

CS5



Contents

Conceptos básicos de CSS3.....	3
¿Qué es CSS?	3
¿Cómo trabajar con CSS?	4
Hoja de estilos CSS externa	4
Estructura de CSS3.....	6
Estructura de CSS3.....	Error! Bookmark not defined.
Selectores y combinadores.....	8
Pseudoclases y Pseudoelementos	20
Colores.....	40
Modelo Caja y Position	44
Variables	58
Fondos e imágenes.....	60
Fuentes tipográficas	63
Web responsive	67
Display básico	74
Display: Flex.....	76
Sombras y animaciones	93
Filtros.....	99
Recursos	109

Conceptos básicos de CSS3

Imagina una aplicación web como un lienzo en blanco. El código HTML define la estructura básica, como un boceto a lápiz.

Pero para darle **color, forma y vida**, necesitas el pincel mágico de **CSS**.

¿Qué es CSS?

CSS3, siglas de **Cascading Style Sheets** (Hojas de Estilo en Cascada), es la tercera versión de un lenguaje que te permite controlar la apariencia de una aplicación web.

Define cómo se **muestran** los elementos, desde el **color** del texto hasta la ubicación de las **imágenes, tipografías, animaciones** y mucho más. Es como darle instrucciones al navegador web sobre cómo interpretar el código HTML y convertirlo en una experiencia visual atractiva, intuitiva y accesible.

¿Cómo funciona CSS?

Imagina una cascada de estilos que se aplica a cada elemento de la aplicación web.

Puedes definir **reglas** generales para todos los elementos, o reglas específicas para cada tipo de elemento. Por ejemplo, puedes definir que todos los títulos de la página web sean de color azul y tamaño 16px, o que las imágenes tengan un borde de 1px de color gris.

La magia de separar HTML y CSS: ventajas para tu aplicación web y el navegador

Imagina una aplicación web como una obra de arte:

- El **HTML** es el boceto: define la estructura básica, como los cimientos y la forma general.
- El **CSS** es la pintura: da color, forma y estilo a la obra, como los detalles, texturas y la paleta de colores.

Separar el **HTML** y el **CSS** es como separar el boceto de la pintura.

Te permite trabajar en cada uno de forma independiente, optimizando el proceso y el resultado final.

¿Por qué es tan importante esta separación?

- **Flexibilidad y eficiencia:** Modificar el diseño es más sencillo.
- Solo necesitas editar el archivo **CSS**, sin tocar el contenido.
- Puedes cambiar la paleta de colores, la tipografía o el diseño de la página en cuestión de minutos, sin afectar la estructura del sitio web.
- **Mantenimiento simplificado:** Si necesitas realizar un cambio en el diseño a lo largo de todo el sitio, solo necesitas hacerlo en el archivo CSS, lo que te ahorra tiempo y esfuerzo.
- **Escalabilidad:** Adaptar tu sitio web a diferentes dispositivos es más fácil. Puedes crear diferentes estilos para móviles, tablets y escritorios utilizando CSS, sin necesidad de reescribir el código HTML.
- **Mejor legibilidad:** El código **HTML** y **CSS** es más claro y fácil de entender, lo que facilita la tarea de los desarrolladores web. Nos podemos olvidar de la etiqueta y atributo **style**, así no mezclaremos nunca un lenguaje con el otro.

¿Cómo trabajar con CSS?

