necesita para mostrarse.
- **Padding box:** Es la parte de la caja dedicada al padding, sigue siendo parte del elemento, pero es un espacio reservado entre el contenido y el exterior de la caja. Al establecer un fondo (imagen, color plano o gradiente) al elemento, esta zona también estará incluida como espacio para mostrar dicho fondo.
- **Border box:** Espacio reservado para el borde, a partir del borde «acaba» el elemento.
- **Margin box:** Espacio reservado para los márgenes, Este espacio no corresponde al elemento en sí, sino que es el espacio que separa un elemento de los que tiene alrededor.

Mediante CSS se puede modificar el tamaño, estilo (colores, fondos, etc.) de cada una de estas zonas.

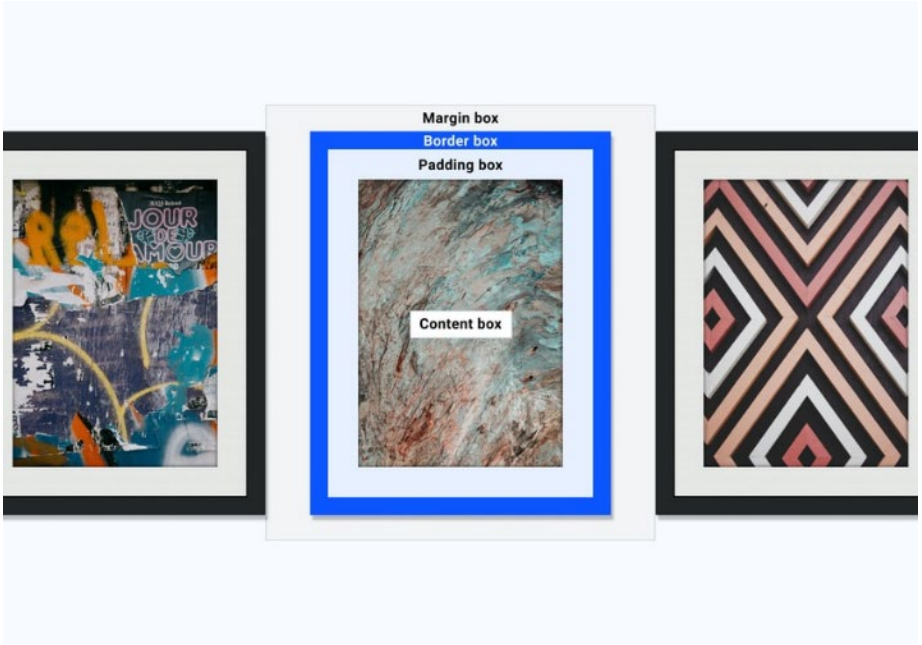

Figura 14. Analogía al mundo real del box model. Fuente: Modelo de Caja, s.f.

Conviene saber que existe la propiedad box-sizing para modificar el comportamiento de este modelo de caja:

- **content-box:** Es el comportamiento por defecto en el que ni el padding ni el borde pertenecen al elemento.
- **border-box:** Con este valor el padding y el borde si pertenecen al elemento, este comportamiento es el habitual a la hora de maquetar.

Como border-box no es el valor por defecto, pero es ampliamente utilizado y por así decirlo: estándar, es habitual encontrar esta regla en todas las hojas de estilo:

```
* {
  box-sizing: border-box
}
```

6.4. Flexbox

Flexbox viene a solucionar problemas a la hora de maquetar, hasta la llegada de flexbox era complicado y controlar cómo los elementos HTML se relacionaban entre sí: algo tan fácil como poner un elemento al lado de otro no era una labor sencilla.

El modelo de flexbox mejora la manera en la que se alinean y distribuyen hijos con respecto de su contenedor y hace más sencillo su desarrollo y mantenimiento.

Terminología

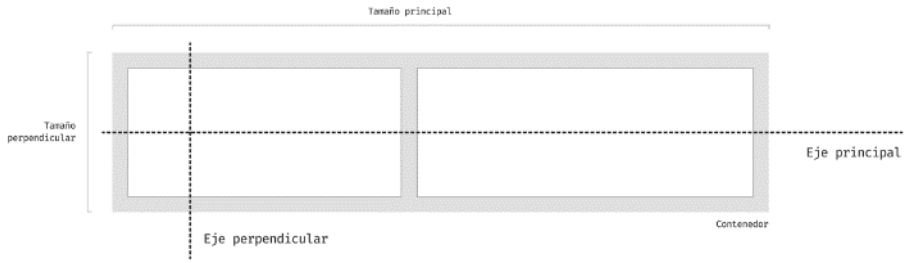

Figura 15. Terminología de flexbox. Fuente: elaboración propia.

- **Eje principal**: Es el eje que los elementos van a seguir para ir colocándose.
- **Eje perpendicular**: El eje perpendicular al principal establece el otro eje mediante los elementos pueden ubicarse.
- **Tamaño principal:** El espacio que los elementos tienen disponible respecto del eje principal.
- **Tamaño perpendicular:** El espacio que los elementos tienen disponible respecto del eje perpendicular.

En cuanto se añade display: flex al padre, se activan estos ejes y los elementos se distribuyen respecto a ellos.

Propiedades

flex-direction

```
.padre {
  flex-direction: row | row-reverse | column | column-reverse;
```

Propiedad	Eje principal	Eje perpendicular	Dirección eje principal
row (por defecto)	horizontal	vertical	izquierda a derecha
row-reverse	horizontal	vertical	derecha a izquierda
column	vertical	horizontal	arriba a abajo
column-reverse	vertical	horizontal	abajo a arriba

Tabla 1. Propiedades para flex-direction. Fuente: elaboración propia.

justify-content

Esta propiedad maneja cómo se distribuyen los elementos a través del eje principal (siempre que haya espacio disponible):

- **flex-start:** Los elementos se sitúan pegados al inicio del eje principal.
- **flex-end**: Los elementos se sitúan pegados al final del eje principal.
- **center**: Los elementos se sitúan pegados en el medio del eje principal.
- **space-between**: Los elementos se distribuyen uniformemente a través del eje principal.
- **space-around**: Los elementos se distribuyen de tal manera que tengan el mismo espacio alrededor.

align-items

Esta propiedad manera cómo se distribuyen los elementos a través del eje perpendicular (siempre que haya espacio disponible), es como el justify-content pero para el eje perpendicular.

- **flex-start:** Los elementos se sitúan pegados al inicio del eje perpendicular.
- **flex-end:** Los elementos se sitúan pegados al final del eje perpendicular.
- **center:** Los elementos se sitúan pegados en el medio del eje perpendicular.
- **strech:** Los elementos se estiran para ocupar todo el eje perpendicular.
- **baseline:** Los elementos se ubican en el eje perpendicular de tal manera que todas las líneas base de texto están alineadas.

flex-basis, flex-grow, flex-shirnk

Estas propiedades se utilizan para elementos dentro de un flujo flex. Manejan la manera en la que los elementos se reparten el espacio disponible del tamaño principal (el correspondiente al eje principal).

- **flex-basis:** Especifica el espacio que inicialmente debe ocupar un elemento. Inicialmente porque los elementos tienden a ocupar todo el espacio disponible: si no ocupan todo el espacio disponible se estirarán por encima del flex-basis establecido y si ocupan demasiado, se harán más cortos midiendo menos que su flex-basis.
- **flex-grow:** Indica el factor de crecimiento de un elemento cuando hay espacio disponible. Es decir, si de entre todos los hermanos sobra espacio del tamaño principal estos se lo repartirán atendiendo al factor establecido en flex-grow. Por ejemplo, un elemento con flex-grow: 3 ocupará 3 veces más de ese espacio disponible que un hermano con flex-grow: 3.
- **flex-shirnk:** Similar a flex-grow pero como factor de contracción. Es decir, como a mayor flex-shirnk, más pequeño se hará un elemento cuando falte espacio en el tamaño principal.

order

Se utiliza en elementos de dentro de un flujo de flex. Con esta propiedad se puede alterar la posición de ese elemento particular, simplemente añadiendo la propiedad order y la posición en el que debe aparecer el elemento.

```
  .item {
  order: 3
}
```

gap

Establece un espacio de separación entre elementos.

```
.padre {
  display: flex;
  row-gap: 10px;
  column-gap: 15px;
  /* atajo para row-gap y column-gap */
  gap: 10px 15px;
}
```

En el vídeo *Introducción a flexbox* se repasan todas estas propiedades de flexbox y se replica la maqueta de una cabecera de un sitio web real.

 Accede al vídeo

6.5. Flotación

La propiedad float sirve para envolver a un elemento alrededor de otro. Lo habitual es utilizarlo para un elemento como una imagen flotada con texto alrededor que la envuelve:

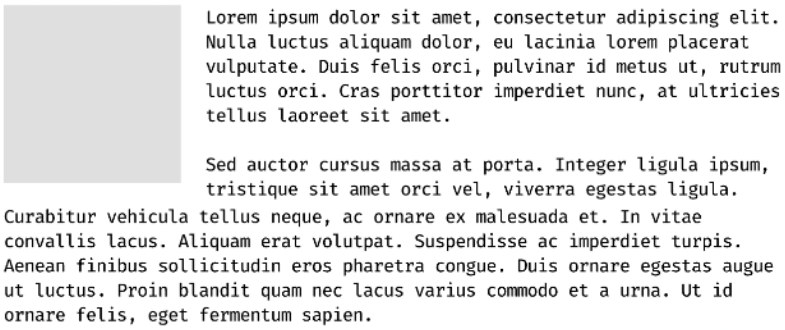

Figura 16. Utilización de float para envolver elemento con texto. Fuente: elaboración propia.

```
.clear {
    clear: both;
  }
  img {
    float: left;
  }

<div class="clear">
    <img src="https://via.placeholder.com/150x150">
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla luctus aliquam dolor, eu lacinia lorem placerat vulputate. Duis felis orci, pulvinar id metus ut, rutrum luctus orci. Cras porttitor imperdiet nunc, at ultricies tellus laoreet sit amet.</p>
    <p>Sed auctor cursus massa at porta. Integer ligula ipsum, tristique sit amet orci vel, viverra egestas ligula. Curabitur vehicula tellus neque, ac ornare ex malesuada et. In vitae convallis lacus. Aliquam erat volutpat. Suspendisse ac imperdiet turpis. Aenean finibus sollicitudin eros pharetra congue. Duis ornare egestas augue ut luctus. Proin blandit quam nec lacus varius commodo et a urna. Ut id ornare felis, eget fermentum sapien.</p>
  </div>
```

La propiedad clear sirve para reiniciar el float, esto quiere decir que sin un elemento envolviendo un elemento flotado, ese comportamiento se extendería al resto de la página, provocando comportamientos no deseados, es por ello por lo que se reinicia el flujo normal de HTML con un elemento que envuelva elementos flotados. clear puede tener los siguientes valores:

- left. Reinicia elementos flotados a la izquierda.
- right. Reinicia elementos flotados a la derecha.
- both. Reinicia elementos flotados a ambos lados (el más habitual).