La manera tradicional de trabajar con CSS es a través del

elemento style

dentro de un HTML, por ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document</title>
    <style>
      div {
        background: black;
        color: whitesmoke;
      }
    </style>
  </head>
  ...
</html>
```

O también mediante el atributo **style** en cada uno de los elementos en HTML:

```
<p style="color:blue">Este párrafo es de color azul</p>
```

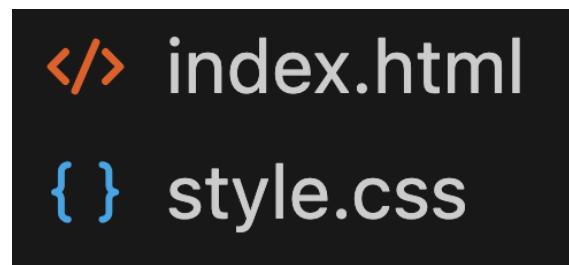
Pero no son la manera **recomendada** de estilar una aplicación, ya que mezcla nuestros dos lenguajes en un mismo fichero y hace que sea una manera poco escalable y repartida de trabajar.

Incluso en nuestro código nos vamos a liar si mezclamos varios lenguajes en uno solo. Por ello, la manera recomendada de trabajar es a través de un **archivo CSS externo**:

Hoja de estilos CSS externa

Vamos a enlazar nuestra primera hoja de estilos con un documento HTML:

Paso 1: Crea un archivo CSS bajo el nombre "**style.css**" paralelo a "**index.html**". Es decir, en la misma carpeta. Recordemos que cada proyecto tendrá una carpeta con, de momento, estos dos ficheros.



Paso 2: Tras construir nuestro "esqueleto" en HTML5, añadiremos la etiqueta link con una **relación "stylesheet"** y como **href** la ruta de nuestro fichero. Al ser un archivo paralelo, con el propio nombre del fichero o un `./` precediendo al mismo bastará.

```
<!DOCTYPE html>
<html lang="es">
```

```
<head>
<meta charset="UTF-8">
<title>Mi aplicación web</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<h1>Título de la página</h1>
<p>Este es un párrafo de texto.</p>
</body>
</html>
```

Paso 3: Vamos a probar si funciona nuestro código, vamos a insertar el siguiente fragmento en **style.css** y ver si se refleja al abrir nuestro fichero **index.html** en el navegador:

```
body{
color: white;
background-color: cadetblue;
}
```

¡Acabamos de estilar por primera vez un documento HTML!

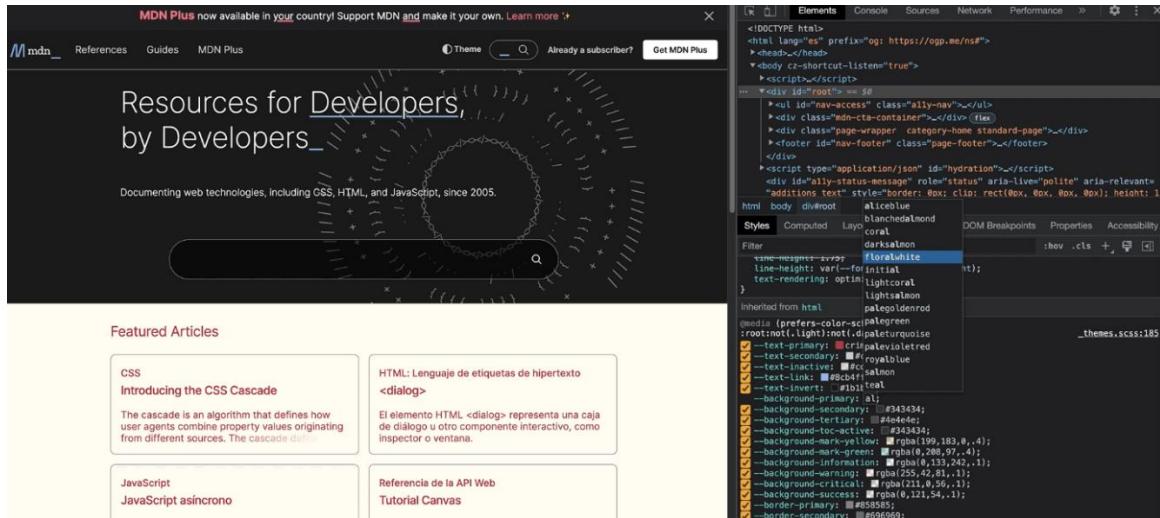


Os animamos a que intentéis cambiar ciertas cosas del documento, pero no os preocupéis, vamos a ver con detenimiento qué hacer exactamente con estos ficheros CSS y cual es la sintaxis para estilar correctamente todas nuestras aplicaciones.

Ademas de esto, al igual que inspeccionamos el código HTML en cualquier aplicación web, veremos una pestaña llamada "**Styles**" en la parte inferior de nuestro **Inspeccionador** del navegador todo el código de estilos, los cuales podremos leer, cambiar y activar/desactivar a nuestro antojo.

Recordad que podemos abrir el inspector de elementos con **Control + Mayús + J** en Windows o **⌘ + Opción + I** en macOS.

Probad a "romper" cualquier aplicación que se os ocurra e id apuntando ideas.



Estructura de CSS

Un fichero CSS es un conjunto de **reglas** que define la apariencia de una aplicación web.

Estas reglas controlan aspectos como el color, la tipografía, el tamaño de las imágenes, el layout de la página y mucho más.

Anatomía de una regla en CSS

Una **regla** de CSS es una instrucción que define cómo se debe mostrar un elemento HTML.

Es como una pequeña receta que le dice al navegador web cómo transformar un elemento en la pantalla.

Estas reglas son un conjunto de **propiedades** (claves) y **valores** asignados a un selector.

La estructura de CSS en pseudocódigo sería la siguiente:

```
selector{
  propiedad: valor;
  propiedad: valor;
  propiedad: valor;
}
```

```
selector2{
  propiedad: valor;
  propiedad: valor;
  propiedad: valor;
}
```

Selector

Es el elemento o elementos del documento que vamos a estilar, es decir, los elementos seleccionados para darles el aspecto visual deseado.

Propiedad

Es la **característica** que vamos a alterar del selector, por ejemplo, el color, el tamaño o la posición.

Valor

Será el valor concreto de la propiedad elegida, por ejemplo, la propiedad color tendrá el valor **blue**.

Cada par de propiedad y valor conformará una **regla**. Es decir, por cada propiedad y valor que asignemos a un selector le aplicaremos una **regla** de CSS.

Veamos un ejemplo estilando un párrafo con 3 reglas:

```
p{  
    color: blue;  
    font-size: 16px;  
    letter-spacing: 1.5px;  
}
```

Importante

Cada conjunto de **propiedad** y **valor** deberá ir finalizado con un **punto y coma**, y cada una de las reglas irá encerrada dentro del símbolo de las llaves: { }

Comentarios en CSS

En CSS también podemos anular código, comentar o anotar las reglas o propiedades que queramos mediante los símbolos:

```
/*.....*/  
/*Esto es un comentario en CSS*/
```

```
h1{  
    color: black;  
    font-size: 30px;  
    /*font-weight: 800*/;  
}
```

Indentación y formato de un documento CSS

- Intenta formatear el documento CSS con **Prettier** siempre que insertes una nueva regla para poder ver el código con claridad.
- Cada regla de CSS debe ocupar una línea, no es recomendable escribir las reglas en líneas por la legibilidad del documento.
- Ten en cuenta en todo momento que es una cascada, es decir, se lee de arriba abajo, así que mantén un orden correcto en todo momento para que tenga sentido consultar el documento HTML y CSS de forma más o menos paralela.

Selectores y combinadores

En CSS, los **selectores** son patrones que se utilizan para seleccionar los elementos HTML a los que se les aplicarán estilos. Los **combinadores**, por otro lado, son caracteres o palabras que se utilizan para definir la relación entre los elementos seleccionados.

Esto permite aplicar estilos de forma selectiva y precisa.

Selectores básicos

Selectores básicos: Selector de tipo o etiqueta

Este selector definirá **todos** los elementos coincidentes en base a un nombre de **etiqueta**, a los cuales se le aplicarán las reglas correspondientes.

Os dejamos el HTML:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>Document</title>
  </head>
  <body>
    <main>
      <p>Batman: The Killing Joke</p>
      <p>Batman: The Long Halloween</p>
      <p>Batman: Under the Red Hood</p>
    </main>
  </body>
</html>
```

Y el fichero CSS:

```
p {
  font-family: "Segoe UI", sans-serif;
  font-weight: 800;
  color: royalblue;
}
```

En este ejemplo le estamos aplicando una serie de reglas a **todos** los elementos `<p>` del documento HTML.