Hasta la llegada de flexbox esta propiedad era la única manera de posicionar elementos uno al lado de otro, pero era muy engorroso de utilizar ya que no estaba diseñado para ello. Afortunadamente con la llegada de flexbox la utilización de float ha vuelto a su cometido de origen.

6.6. Visualización

La propiedad display controla la visualización y/o el comportamiento en línea / bloque de un elemento.

- none: Oculta un elemento, el navegador no lo representará
- block: Fuerza a que un elemento sea de bloque.
- inline: Fuerza a que un elemento sea de línea.
- inline-block: Un híbrido entre inline y block: se puede setear ancho / y alto y paddings verticales cosa que en un elemento en línea no se puede y no tiende a ocupar tanto espacio horizontal como pueda como si hace un elemento de bloque.

```
/* Esta caja no se verá en el navegador */
.caja {
  display: none;
}
```

```
  /* Este elemento ahora se comporta como un elemento de bloque */
  span {
    display: block;
  }

  /* Este elemento ahora se comporta como un elemento de línea */
  div {
    display: inline;
  }

  /*
  Ahora un enlace puede tener el aspecto de botón (con paddings verticales)
y permite elementos al lado
  */
  a {
    display: inline-block;
  }
```

6.7. Posicionamientos especiales

Por defecto, como es lógico, los elementos HTML se van mostrando unos detrás de otros siguiendo un flujo de izquierda a derecha y de arriba a abajo (salvo que ese flujo se vea alterado por CSS). Una de las maneras de modificar ese flujo es utilizando la propiedad CSS position.

Static

Posición por defecto de todos los elementos HTML, se posicionan donde les toque, es decir, el navegador va colocando elementos HTML uno detrás de otro y su posición no se ve alterada de ninguna forma.

Relative

La posición relativa de un elemento empieza igual que la estática, pero se puede ver alterada por las propiedades: top, left, bottom o right, de tal manera que por ejemplo añadiendo la declaración top: 10px el elemento se desplaza 10 píxeles en el eje vertical hacia abajo tomando como referencia su posición estática.

```
p.movido {
  position: relative;
  top: -10px;
  left: 20px;
}
```

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla luctus aliquam dolor, eu lacinia lorem placerat vulputate. Duis felis orci, pulvinar id metus ut, rutrum luctus orci. Cras porttitor imperdiet nunc, at ultricies tellus laoreet sit amet.</p>

<p class="movido">Sed auctor cursus massa at porta. Integer ligula ipsum, tristique sit amet orci vel, viverra egestas ligula.</p>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla luctus aliquam dolor, eu lacinia lorem placerat vulputate. Duis felis orci, pulvinar id metus ut, rutrum luctus orci. Cras porttitor imperdiet nunc, at ultricies tellus laoreet sit amet.</p>
```

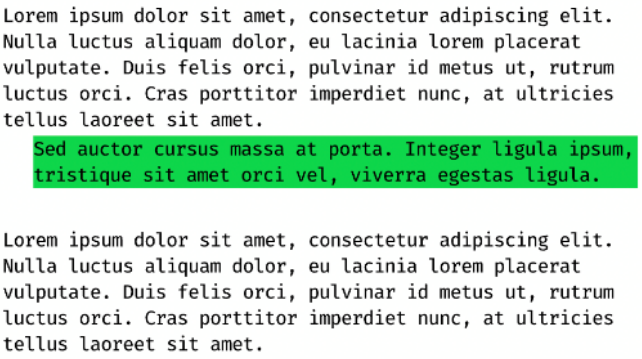

Figura 17. Ejemplo de párrafo con posición relativa. Fuente: elaboración propia.

Al aplicar a un elemento un desplazamiento con position: relative el elemento se desplaza, pero el lugar que ocupa en el flujo HTML se mantiene.

absolute

La posición absoluta de un elemento es muy similar a la relativa, la diferencia está en que la posición desde la que va a empezar a moverse el elemento no es su posición estática si no **la posición estática del primer padre que tenga una posición relativa**. Si no encuentra un elemento con ninguna posición relativa, tomará la posición estática del elemento body que coincide con la coordenada 0,0 del viewport.

```
p.movido {
  position: absolute;
  top: -10px;
  left: 20px;
}
```

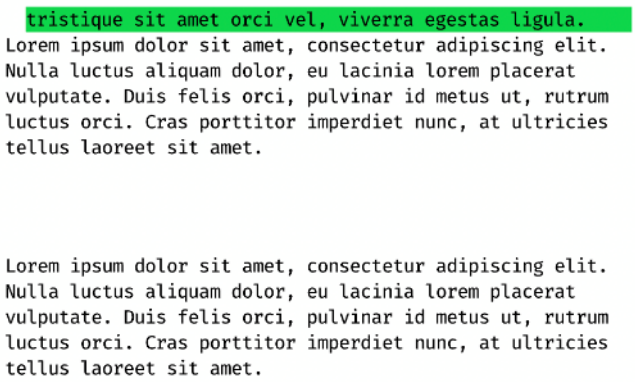

Figura 18. Ejemplo de párrafo con posición absoluta. Fuente: elaboración propia.

En el vídeo *Posicionamientos especiales* se explica en detalle esta propiedad utilizando para ello un ejemplo aplicable en el mundo real.

Accede al vídeo

6.8. Contenido desbordado

Aunque por defecto los elementos HTML tanto de línea como de bloque tienden a ocupar al menos todo lo que ocupe su contenido, es posible que un elemento no tenga tanto espacio disponible para mostrarse como contenido a mostrar y por lo tanto se produce un desbordamiento.

```
  <p class="text">En un lugar de la mancha</p>

  /* Una caja de 100x50 no es suficiente para mostrar todo ese texto, por lo tanto se producirá desbordamiento */
  .text {
    width: 100px;
    height: 50px;
}
```

Gracias a las propiedades overflow (para ambos ejes), overflow-x y overflow-y se puede controlar qué ocurre cuando aparece el desbordamiento. Acepta los siguientes valores:

- **hidden**: Oculta la parte desbordada.
- **scroll**: Añade automáticamente al elemento scroll en la parte desbordada.

6.9. Tabulación de datos a través de tablas

La utilización de etiquetas de tabla en HTML es correcta cuando la información a mostrar debe tiene la naturaleza de una tabla: Tiene encabezados de tabla, filas, columnas, etc.

Sin embargo, podemos recrear el comportamiento visual de una tabla para maquetar otros elementos HTML que no sean una tabla a través de CSS.

Gracias a valores que acepta la propiedad display se puede conseguir tal fin:

- **table:** Aplicado al contenedor principal para que se comporte como una tabla. Como la etiqueta HTML table.
- **table-row:** Se añade a los elementos HTML que hacen de contenedor para cada una de las filas. Como la etiqueta HTML tr.
- **table-cell:** Se añade a los elementos HTML que hacen de celda. Como la etiqueta HTML td.

6.10. Uso de listas como menús y barras de navegación

Un elemento muy común en sitios web es el menú de navegación. Dicho menú ayuda al usuario a navegar por el sitio no deja de ser un listado de enlaces, y como listado debe estar marcado en HTML como tal.

HTML da el marcado y CSS se debe encargar del aspecto visual para convertir un listado en un menú de navegación.

Este es un ejemplo de menú de navegación:

```
<ul class="menu">
  <li>
    <a href="index.html">Inicio</a>
  </li>
  <li>
    <a href="sobre-nosotros.html">Sobre nosotros</a>
  </li>
  <li>
    <a href="contacto.html">Contacto</a>
  </li>
```

```
</ul>

.menu {
  /* Quitar estilos propios de un listado */
  list-style: none;
  padding: 0;
  margin: 0;
  /* Un elemento al lado de otro */
  display: flex;
  /* Separación entre elementos de 10px */
  gap: 10px;
}

.menu a {
  /* Quitar estilos propios de un enlace */
  text-decoration: none;
  /* Darle aspecto de botón */
  display: inline-block;
  padding: 10px;
  background-color: blue;
  color: white;
}

.menu a:hover {
  /* Cambiar el color de fondo cuando el usuario sitúa el cursor encima
*/
  background-color: green;
}
```

6.11. Font fase

Sin font face la lista de tipografías que se pueden utilizar en HTML se reduce a las denominadas Web Safe Fonts: un listado muy acotado de tipografías que están disponibles en la mayoría de los dispositivos que son capaces de mostrar sitios web

que están disponibles en este enlace: https://developer.mozilla.org/en-US/docs/Learn/CSS/Styling_text/Fundamentals#web_safe_fonts

Gracias a font face se puede utilizar cualquier tipografía en un sitio web, al igual que el sitio entrega al navegador su marcado HTML, sus imágenes, etc. también entrega el archivo de tipografía para que le navegador pueda mostrar texto utilizando esa familia tipográfica.

```
/* Declaración del font face */
@font-face {
  /* Nombre de la tipografía que se declara */
  font-family: 'Roboto Condensed';
  /* Estilo de la fuente (normal | italic | oblique) */
  font-style: normal;
  /* Peso de la fuente ([100-900]) */
  font-weight: 400;
  /* Archivos fuente */
  src: url(robotocondensed-400.woff2) format('woff2');
}

@font-face {
  font-family: 'Roboto Condensed';
  font-style: normal;
  font-weight: 700;
  src: url(robotocondensed-700.woff2) format('woff2');
}

/* Utilización */
body {
  font-family: 'Roboto Condensed';
}
```

Google Fonts

La utilización de font-face conlleva cierto trabajo al maquetador: Debe disponer del archivo fuente de la tipografía, licencia para utilizarla, utilizar el formato correcto para distintos navegadores, hacer todas las declaraciones en el CSS, etc.

Google Fonts: (https://fonts.google.com/) simplifica el uso de tipografías en los sitios web: Simplemente seleccionando la tipografía que se desea utilizar, genera un archivo CSS listo para añadir a cualquier sitio web y el dueño del sitio web ya no debe de preocuparse de nada más como licencias, alojar las tipografías, etc.

Ejemplo:

```
<html>
  <head>
    <!-- Añadiendo estas 3 etiquetas link ya estará disponible la tipografía Roboto Condensed para los pesos 400 y 700 -->
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Roboto+Condensed:wght@400;700&display=swap" rel="stylesheet">
  </head>
</html>
```

Iconos tipográficos

Al igual que una tipografía no deja de ser un conjunto de formas relacionadas con un carácter, en el desarrollo web se adoptaron los iconos tipográficos como una manera de mostrar iconos apoyados en la funcionalidad font-face.

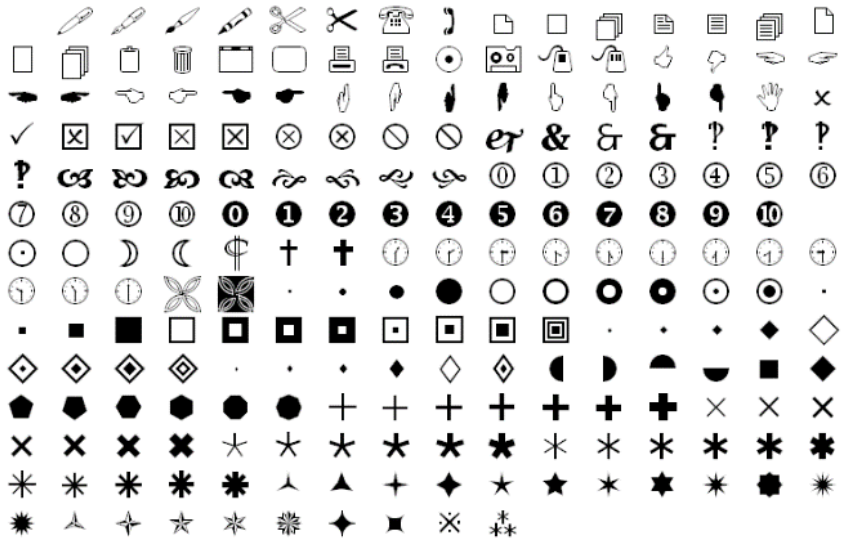

Figura 19. Tipografía Wingdigs Fuente: Monedula, 2004.

Tipografía Wingdings que, en lugar de mostrar tipos, muestra iconos.

Gracias a utilizar una tipografía en lugar de imágenes de mapa de bit (como jpg o png) para mostrar iconos:

- Se gana en facilidad de uso y calidad de visualización. Las tipografías almacenan formas vectoriales de tal manera que se visualizarán correctamente independientemente del tamaño al que se muestren.
- Aunque muestre un icono, **para el navegador no deja de ser un texto**, por lo que están disponibles todas las **propiedades CSS para textos** como el cambio de color.

Mediante diferentes herramientas un maquetador puede construir su propia colección de iconos recogida en un archivo de tipografía como por ejemplo IcoMoon.

6.12. Planificación, organización y mantenimiento de CSS

A medida que un proyecto web se va haciendo grande, mantener las hojas de estilo puede ser un reto si no se planifica correctamente desde un inicio y se siguen buenas prácticas.

Selectores al mínimo

A medida que se encadenan selectores en una regla CSS aumenta su especificidad y al aumentar su especificidad a la larga será más difícil sobrescribir esas reglas (habrá que crear selectores más específicos y se puede convertir en una guerra de especificidad).

Por lo tanto, es conveniente crear nuevas clases (¡son gratis!) para cada elemento que necesite código CSS relacionado que crear selectores muy específicos.

Por ejemplo:

```
<ul class="lista">
  <li>
    <a href="#" class="lista-enlace">Enlace</a>
  </li>
</ul>

/* Opción 1*/
.lista a {
  color: Green
}

/* Opción 2 */
/* Más recomendable, ya que no hay problema ninguno en crear una clase lista-enlace y mantener los selectores del CSS simple */
```

```
.lista-enlace {
  color: green
}
```

Evitar la utilización de important

Relacionado con la guerra de especificidad, es habitual en programadores con poca experiencia recurrir a !important. Esta característica de CSS hace que el valor aplicado tenga una especificidad muy alta que hace que sea muy difícil sobrescribirla.

Un maquetador nunca puede estar seguro de cuándo va a necesitar sobrescribir una propiedad, por lo que siempre debe dejar la posibilidad de hacerlo. De esta manera hace más fácil el desarrollo posterior de nuevas características del proyecto y mantenimiento de su código.

Separar contenedor del contenido

Uno de los principios S.O.L.I.D. es el Principio de Responsabilidad Única que establece que cada función, clase o módulo debe tener responsabilidad sobre una sola parte y dicha responsabilidad debe estar encapsulada en su totalidad. Dicho principio puede aplicarse a la maquetación, de tal manera que cada elemento es responsable de su propia visualización y no debe condicionar ni estar condicionado por lo que haya alrededor (ni por sus padres ni por sus hijos).

Este principio entendido en la maquetación evita muchos errores y mejora el mantenimiento del código. Un ejemplo de este principio:

```
<div class="fila">
  <div class="item">...</div>
  <div class="item">...</div>
  <div class="item">...</div>
</div>
```

```
<!-- opción más recomendable -->
<div class="fila">
  <div class="columna">
    <div class="item-ok">...</div>
  </div>
  <div class="columna">
    <div class="item-ok">...</div>
  </div>
  <div class="columna">
    <div class="item-ok">...</div>
  </div>
</div>

.fila {
  display: flex;
  flex-wrap: wrap;
}

.item,
.columna {
  flex-basis: 50%;
}
```

La segunda opción en la que se añade un div extra para establecer el ancho es más recomendable: utiliza el **principio de responsabilidad única** en la que el elemento item solo debe ser responsable de si mismo, debe solo saber aquello que necesita para mostrarse, pero no debe ser responsable del grid de items al que pertenece.

La responsabilidad del grid debe recaer al elemento fila y columna, aunque se utilice un div a modo de contenedor. De esta manera, el elemento item puede ser visualizado directamente sin cambiar su código en otros grids o en otras partes del documento.

Comentarios en CSS

Al igual que en lenguajes de programación, la utilización de comentarios en CSS es una buena práctica. Si la propiedad CSS o determinado selector es auto explicativo, no es necesario comentarlo, pero para todo lo demás, como decisiones que se toman en el proyecto, explicaciones de qué hace determinada regla, etc. es recomendable comentarlo.

También puede ser interesante comentar las diferentes partes o zonas de un fichero CSS: Como reglas comunes, reglas específicas para determinadas páginas, etc.

```
/* Grid de items */
.fila {
  display: flex;
}

.columna {
  flex-basis: 50%;
}

/* Definición de item */
.item {
  background-color: blue;
  color: white;
  border: 1px solid green;
}

.item:hover {
  border-width: 2px; /* Cuidado, esto hace que el item aumente de tamaño
*/
}
```

Metodologías de naming en CSS

Aunque CSS parezca un lenguaje sencillo, se tiende a complicar en proyectos grandes y si se cuenta con una buena estructura y organización del código CSS el desarrollo de nuevas funcionalidades y mantenimiento de este será más sencillo.

El uso de metodologías como en lenguajes de programación, propone establecer reglas y métodos de trabajo ya comprobados y contrastados que ayudan a escribir mejor código, en el caso de CSS más mantenible y escalable.

Las metodologías más conocidas de CSS son: OOCSS y BEM

OOCSS

Su propio nombre indica la filosofía de esta metodología: CSS Orientado a Objetos. Esta metodología tiene dos ideas principales:

- Separar la estructura del diseño.
- Separa el contendor del contenido.

De tal manera que tenemos clases generales (como si fuesen objetos) y un elemento HTML puede tener muchas de esas clases generales (como si heredase de esos objetos).

Por ejemplo, un elemento de un listado de productos:

```
<div class="product list-item">
  ...
</div>
```

Tiene dos clases:

- product: A través de product el elemento recibirá estilos relacionados con ser un producto: como su imagen, descripción, etc.
- list-item: A través de list-item el elemento recibirá estilos relacionados por ser un elemento dentro de un listado: posición, transformaciones especiales por estar en un listado de su contenido, etc.

BEM

BEM (Bloque/Elemento/Modificador) es una convención de nombres para código CSS que lo hace más sencillo de leer, mantener y escalar.

Define 3 elementos con los que construirá los selectores:

- **Bloque**: Componente independiente que tiene significado por sí mismo.
- **Elemento**: Una parte de un bloque, solo tiene sentido si está dentro de este y no como parte independiente (si no sería un bloque).
- **Modificador**: Relacionado con un bloque o elemento, se utiliza para cambiar la apariencia dentro de un conjunto discreto de modificadores.

Cada elemento tendrá únicamente un selector y se compone de la siguiente manera: [BLOQUE]__[ELEMENTO]. Si un bloque o un elemento requiere un modificador se separa con --.

Tanto el separador entre bloque y elemento __ como el prefijo para el modificador -- y aunque no es recomendable, pueden ser cambiados por el maquetador a su conveniencia (siempre que mantenga la misma convención de nombrado en todo el proyecto).

Ejemplo de uso de BEM:

```
<!-- bloque grid -->
<div class="grid">
  <!-- elemento del bloque grid con modificador featured -->
  <div class="grid__item grid__item--featured">
    <!-- bloque item con modificador featured -->
    <div class="item item--featured">
      <!-- elemento del bloque item -->
      <div class="item__title">
        Título del item
      </div>
      <div class="item__price">
        29.99€
      </div>
    </div>
  </div>
  <!-- elemento del bloque grid -->
  <div class="grid__item">
    <!-- bloque item -->
    <div class="item">
      ...
    </div>
  </div>
</div>
```

6.13. Referencias bibliográficas

Modelo de Caja. (s.f.). Obtenido de web.dev: https://web.dev/learn/css/box-model/#una-analogia-util

Monedula. (22 de junio de 2004). Wikipedia. Obtenido de Mosaico de caracteres Wingding 2: https://es.wikipedia.org/wiki/Wingdings#/media/Archivo:Wingdings2.png

Tema 7. Responsive web design

7.1. Introducción

El diseño web responsive es una filosofía a la hora de diseñar y desarrollar sitios web que se basa en la adaptación del sitio a cualquier escenario en el que se visite: tamaño de pantalla, dispositivo, velocidad de conexión, etc. es decir: **darle al usuario la mejor experiencia** independientemente del medio que utilice explorar el sitio web.

Aunque no siempre ha sido así, actualmente existe tecnología de sobra para conseguir un sitio responsive, lo cual tiene las siguientes ventajas:

- **Una única versión del sitio web**. Si solo existe un único sitio pero que puede adaptarse a cualquier dispositivo mejora la experiencia del usuario ya que es independiente desde el dispositivo que visite el sitio. Puede darse por ejemplo que inicie la exploración del sitio en un dispositivo de escritorio con conexión de fibra para acabarla en un dispositivo móvil con conexión intermitente 3G.
- **Reducción de costes**. Al tener un único sitio solamente debe desarrollarse, actualizarse y mantenerse uno. Esto reduce el tiempo de desarrollo, gestión etc.

7.2. Responsive vs. adaptive

Puede parecer lo mismo, pero no lo es: En la filosofía del **diseño adaptive** se trabaja con anchuras fijas de dispositivos concretos y si el sitio se visita desde un dispositivo con un tamaño de pantalla diferente: La diferencia entre el espacio fijo al que está diseñado el sitio y el tamaño extra del que dispone el dispositivo se desperdiciará, es decir, el sitio no ocupará todo el espacio disponible.

En el mundo real no existen un número limitado de resoluciones ni de dispositivos, por lo que no tiene sentido trabajar mediante diseño adaptative si no que lo ideal es el **diseño responsive** donde el sitio ofrece la mejor experiencia aprovechando todo lo que el dispositivo pueda ofrecer.

Siguiendo el criteo del flujo HTML donde un elemento tenderá a ocupar lo que necesite (elemento de línea) o todo el espacio disponible (elemento de bloque) y de arriba a abajo y de izquierda a derecha **el diseño responsive parece creado para tal fin** mientras que establecer anchos y/o altos fijos y no dejar que los elementos fluyan (diseño adaptative) no parece ser el comportamiento natural de un documento HTML.

7.3. Diseño líquido: columnado, estructuración y cajas flexibles

Figura 20. Content is like water. Fuente: (Walter, 2018).

Para pensar en un diseño líquido de sitios se deben tener claros dos puntos en los que se basa:

- **El contenido es líquido**, es decir, se adapta al recipiente que lo contiene y este es el comportamiento por defecto que tiene el contenido gracias al flujo HTML y debe mantenerse así.
- **Principio de responsabilidad única**, uno de los principios S.O.L.I.D. dice que cada módulo debe tener responsabilidad sobre una sola parte de la funcionalidad.

Juntando estos dos conceptos se puede concluir que el contenido no debe ser responsable de *mantenerse*, sino que debe ser un padre encargado (y únicamente encargado de ello) de contenerlo. Es entonces si se dota de flexibilidad a esos padres contenedores y son capaces de rellenar todo el espacio disponible se conseguirán sitios responsive de una forma mucho más sencilla y robusta.

Este concepto de que los padres deben ser quienes manejen al contenido es lo que se conoce como grid.

Figura 21. Ejemplo de grid a 12 columnas en diseño web responsive Fuente: elaboración propia.

Un ejemplo de rejilla muy común es la de dividir el espacio disponible en 12 columnas, de tal manera que cada celda puede tener de ancho entre 1 a 12 columnas (el contenido como es líquido ocupará todo el espacio disponible de esa celda) y, además, la misma celda puede no tener el mismo ancho en cada tamaño de dispositivo, por ejemplo, un item puede ocupar 12 celdas de ancho en tamaño mobile (ancho completo) pero ocupar únicamente 7 celdas en tamaño escritorio (7 / 12 del ancho).

7.4. @ media queries

Siendo responsabilidad del HTML el contenido y de la presentación de este del CSS, tiene sentido entonces que la responsabilidad de orquestar el diseño responsive sea del CSS.

Para ello, existe la funcionalidad @media que es la clave del desarrollo web responsive, con ella se puede detectar: tipo de dispositivo, tamaño de pantalla, orientación de esta, etc. es decir, todo lo que se necesita para crear sitios responsive.

Ejemplo de uso:

```
  /*
  Para tamaños de pantalla menores de 768px el color de fondo será azul, mientras que para tamaños de pantalla mayores será verde
  */
  body {
    background-color: blue
  }
```

```
@media screen and (min-width: 768px) {
  body {
    background-color: Green
  }
}
```

Adaptación a los tipos de medios: pantallas e impresión

Es posible detectar si el sitio se va a ver en una pantalla o impreso en un papel con la media-type de una @media query.

La utilización de esta funcionalidad es opcional.

```
@media screen {
  body {
    font-size: 14px
  }
}

/* El tamaño de fuente será 12px si el documento es impreso */
@media print {
  body {
    font-size: 12px
  }
}
```

Operadores lógicos @media: and, comma, not

Los operadores lógicos pueden utilizarse al redactar queries para concatenar afirmaciones y conseguir más potencia de selección para estas queries:

```
/* And */
@media (min-width: 768px) and (max-width: 1023px) {
  /* Estas reglas solo aplicarán a tamaños entre 768px y 1023px */
}
```

```
  /* Comma (similar or) */
  @media print, screen and (max-width: 1000px) {
    /* Estas reglas solo aplicarán a impresión y tamaños de pantalla menores
de 1000px */
  }

  /* Not: aplica a todo el media query, no solo a la parte que tiene al lado
*/
  @media not all and (min-width: 480px) {
    /* Estas reglas aplican a tamaños menores de 480px */
  }
```

Otras características que puede detectar @media

Característica	Definición	Puede utilizarse con prefijo min-, max-
width	Ancho del viewport	Sí
height	Alto del viewport	Sí
orientation	Orientación del dispositivo: portrait o landscape	No
aspect-ratio	Relación entre el ancho y alto del viewport	Sí

Tabla 2. Listado de características detectables por @media más utilizadas. Fuente: elaboración propia.

7.5. Mobile first

El Mobile first como su propio nombre indica: primero móvil. Es una filosofía a la hora de construir sitios web, ya sea diseñando, maquetando o pensando en el producto se debe ser consciente de que el móvil va primero. Esto quiere decir, que, aunque lo habitual sea trabajar (diseñadores, maquetadores y programadores) en dispositivos de escritorio, el consumo de sitios y aplicaciones se realiza mayoritariamente en

dispositivos móviles, por lo tanto, se debe trabajar primero para tamaños móviles y después para el resto de los dispositivos.

Llevada esta filosofía a la maquetación, es razonable entonces utilizar únicamente media queries para tamaños de pantalla por encima de tamaños móviles en lugar de al revés, es decir, al maquetar un sitio primero es móvil y luego hay media queries tratando las *excepciones* como tamaños de pantalla diferentes al móvil:

```
  /* Esto no es mobile first */
  /* La caja por defecto tiene un color de fondo azul */
  .caja {
    background-color: blue;
  }

  /* Hay una excepción para tamaños de pantalla menores que 768px (mobile)
*/
  @media (max-width: 767px) {
    .caja {
      background-color: green;
    }
  }

  /* Esto SI es mobile first */
  .caja {
    background-color: green;
  }

  @media (min-width: 768px) {
    .caja {
      background-color: blue;
    }
  }
```

7.6. Referencias bibliográficas

Walter, S. (30 de abril de 2018). *Stéphanie Walter. Obtenido de Download the "Content is like Water" Illustration:* https://stephaniewalter.design/blog/download-illustration-content-is-like-water/

Tema 8. Dinamización del entorno digital

8.1. Introducción

Parte de crear una buena experiencia de usuario en un sitio web son las animaciones. El contenido de un sitio web (y cada vez más) no permanece estático, si no que una vez cargado se ve alterado por el usuario o por nuevo contenido que entra en el sitio.

Este feedback que debe recibir el usuario o simplemente una expresión artística del sitio se ve enormemente mejorado si se hace uso de animaciones.

En este aspecto CSS no es todo lo potente que un artista de animaciones podría esperar, pero si resuelve de una manera sencilla animaciones y transiciones que se requieren en el día a día de la interacción con un sitio web.

8.2. Transacciones

Una transición ocurre cuando un elemento cambia de un estado hacia otro. En el momento que hay un cambio de estado, **el navegador genera una secuencia de estados** y los coloca justo en medio del estado inicial y final. Son solo esos 2 estados (inicial y final) los que el maquetador debe indicar, todos los pasos intermedios corren por cuenta del navegador.

Figura 22. Estados de una transición. Fuente: elaboración propia.

Ya que las transiciones están limitadas a estos dos estados, pueden carecer de cierta flexibilidad, pero al mismo tiempo son bastante fáciles de utilizar. En una regla CSS se establece el punto inicial y en otra regla CSS se establece el punto final y con la propiedad transition se especifica que otras propiedades de CSS se desean animar.

```
.box {
    width: 100px;
    height: 100px;
    background-color: red;
    transition: background-color .3s ease-in;
  }

  .box:hover {
    background-color: #ffcc11
  }
```

En el ejemplo anterior el color de fondo tiene dos estados: el inicial especificado por la primera regla y el final especificado por la segunda regla. En la primera regla se añade la propiedad transition, la cual es un shorthand de las siguientes propiedades:

- **transition-property:** Especifica la propiedad que se va a transformar.
- **transition-duration:** Especifica la duración de la transformación.
- **transition-timing-function:** Especifica la curva de aceleración de la transformación.

No todas las propiedades CSS se pueden transformar, solo es posible animar aquellas propiedades recogidas en la siguiente lista, las cuales sobre todo son relacionadas con propiedades de visualización de elementos: (https://developer.mozilla.org/es/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions#propiedades_que_pueden_ser_animadas)

```
/*
background-color: propiedad que se anima
.3s: duración de la animación
ease-in: curva de aceleración de la animación
*/
.example {
  transition: background-color .3s ease-in;
}
```

Las curvas de aceleración están prefijadas en CSS, pero se pueden personalizar utilizando la función cubic-bezier. En cubic-bezier.com se puede manejar de una forma sencilla dicha función y compararla con el resto de las curvas predefinidas por CSS.

En el vídeo llamado *Transiciones en CSS* se maqueta un botón que transiciona entre colores de fondo.

8.3. Animaciones

CSS permite realizar animaciones más complejas que las transiciones, basadas en línea de tiempo. Una animación al igual que las transiciones se basan en propiedades que van los elementos van mutando y así se consigue la animación.

Para realizar una animación en CSS se necesitan dos partes: el código de la animación en sí (@keyframes) y aplicar dicha animación a un elemento (propiedad animation).

```
/* Define la animación pulse con una transición de color de letra y de fondo
*/
  @keyframes pulse {
    0% {
      background-color: blue;
      color: white;
    }
    55% {
      background-color: blue;
      color: red;
    }
    100% {
      background-color: green;
      color: yellow;
    }
  }

  .element {
    /* Aplica al elemento la animación "pulse", durará 5 segundos y se
repetirá infinitamente */
    animation: pulse 5s infinite;
  }
```

La propiedad animation es un shorthand de unas cuantas propiedades relacionadas con la animación de un elemento, las más utilizadas son:

- **animation-name.** Indica el nombre la animación que se va a aplicar al elemento. La animación debe estar declarada con un @keyframes.
- **animation-duration**. La duración de cada ciclo de la animación.
- **animation-timing-function.** Especifica la curva de aceleración de la animación.
- **animation-delay.** La animación puede retrasarse desde que el elemento se muestra en pantalla.

8.4. Instrucción a los Gráficos Vectoriales Escalables SVG

Ya no solo en web, en artes gráficas y en multimedia en general existen dos tipos de imágenes según el método que utilizan para representarse: imágenes de mapa de bits e imágenes vectoriales.

- **Mapa de bits**: La imagen es representada por una cuadrícula de pixeles, en cada píxel hay un color y ampliando una imagen se pueden apreciar dichos pixeles.
- **Imagen vectorial**: Una serie de funciones y coordenadas matemáticas que definen la posición, forma, color, etc. Un dispositivo para mostrar una imagen vectorial debe que *resolver* ecuaciones ya que no hay pixeles representando una forma y por lo tanto no hay niveles de aumento donde una imagen vectorial muestre pixeles porque no los tiene, tiene una resolución infinita.

Las imágenes vectoriales son la mejor forma para representar formas simples ya que el tamaño del archivo será ser menor y la calidad mejor, como logotipos o iconos. Sin embargo, para representar imágenes con mucho detalle, colores, etc. como por ejemplo una imagen del mundo real se debe utilizar una imagen en mapa de bits.

SVG

Los gráficos SVG son un formato de imagen vectorial utilizado inmensamente en sitios web. Es un formato de estándar abierto desarrollado por World Wide Web Consortium y utiliza el estándar XML por lo que tiene alta compatibilidad en sitios HTML (también basado en XML) y es posible modificar gráficos SVG con CSS y/o JavaScript.

Puede utilizarse en web como cualquier otra imagen: utilizando la etiqueta <img> pero los usos más interesantes de SVG son incorporándolos al HTML directamente:

```
.circle1 {
  fill: green;
}

.circle2 {
  fill: blue;
  stroke: red;
  stroke-width: 4;
}

.circle3 {
  fill: aquamarine;
}

<body>
  <p>Tres elipses</p>
  <svg height="150" width="500">
    <ellipse cx="240" cy="100" rx="220" ry="30" class="circle1" />
    <ellipse cx="220" cy="70" rx="190" ry="20" class="circle2" />
    <ellipse cx="210" cy="45" rx="170" ry="15" class="circle3" />
  </svg>
  <!-- Más ejemplos de SVGs simples https://www.w3schools.com/graphics/svg_examples.asp -->
</body>
```

La etiqueta <svg> funciona como un elemento más en HTML. Dentro de ella se mezclan etiquetas y sus atributos propios de SVG con otros de HTML (como class), haciendo de esta manera que las etiquetas del SVG puedan ser transformadas (y por lo tanto la imagen) de la misma manera que se transforman elementos HTML, como por ejemplo con la utilización de CSS.

Sprite sheets

SVG permite la utilización de sprite sheets que en determinados casos hace más sencillo trabajar con este tipo de imágenes. Un sprite sheet es una imagen grande (puede ser vectorial o de bitmap) compuesta por otras imágenes más pequeñas que se utiliza con el fin de ahorrar memoria, mejorar el rendimiento y reducir los costes de mantenimiento.

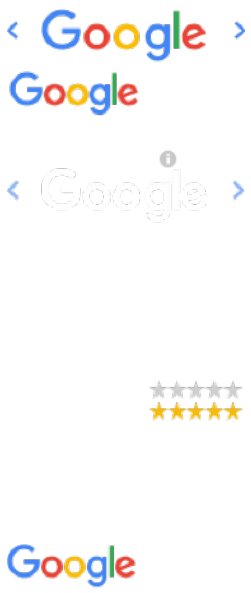

Figura 23. Sprite sheet utilizado en el buscador Google. Fuente: www.google.com

En el siguiente ejemplo, se crea un sprite sheet de svg con un círculo y un cuadrado para posteriormente utilizar el círculo en el documento utilizando par.

```
  <body>
    <!-- declaración del spritesheet con dos elementos, un círculo y un cuadrado -->
    <svg>
      <symbol id="circle" viewBox="0 0 100 100">
        <circle cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="red" />
      </symbol>
      <symbol id="rectancle" viewBox="0 0 100 100">
        <rect width="100" height="100" fill="blue" />
```

```
      </symbol>
    </svg>

    <!-- un svg normal en el que se usa el círculo declarado antes, usando
para ello la etiqueta use con un enlace a circle -->
    <svg>
      <use xlink:href="#circle"></use>
    </svg>
  </body>
```

Con la herramienta IcoMoon se pueden crear de manera sencilla spritesheets tanto de mapa de bits como de SVG (además de fuentes iconográficas).

Tema 9. Librerías interesantes

9.1. Introducción

Al pensar en un sitio web, se puede apreciar que gran parte de ellos comparten partes de interfaz: una modal, un desplegable o una alerta. Una máxima de la programación dice que **no hay que reinventar la rueda**, y es por ello por lo que si gran parte de sitios web comparten interfaz ¿por qué no deben compartir código?

Y ya no solo código de elementos de la interfaz, al final todos los sitios web se enfrentan a los mismos retos, todos sirven un sitio web desde un servidor y debe ser consumido por dispositivos por todo el mundo, por lo que hay librerías, frameworks y en definitiva: código ya hecho al que los maquetadores y/o programadores pueden recurrir para construir sus interfaces.

Estas librerías ya no solo son útiles para utilizarlas, su estudio por parte de otros programadores/maquetadores enriquecen el producto final, aunque no se utilicen de manera directa.

9.2. Bootstrap

¿Qué es Bootstrap?

Bootstrap (https://getbootstrap.com/) se define a sí mismo como un kit de herramientas open source para desarrolladores front end. Al utilizar Bootstrap un maquetador y/o programador front end podrá empezar proyectos de manera más sencilla y obtener resultados más rápido ya que este conjunto de utilidades facilita

su trabajo. Estas utilidades son, por ejemplo: un reset de navegadores, una grid, controles de formulario, componentes reutilizables como dropdowns, modales, etc. Bootstrap puede ser una buena opción para prototipados rápidos o para sitios en los que no se requiere una alta personalización del diseño, ya que precisamente todo ese conjunto de utilidades y componentes ya predefinidos puede ser un problema si lo que se busca es una alta personalización, será entonces, una mejor idea hacer estas utilidades y componentes desde cero en lugar de personalizar lo que provee Bootstrap.

Aun así, se utilice en un proyecto o no, al ser open source se puede consultar el código fuente y es interesante estudiar la manera en la que está programado, ver patrones y comprobar las decisiones de arquitectura que han tomado con el fin de aprender y aplicar todo ello a proyectos de front end.

Inicio rápido de Bootstrap 5

Bootstrap en el fondo es código JavaScript y código CSS, se distribuye de múltiples maneras dependiendo del grado de integración que se necesite en el proyecto.

La manera más sencilla de empezar a utilizar Bootstrap en un proyecto es utilizar el JavaScript y CSS compilado:

```
<!doctype html>
  <html lang="en">
    <head>
      <!-- Required meta tags -->
      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <!-- Bootstrap CSS -->
      <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.
css"                  rel="stylesheet"                  integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
```

```
      <title>Hello, world!</title>
    </head>
    <body>
      <h1>Hello, world!</h1>

      <!-- Bootstrap JavaScript -->
      <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle
.min.js"                                                integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>
    </body>
  </html>
```

Este Starter template (https://getbootstrap.com/docs/5.1/getting-started/introduction/#starter-template) añade al documento HTML todo Bootstrap: todas las utilidades, componentes, etc. se añadirán y estarán disponibles sin mucha capacidad de personalización.

Layout

El layout de Bootstrap representa la culminación de décadas de prueba y error tanto de desarrolladores de Bootstrap como de otros desarrolladores utilizándolo. Su filosofía de layout de contenedor, filas y columnas es ampliamente aceptada y utilizada (aunque no se utilice Bootstrap).

Breakpoints

Una de las decisiones que Bootstrap toma es la de acotar todas las resoluciones posibles de los dispositivos en 6 tamaños preestablecidos:

Breakpoint	Infijo de clase	Dimensiones
extra small	Ninguno	< 576px
small	sm	≥ 576px
medium	md	≥ 768px
large	lg	≥ 992px
extra large	xl	≥ 1200px
extra extra large	xxl	≥ 1400px

Tabla 3. Relación entre tamaños de pantalla y nombre del punto de corte. Fuente: (Team, 2022).

Internamente utiliza media queries con esos pixels establecidos, pero a la hora de utilizarlo abstrae esa complejidad y lo único que el maquetador tiene que pensar es en 6 tamaños de pantalla posible.

Containers

Un contenedor es el elemento más básico de un layout en Bootstrap, como su propio nombre indica hace de contenedor y salvo que se desactive, establece un ancho máximo y centra el contenido en la pantalla.

```
  <!-- Este contenedor centrará en pantalla aquello que contenga y establecerá un ancho máximo para cada punto de corte -->
  <div class="container">
    <!-- aquí va el contenido -->
  </div>

  <!-- Con el sufijo de punto de corte, el contenedor empezará a establecer un tamaño máximo a partir de ese tamaño indicado -->
  <div class="container-sm">
    <!-- aquí va el contenido -->
  </div>

  <!-- Este contenedor desactiva el ancho máximo y ocupa toda la pantalla -->
  <div class="container-fluid">
    <!-- aquí va el contenido -->
  </div>
```

Filas/columnas

Dentro de un contenedor y utilizando la tecnología flexbox que proporciona CSS, se establecen una serie de utilidades para conseguir dividir el espacio en columnas y de esa manera maquetar de una manera más sencilla y robusta en cualquier tamaño de dispositivo.

Columnas sin ancho predefinido

Al estar basado en flex-box, las columnas tenderán a ocupar todo el espacio disponible, de esta manera, las 3 columnas tendrán el mismo ancho y entre todas ocuparán la fila entera

```
<div class="container">
  <div class="row">
    <div class="col">Columna 1</div>
    <div class="col">Columna 2</div>
    <div class="col">Columna 3</div>
  </div>
</div>
```

Columnas con ancho predefinido

Bootstrap por defecto define 12 columnas por fila y estas se pueden *juntar* para conseguir columnas que ocupen X columnas.

Figura 24. Fila de 12 columnas con sistema de layout de Bootstrap Fuente: elaboración propia.

```
<div class="container">
  <div class="row">
    <div class="col-9">Columna que ocupa 9 columnas de ancho</div>
    <div class="col-1">Columna que ocupa 1 columna de ancho</div>
    <div class="col-2">Columna que ocupa 2 columnas de nacho</div>
  </div>
</div>
```

Incluso, ese espacio que reserva una columna dentro de una fila puede cambiar dependiendo del punto de corte. Esta característica es muy potente y es el *secreto* de la maquetación web responsive utilizando el grid de Bootstrap:

```
<div class="container">
  <div class="row">
    <div class="col-9 col-md-6">
      Columna que ocupa 9 columnas de ancho por defecto
      pero a partir de tamaños de pantalla ≥768px ocupa 6 columnas
    </div>
    <div class="col-1 col-md-2">
      Columna que ocupa 1 columna de ancho por defecto
      pero a partir de tamaños de pantalla ≥768px ocupa 2 columnas
    </div>
    <div class="col-2 col-md-4">
      Columna que ocupa 2 columnas de ancho por defecto
      pero a partir de tamaños de pantalla ≥768px ocupa 4 columnas
    </div>
  </div>
</div>
```

Gutter

Gutter es el espacio que se deja entre columnas, por defecto no hay espacio, pero puede resultar interesante para el diseño dejar aire entre elementos. Se añade en el elemento que hace de fila y establece el mismo gutter para todos los elementos.

- **g-*:** Para gutter tanto vertical como horizontal.
- **gy-*:** Para gutter vertical.
- **gx-*:** Para gutter horizontal.
- El * es un número entre el 0 y el 5 y corresponde al índice de espaciado definido por Bootstrap. (https://getbootstrap.com/docs/5.1/utilities/spacing/).

```
<div class="container">
  <!-- se añade un espaciado horizontal de 5 entre elementos -->
  <div class="row gx-5">
    <div class="col">
      Columna
    </div>
    <div class="col">
      Columna
    </div>
  </div>
</div>
```

Formularios

Los controles de formulario de HTML nativos son estilados por Bootstrap siguiendo sus directrices de diseño y además se añaden algunas utilidades para maquetar formularios de una manera sencilla, rápida y cumpliendo estándares.

Figura 25. Ejemplo de formulario utilizando Bootstrap Fuente : (Team, Bootstrap, 2021).

```
  <form>
    <div class="mb-3">
      <label       for="exampleInputEmail1"       class="form-label">Email
address</label>
      <input   type="email"   class="form-control"   id="exampleInputEmail1"
aria-describedby="emailHelp">
      <div  id="emailHelp"  class="form-text">We'll  never  share  your  email
with anyone else.</div>
    </div>
    <div class="mb-3">
      <label            for="exampleInputPassword1"            class="form-
label">Password</label>
      <input              type="password"              class="form-control"
id="exampleInputPassword1">
    </div>
    <div class="mb-3 form-check">
      <input type="checkbox" class="form-check-input" id="exampleCheck1">
      <label    class="form-check-label"    for="exampleCheck1">Check    me
out</label>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
  </form>
```

- **form-control:** Esta clase es utilizada para los controles de texto como input o textarea. Además, se definen las clases form-control-lg y form-control-sm para variar el tamaño.
- **form-label:** Esta clase se utiliza para estilar las etiquetas label que acompañan a un control de formulario.
- **form-select:** Estila controles de tipo select. También tiene definición de tamaño como control: form-select-lg, form-select-sm.
- **form-check:** Para estilar radios y checkboxes. Conforma una serie de clases como form-check-input para la etiqueta input y form-check-label para la etiqueta label.

Componentes

Los componentes es la parte más interesante de Bootstrap. Un componente es una pieza pequeña de una interfaz de usuario que resuelve un cometido específico.

El listado de componentes de Bootstrap es muy amplio y cubre necesidades que se requieren en la mayoría de los sitios web. Son componentes que ampliamente se utilizan y ya están en el modelo mental del usuario, es decir, son reconocibles por este y no suponen una barrera para empezar a utilizarlos, por ejemplo, un modal, un acordeón son componentes de la colección de Bootstrap que el usuario ya sabe cómo interactuar con ellos.

Consultando la documentación de Bootstrap acerca de los componentes, un maquetador/programador ya tiene código listo para usar en su proyecto (siempre que tenga Bootstrap disponible en él) y al igual que otras características de Bootstrap, el código de los componentes está más que probado en diferentes navegadores y por millones de usuarios.

Un componente se compone de código CSS y habitualmente de código JavaScript que le dota de interactividad y ya sea utilizando clases extra o definiendo parámetros específicos para JavaScript se puede alterar el comportamiento de un componente, siempre que esté especificado en su documentación.

Ejemplos de customización:

```
  <!-- El componente alert tiene customización de color, añadiendo la clase CSS correspndiente se puede alterar la finalidad de la alerta, por ejemplo, color verde para una alerta de que algo se ha conseguido sin errores (alert-success) -->
  <div class="alert alert-primary" role="alert">
    A simple primary alert with <a href="#" class="alert-link">an example link</a>. Give it a click if you like.
  </div>
```

```
  <div class="alert alert-secondary" role="alert">
    A simple secondary alert with <a href="#" class="alert-link">an example link</a>. Give it a click if you like.
  </div>
  <div class="alert alert-success" role="alert">
    A simple success alert with <a href="#" class="alert-link">an example link</a>. Give it a click if you like.
  </div>

  <!--
  Para utilizar el componente tooltip es necesario "engancharlo" con JavaScript, para ello se utiliza el atributo data-bs-toggle.
  Además, es posible personalizar su comportamiento, en este caso se puede especificar la dirección a la que abre el tooltip, para ello el código JavaScript de Bootstrap lee el atributo data-bs-placement cuyos valores pueden ser: top, right, bottom o left.
  -->
  <button type="button" class="btn btn-secondary" data-bs-toggle="tooltip" data-bs-placement="top" title="Tooltip on top">
    Tooltip on top
  </button>
```

9.3. Tailwind CSS

Tailwind CSS es un framework de CSS que sigue la filosofía utility-first. Tailwind CSS provee un conjunto de utilidades primitivas que en su mayor medida corresponden a atributos CSS. Esto puede parecer una mala idea, y durante mucho tiempo se tendió a prácticamente lo contrario (metodologías CSS semánticas) pero este tipo de filosofía aporta muchas ventajas como:

- Ficheros CSS más pequeños.
- Mantenimiento mucho más sencillo en proyectos grandes.
- No hace falta estar decidiendo nombres de bloques CSS ni gastar el tiempo siguiendo ninguna metodología.

Al igual que otros framework o conjuntos de utilidades de front end, aunque no se utilicen directamente en un proyecto, siempre resulta interesante estudiarlos para comprender los razonamientos y patrones que se utilizan que quizá puedan resultar útiles en el futuro.

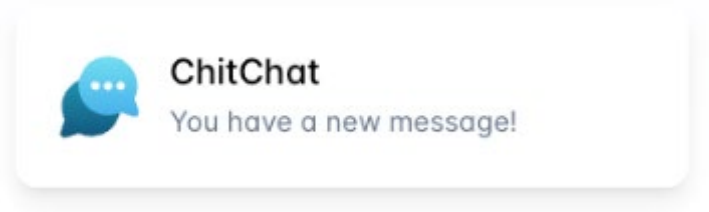

Figura 26. Ejemplo de maquetación de componente Fuente: (Tailwind Labs Inc., 2022).

Ejemplo de su documentación (https://tailwindcss.com/docs/utility-first) adaptado a metodología BEM:

```
  <!-- Metodología CSS BEM -->
  <div class="chat-notification">
    <div class="chat-notification__logo-wrapper">
      <img class="chat-notification-logo" src="/img/logo.svg" alt="ChitChat
Logo">
    </div>
    <div class="chat-notification__content">
      <h4 class="chat-notification__title">ChitChat</h4>
      <p class="chat-notification__message">You have a new message!</p>
    </div>
  </div>

  <style>
    .chat-notification {
      display: flex;
      max-width: 24rem;
      margin: 0 auto;
      padding: 1.5rem;
      border-radius: 0.5rem;
      background-color: #fff;
      box-shadow: 0 20px 25px -5px rgba(0, 0, 0, 0.1), 0 10px 10px -5px
rgba(0, 0, 0, 0.04);
```

```
    }
    .chat-notification__logo-wrapper {
      flex-shrink: 0;
    }
    ...
  </style>
```

Maquetación del mismo componente utilizando TailwindCSS:

```
  <!-- Utilizando TailwindCSS -->
  <div class="p-6 max-w-sm mx-auto bg-white rounded-xl shadow-lg flex items-
center space-x-4">
    <div class="shrink-0">
      <img class="h-12 w-12" src="/img/logo.svg" alt="ChitChat Logo">
    </div>
    <div>
      <div class="text-xl font-medium text-black">ChitChat</div>
      <p class="text-slate-500">You have a new message!</p>
    </div>
  </div>
```

Al dar TailwindCSS una batería de utilidades CSS hace que el maquetador no escriba CSS directamente si no que estile directamente en el HTML.

En una metodología como BEM, los estilos se escriben de manera que están relacionados directamente con el componente que están estilando y que rara vez puedan ser reutilizables, sin embargo, utility first es todo lo contario, son muchas utilidades, pero todas ellas tienden a ser extremadamente reutilizables por cualquier componente.

Probando Tailwind CSS

La integración de Tailwind CSS es mucho más profunda y customizable que otros framework. La decisión de utilizar esta librería se debe tomar al empezar un proyecto

y su integración es tal que será muy difícil dejar de utilizarla y quitarla completamente del proyecto.

La manera más sencilla de empezar a utilizar Tailwind CSS y probar sus características es la integración con su Play CDN (https://tailwindcss.com/docs/installation/play-cdn). Esta forma de utilizar Tailwind CSS no está recomendada para proyectos reales, pero es una buena manera de empezar a escribir código y que el maquetador se haga una idea de lo que es utilizar este framework.

Para ello, simplemente hay que añadir una etiqueta script y otra con un JSON para la configuración (se puede consultar en: https://tailwindcss.com/docs/configuration) y ya se pueden empezar a utilizar las utilidades:

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://cdn.tailwindcss.com"></script>
    <script>
      tailwind.config = {
        theme: {
          extend: {
            colors: {
              clifford: '#da373d',
            }
          }
        }
      }
    </script>
  </head>
  <body>
    <h1 class="text-3xl font-bold underline">
      Hello world!
    </h1>
```

```
    </body>
  </html>
```

Prefijos de las utilidades

Todas las utilidades de Tailwind pueden aplicarse condicionalmente añadiendo un modificador al principio de la utilidad, de tal manera que el cometido de esa utilidad solo estará disponible cuando se cumpla el modificador:

- **Modificadores de pseudoclase:** hover, focus o first-child
- **Modificadores de punto de corte:** sm, md, lg, xl o 2xl.
- **Modificador de dark mode:** dark.

Por ejemplo, color-white es la utilidad para añadir color de texto blanco, sin embargo, este comportamiento se puede modificar con prefijos:

```
  <!-- al poner el cursor encima, este botón tendrá color de texto negro --
>
  <a class="color-white hover:color-black">
    Botón
  </a>

  <!-- este div tendrá color de texto negro cuando el ancho de pantalla sea
mayor que 768px -->
  <div class="color-white md:color-black">
    ...
  </div>

  <!-- este div tendrá color de texto blanco cuando esté activado el dark
mode -->
  <div class="color-black dark:color-white">
    ...
  </div>
```

9.4. Referencias bibliográficas

Tailwind Labs Inc. (2022). *Utility-First Fundamentals*. Obtenido de Tailwindcss: https://tailwindcss.com/docs/utility-first

Team, B. (7 de diciembre de 2021). *Bootstrap.* Obtenido de Forms: https://getbootstrap.com/docs/5.1/forms/overview/

Team, B. (31 de enero de 2022). *Breakpoints.* Obtenido de Bootstrap: https://getbootstrap.com/docs/5.1/layout/breakpoints/

Tema 10. Procesador SASS

10.1. Introducción

SASS (https://sass-lang.com/) Syntactically Awesome Stylesheets es un preprocesador de código CSS, esto quiere decir que tiene una sintaxis no estándar con CSS, pero es capaz de convertirse en código CSS válido mediante un proceso.

El lenguaje CSS nativo que leen los navegadores no ha evolucionado tanto como el resto de las tecnologías web y gracias a SASS (metalenguaje de CSS) podemos tener de cierta manera recursos que en CSS válido no tenemos como funciones, módulos, etc.

No todo son ventajas en SASS, al no estar escribiendo código válido al navegador añade una capa de complejidad al proyecto ya que es necesario un procesador que transforme código SASS en código CSS. Afortunadamente SASS es prácticamente un estándar en el desarrollo web por lo que este proceso de traducción está integrado en la mayoría de los procesos de compilado de aplicaciones web.

Por ejemplo, el siguiente código SCSS se compila a CSS:

```
/* Código SCSS */
$color-bg: #e60000;
$color-text: #ededed;
.caja {
  background-color: $color-bg;

  .texto {
    color: $color-text;
  }
}
/* Código CSS */
```

```
.caja {
  background-color: #e60000;
}
.caja .texto {
  color: #ededed;
}
```

10.2. Instalación y uso de SASS

Aunque al final el código del compilador es el mismo, se puede utilizar SASS de diferentes maneras, más amigables para el maquetador o con mayor integración con el entorno de desarrollo:

- **Aplicaciones de escritorio**: Muy sencillo para empezar, pero con nula integración con entorno de desarrollo. Por ejemplo: Koala (http://koala-app.com).
- **Mediante línea de comandos**: La opción más recomendada, aunque un poco más difícil de instalar, integra perfecto con entornos de desarrollo. Instrucciones oficiales de instalación por la línea de comandos (https://sass-lang.com/install).
- **Plugin en IDE**: La opción recomendada para fines académicos donde de una manera automática el código SASS que se encuentre en la carpeta del proyecto se compilará a CSS listo para usar. Por ejemplo un plugin para el IDE Visual Studio Code: Live Sass Compile (https://marketplace.visualstudio.com/items?itemName=ritwickdey.live-sass).

Sintaxis SCSS/SASS

SASS como preprocesador de código CSS tiene dos sintaxis: scss y sass. Ambas se compilan con el mismo compilador y dado un código con sintaxis scss se podría convertir en sass y viceversa. Simplemente sass prescinde de las llaves: {} (se utiliza la indentación para anidar) y del carácter; al final de cada declaración.

```scss
/* SCSS */
$color-bg: #e60000;
$color-text: #ededed;

.caja {
  background-color: $color-bg;

  .texto {
    color: $color-text;
  }
}
```

```sass
/* SASS */
$color-bg: #e60000
$color-text: #ededed

.caja
  background-color: $color-bg

  .texto
    color: $color-text
```

Actualmente es más común la utilización de la sintaxis scss ya que está más cercana a CSS y el suprimir las llaves {} por indentación es bastante propenso a errores. En esta documentación se utilizará la sintaxis scss.

10.3. Variable en SASS

Al igual que en cualquier lenguaje de programación, SASS dispone de variables que hace más sencillo y robusto escribir código SASS (que posteriormente compilará a CSS).

La declaración de una variable se hace de la misma manera que una declaración CSS, pero en lugar de propiedad, se indica el nombre de la variable empezando por el carácter: $ que en SASS significa que es una variable.

```
/* SCSS */
  $color-bg: #e60000;
  .caja {
    background-color: $color-bg;
  }


  /* CSS */
  .caja {
    background-color: #e60000;
  }
```

Valores default

Si una variable se declara 2 veces, el comportamiento habitual es que la segunda declaración sobrescriba a la primera:

```
  /* SCSS */
  $color-bg: blue; /* Primera declaración */
  $color-bg: #e60000; /* Segunda declaración */
  .caja {
    background-color: $color-bg;
  }

  /* CSS */
  .caja {
    background-color: #e60000;
  }
```

Sin embargo, SASS dispone de una etiqueta !default que previene de esta doble declaración. Solo escribirá el valor en la variable si esa variable **no estaba declarada antes**:

```
/* SCSS */
$color-bg: blue; /* Primera declaración */
$color-bg: #e60000 !default; /* Segunda declaración pero con !default */

.caja {
  background-color: $color-bg;
}

/* CSS */
.caja {
  background-color: blue;
}
```

Ámbito de las variables

Otra característica heredada de lenguajes de programación es el ámbito de las variables.

En SASS el ámbito de una variable declarada fuera de un bloque (en scss fuera de llaves {}) es considerada global y es accesible desde cualquier sitio. Sin embargo, una variable declarada dentro de un bloque solo será accesible para ese bloque.

```
/* SCSS */
$color: #e60000;

.caja-1 {
  $color-caja-1: blue;
  background-color: $color; /* OK */
  color: $color-caja-1; /* OK */
}
```

```
.caja-2 {
  $color-caja-2: red;
  $background-color: $color; /* OK */
  color: $color-caja-1; /* Esto fallará ya que el color-caja-1 no está en el ámbito */
}
```

10.4. Anidación

La anidación de SASS es una herramienta muy potente que permite anidar estilos que corresponden a elementos hijos dentro del padre. Para el maquetador reducirá el código que tiene que escribir y permite organizar el código SASS visualmente en *bloques,* pero se debe tener precaución ya que se corre el riesgo de utilizar demasiados selectores lo cual es una mala práctica de CSS.

Para determinados selectores se necesita utilizar una referencia al padre: & ya que el anidamiento por defecto de SASS es el de CSS dejando un espacio entre selectores (hijo dentro de un padre).

```
/* SCSS */
.listado {
  display: inline-flex;

  li {
    background-color: blue;
  }

  a {
    text-decoration: none;

    &:hover {
      color: green;
    }
  }
```

```
}

/* CSS */
.listado {
  display: inline-flex;
}
.listado li {
  background: color: blue;
}
.listado a {
  text-decoration: none;
}
.listado a:hover {
  color: green;
}
```

10.5. Modularización de código

Aunque en CSS ya existe la regla @import haciendo que un fichero CSS requiera otro fichero CSS, esto hace que el navegador tenga que hacer múltiples llamadas para conseguir todo el código. Sin embargo, SASS hace ese proceso en el momento de la compilación, consiguiendo juntar varios ficheros CSS en uno solo.

Esta característica de SASS hace que se pueda escribir código CSS de manera modular lo que hace más sencillo de mantener, manejar y entender el código CSS de un proyecto.

```
/* Fichero _variables.scss */
$blue: #0000FF;
$gutter: 10px;
```

```scss
/* Fichero _caja.scss */
.caja {
  background-color: $blue;
}

/* Fichero _grid.scss */
.grid {
  display: flex;
  gap: $gutter;
}

/* Fichero index.scss */
@import variables;
@import caja;
@import grid;

/* Código CSS tras compilar index.scss */
.caja {
  background-color: #0000FF;
}
.grid {
  display: flex;
  gap: 10px;
}
```

En el anterior ejemplo se juntan muchos conceptos relacionados con la característica @import de SASS:

- **Arquitectura de ficheros:** Gracias a la modularización y siguiendo el principio de responsabilidad única, es una buena práctica tener pequeños ficheros scss que resuelvan algo muy concreto (como por ejemplo un grid) y que se importen todos desde un fichero índice. De esta manera cada trozo de código scss puede ser reutilizable en el mismo o en otros proyectos y hace que el código sea más sencillo de leer por el maquetador al tener únicamente en pantalla aquello que cumple un cometido específico.

- **Carácter _ delante del nombre de fichero:** Esto es una convención de SASS, este carácter delante del nombre en el fichero hace que un compilador de SASS nunca compile ese fichero, pasará al siguiente y se utiliza en ficheros creados para ser importados desde otro, como es en este ejemplo.

- **Ámbito de variables:** El @import de SASS respeta el ámbito de variables, utilizar un @import es como si se copiase/pegase el código del fichero que se está enlazando por lo que las variables de un fichero están disponibles en otro. De esta manera, es una buena práctica tener un fichero único de variables, importarlo al principio y ya está disponible de manera global en todo el proyecto, pero centralizado en un único fichero especializado en albergar variables.

10.6. Mixins

Los mixins de SASS pueden ser comparables a funciones en un lenguaje de programación. Nacen con la idea de encapsular declaraciones CSS con el fin de reutilizarlas en otras reglas, incluso permiten parámetros al igual que las funciones.

```
/* SCSS */
@mixin no-listado {
  list-style: none;
  margin: 0;
  padding: 0;
}

.menu {
  @include no-listado;

  a {
    background-color: red;
  }
}
```

```
/* CSS */
.menu {
  list-style: none;
  margin: 0;
  padding: 0;
  background-color: red;
}

/* SCSS */
@mixin cuadrado($tamaño, $radius: 0) {
  display: inline-block;
  width: $tamaño;
  height: $tamaño;
  border-radius: $radius;
}

.avatar {
  @include cuadrado(20px, 4px);
  background-image: url(avatar.jpg);
}

/* CSS */
.avatar {
  display: inline-block;
  width: 20px;
  height: 20px;
  border-radius: 4px;
  background-image: url(avatar.jpg);
}
```

10.7. Herencia en SASS

El uso de @extend en SASS es comparable con la herencia en lenguajes de programación, al igual que un mixin permite reutilizar código entre reglas CSS, pero la diferencia entre estos es la manera en la que se compila el CSS resultante.

```scss
/* SCSS */
%boton {
  display: inline-block;
  font-size: 16px;
  border-radius: 4px;
  color: white;
}

.boton-ok {
  @extend %boton;
  background-color: green;
}

.boton-ko {
  @extend %boton;
  background-color: red;
}

/* CSS */

/* Esta ese comportamiento de concatenar selectores es gracias al uso de @extend, de tal manera que realmente se comparte el código entre boton-ok y boton-ko */
.boton-ok, .boton-ko {
  display: inline-block;
  font-size: 16px;
  border-radius: 4px;
  color: white;
}

.boton-ok {
  background-color: green;
}

.boton-ko {
  background-color: red;
}
```

10.8. Control de flujo

Otra característica llevada desde los lenguajes de programación a SASS es el control de flujo (etiquetas como if, else, for, each *y* while).

if/else

La sintaxis if/else evalúa una condición que debe ser booleana: verdadero o falso, en caso de que la expresión sea cierta (true) se evaluará la primera parte de la expresión y en caso contrario, se evaluará la segunda parte de la expresión (la correspondiente al else).

```
/* SCSS */
@mixin button($width, $height, $bigRadius) {
  display: inline-block;
  width: $width;
  height: $height;

  @if $bigRadius {
    border-radius: 6px;
  } @else {
    border-radius: 2px;
  }
}

.button-1 {
  @include button(100px, 20px, true);
}

.button-2 {
  @include button(100px, 20px, false);
}
```

```
/* CSS */
.button-1 {
  display: inline-block;
  width: 100px;
  height: 20px;
  border-radius: 6px;
}
.button-2 {
  display: inline-block;
  width: 100px;
  height: 20px;
  border-radius: 2px;
}
```

each

Esta expresión maneja el flujo de tal manera que recorre una lista de elementos, se puede entender como que para cada elemento se ejecuta la parte incluida en el @each. A diferencia de otras instrucciones de control de flujo, @each recorrerá siempre uno a uno todos los elementos de un listado, empezando por el primero y acabando por el último.

```
/* SCSS */

$font-weights: 300, 400, 700;

@each $weight in $font-weights {
  .heading-#{$weight} {
    font-weight: $weight;
  }
}

/* CSS */

.heading-300 {
  font-weight: 300;
}
```

```
.heading-400 {
  font-weight: 400;
}

.heading-700 {
  font-weight: 700;
}
```

for

La expresión @for es utilizada para hacer de contador entre un número de inicio y un número final de manera que el flujo recorre todos los números intermedios de manera automática. Además, deja disponible un índice para poder utilizar el número correspondiente a cada vuelta del bucle.

```
/* SCSS */

/* Deja en el índice ($i) números de entre el 1 y el 4 (ambos inclusive) en cada vuelta del bucle */
@for $i from 1 to 5 {
  .gap-#{$i} {
    gap: #{$i}px;
  }
}
/* CSS */

.gap-1 {
  gap: 1px;
}
.gap-2 {
  gap: 2px;
}
.gap-3 {
  gap: 3px;
}
.gap-4 {
  gap: 4px;
}
```

while

La expresión @while deja el flujo en bucle hasta mientras su condición se mantenga verdadera, en el momento en el que en una vuelta la condición sea considerada falsa, el flujo saldrá del while.

Las utilizaciones de bucles while es peligrosa ya que si por la razón que sea la condición de salida nunca evalúa false, el flujo quedará infinitamente atrapado en el while, dándose así lo que se llama como **bucle infinito** y dando lugar a problemas. Afortunadamente al tratarse de un error habitual los compiladores y sistemas operativos están protegidos contra este tipo de fallos.

```
/* SCSS */

.progress-bar {
  height: 10px;
  background: blue;
}

$i: 1;
@while $i <= 5 {
  .progress-bar:nth-child(#{$i}) {
    width: 10px * $i;
  }
  $i: $i + 1; /* MUY IMPORTANTE, si no, el @while se quedará en bucle infinito */
}

/* CSS */

.progress-bar {
  height: 10px;
  background: blue;
}
.progress-bar:nth-child(1) {
  width: 10px;
```

```
}
.progress-bar:nth-child(2) {
  width: 20px;
}
.progress-bar:nth-child(3) {
  width: 30px;
}
.progress-bar:nth-child(4) {
  width: 40px;
}
.progress-bar:nth-child(5) {
  width: 50px;
}
```

A fondo

Fix Your Site With the Right DOCTYPE!

Zeldman, J. (202). *Fix Your Site With the Right DOCTYPE!* [artículo en línea]. https://alistapart.com/article/doctype/

Aunque ya obsoleto, en este artículo se puede consultar los antiguos métodos y formas del Doctype para documentos HTML. Conviene echarle un vistazo para reconocerlos ya que siguen vigentes en algunos proyectos de la actualidad.

The Open Graph protocol

Meta. (2010). *The Open Graph protocol* [sitio web] https://ogp.me/

Este prototolo de metadatos propuesto por Meta (anteriormente Facebook) está basado en la simplicidad para comunicar datos a través de diferentes tecnologías. Está presente en gran número de sitios web ya que es utilizado para mostrar contenido relevante de la página en sitios de la empresa como Facebook o Whatsapp.

Expresiones regulares

Knator, I. (2022). *Expresiones regulares* [Sitio web]. https://es.javascript.info/regular-expressions

Una expresión regular es una manera para realizar búsquedas en cadenas de caracteres de manera simple, eficaz y muy poderosa. Se utilizan mucho en el ámbito de la programación y es conveniente tenerlas en cuenta.

Métodos de petición HTTP

MDN contributors. (2022). *Métodos de petición HTTP* [documentación en línea]. https://developer.mozilla.org/es/docs/Web/HTTP/Methods

Ya sea en el envío de formularios o de cualquier petición HTTP conviene tener en cuenta los tipos de petición HTTP para utilizar la correcta para cada necesidad.

Website Forms Usability: Top 10 Recommendations

Whitenton, K. (2016). *Website Forms Usability: Top 10 Recommendations* [artículo en línea]. https://www.nngroup.com/articles/web-form-design/

No solo se trata solo de codificar formularios, es necesario que los usuarios puedan interactuar con ellos. Aquí es donde maquetación HTML y diseño de experiencia de usuario se juntan y es conveniente que un maquetador tenga ciertas nociones de esta.

Referencia CSS

MDN Contributors. (2022). *Referencia CSS. MDN Web Docs.* [documentación en línea] https://developer.mozilla.org/es/docs/Web/CSS/Reference

Este listado es habitualmente actualizado con todas las propiedades y palabras claves disponibles en CSS. Se debe tomar como un recurso para consulta, no como material de estudio.

Weight & Proportion

Haley A. Weight & Proportion. *Fonts.com*. [artículo en línea] https://www.fonts.com/content/learning/fontology/level-1/type-anatomy/weight-and-proportion

Este artículo explica diferentes propiedades que tiene una misma tipografía. La tipografía es un pilar fundamental del diseño web y conocer su lenguaje es relevante ya que es exactamente el mismo lenguaje utilizado para codificar en CSS.

Weight & Proportion

World Wide Web Consortium (W3C). *Syntax and basic data types. CSS 2.1 Specification.* [documentación en línea] https://www.w3.org/TR/CSS2/syndata.html#length-units

Especificación de CSS, en la sección 4.3.2. se explica los tamaños de fuente en qué se han basado para especificar el *reference pixel*.

A Complete Guide to Flexbox

Coyier, C. (2021). *A.Complete Guide to Flexbox* [artículo en línea]. https://css-tricks.com/snippets/css/a-guide-to-flexbox/

La mejor recopilación de documentación y ejemplos para aprender y consultar el posicionamiento flex de CSS. Todas las propiedades explicadas con tanto en texto como con ilustración y además otros recursos y ejemplos.

Object Oriented CSS

Rodas, G. (2020). *Object Oriented CSS* [documentación en línea]. https://github.com/stubbornella/oocss/wiki

Documentación en profundidad, recursos y todo tipo de ejemplos de uso de esta metodología de CSS.

BEM

BEM Core Team. (2017). *Get BEM* [documentación en línea]. http://getbem.com/

Documentación en profundidad, recursos, comparativa con otras metodologías y un FAQ sobre esta metodología de CSS.

Grid system

Bootstrap Team. (2021). Grid system. *Documentación Bootstrap 5.1*. https://getbootstrap.com/docs/5.1/layout/grid/

Ampliamente utilizado y de los primeros en utilizar la filosofía de dividir el espacio disponible en 12 columnas, pero con posibilidad de adaptarse a diferentes tamaños de pantalla. Se utilice Bootstrap o no, el estudio de este grid puede ser muy beneficioso a la hora de entender y aplicar estas ideas en un proyecto web.

@media

MDN Contributors. (2022). @media. *CSS: Cascading Style Sheets*. https://developer.mozilla.org/en-US/docs/Web/CSS/@media#media_features

Documentación de @media en el que se pueden consultar todas las características que puede detectar esta funcionalidad de CSS como el ancho o alto del dispositivo, densidad de píxeles u orientación.

Test

1. ¿De que no se debe encargar un documento HTML?

   A. De albergar el contenido.

   B. De la forma en la que se representa el contenido.

   C. De dar semántica al contenido.

   D. De permitir la accesibilidad del contenido.

2. Para albergar la información de un evento como título, fecha, lugar, artista, url de compra de entradas, etc. ¿Qué etiqueta semántica es la correcta?

   A. <article>.

   B. <section>.

   C. <address>.

   D. <aside>.

3. ¿Dónde se dé validar los formularios de un sitio web?

   A. En el cliente.

   B. En el servidor.

   C. En cliente y servidor.

   D. Se debe dejar la responsabilidad al usuario, pero si informarle de las consecuencias de enviar mal los datos.

4. Si el tamaño de fuente por defecto en un navegador es de 16px. ¿Qué tamaño tendrá un elemento al que se le ha aplicado la declaración font-size: 2rem?

   A. 16px.

   B. 18px.

   C: 32px.

   D: 256px.

5. ¿Cuál es los siguientes selectores tiene más especificidad?

   A. .caja.activa .elemento1 a.

   B. .caja .elemento1 a:hover.

   C. .caja .elemento1 > a:hover.

   D. .caja #destacado > a:hover.

6. Un elemento al que se le ha aplicado por CSS la declaración: display: none ¿Lo podrá leer un robot de un motor de búsqueda o cualquier proceso automatizado de rastreo de contenido?

   A. No, al no aparecer mostrado en un navegador es como si no existiese.

   B. No, al utilizar HTML5 + CSS no tiene acceso.

   C. Si, pero solo si el motor de búsqueda es el de Bing.

   D. Si.

7. ¿Qué significa mobile-first?

   A. Dispositivos móviles son lo único que va a visitar un sitio web, por esa razón ahora son tan populares las Apps móviles.

   B. Priorizar los dispositivos que actualmente y en la mayoría de los casos más tráfico llevan a sitios web.

   C. Media query para detectar únicamente dispositivos móviles.

   D. Técnica muy popular de SEO.

8. Para realizar cambios propiedades animadas de elementos como el color de fondo o su posición ¿Qué técnica es más recomendable?

   A. Intercambiar el contenido por un vídeo que realice el movimiento para posteriormente ocultarlo.

   B. @animation.

   C. Propiedad transition de CSS.

   D. B y C son ciertas.

9. Tailwind CSS está pensado para:

A. Tener una colección de componentes lista para usar.

B. Ser súper parametrizable.

C. Poder añadirse y quitarse rápidamente de un proyecto.

D. Todas son ciertas.

10. ¿Cómo se declara una variable en SASS?

A. Añadiendo var delante del identificador de la variable.

B. Añadiendo _ delante del identificador de la variable.

C. Simplemente escribiendo el identificador de la variable.

D. Añadiendo $ delante del identificador de la variable.