Batman: The Killing Joke

Batman: The Long Halloween

Batman: Under the Red Hood

Selectores básicos: Selector de clase

Este selector define todos los elementos que contengan el valor coincidente en su atributo **clase**.

La sintaxis de este selector se compondrá con el carácter "." más el nombre de la clase.

Os dejamos el HTML:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>Document</title>
  </head>
  <body>
    <main>
      <ul>
        <li class="highlight">Watchmen</li>
        <li class="highlight">The Sandman</li>
        <li>Bone</li>
        <li>Doom Patrol</li>
        <li>Mister Miracle</li>
        <li class="highlight">From Hell</li>
      </ul>
    </main>
  </body>
</html>
```

Y el CSS:

```
.highlight {
  font-style: italic;
  color: crimson;
}
```

En este ejemplo le estamos aplicando una serie de reglas a todos los elementos que contengan la clase "highlight".

- *Watchmen*
- *The Sandman*
- Bone
- Doom Patrol
- Mister Miracle
- *From Hell*

Selectores básicos: Selector de id

Este selector define un elemento que contenga el valor coincidente en su atributo **id**.

A nivel semántico, en nuestro documento solo puede haber un solo elemento con un determinado **id**. La sintaxis de este selector se compondrá con el carácter '#' más el nombre del id.

Os dejamos el HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Document</title>
</head>
<body>
<main>
  <ol>
    <li id="best">Chris Claremont</li>
    <li>Peter David</li>
    <li>Alan Moore</li>
    <li>Grant Morrison</li>
    <li>Geoff Johns</li>
    <li>Neil Gaiman</li>
  </ol>
</main>
</body>
</html>
```

Y el CSS:

```
#best{
  background-color: crimson;
}
```

Selectores avanzados

En selectores avanzados contemplaremos los métodos para definir a qué elementos aplicarles reglas de CSS en base a sus atributos o seleccionar todos los elementos del documento con restricciones opcionales.

Selectores avanzados: Selector universal

Este selector nos permitirá definir todos los elementos del documento. La sintaxis de este selector se compondrá con el carácter ' '. Opcionalmente, tras el carácter ' ** ' podremos indicar el nombre de una etiqueta, una clase o un atributo para seleccionar todos los elementos coincidentes en un determinado rango.

El **selector universal** es muy utilizado para resetear los estilos por defecto de todas las etiquetas, ya que muchas de ellas tienen un box-sizing, padding o margin predeterminados.

En este ejemplo le estamos aplicando una serie de reglas a todos los elementos del documento HTML.

```
*{  
    padding: 0;  
    margin: 0;  
    box-sizing: border-box;  
}
```

Selectores avanzados: Selector de atributo

Este selector nos permitirá definir todos los elementos del documento que coincidan con un atributo predeterminado.

Este selector suele ir precedido del nombre de la etiqueta cuyo atributo denominaremos a través de los caracteres “[]” y su valor tras el carácter “=”.

Os dejamos el HTML:

```
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <meta charset="UTF-8" />  
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />  
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
    <link rel="stylesheet" href="style.css" />  
    <title>Document</title>  
  </head>  
  <body>  
    <main>  
      <a href="www.google.com"> Google</a>  
      <a href="www.twitter.com">Twitter</a>  
      <a href="www.github.com">GitHub</a>  
    </main>  
  </body>  
</html>
```

Y el CSS:

```
a[href="www.google.com"] {  
    color: blue;  
}  
  
a[href="www.twitter.com"] {  
    color: aqua;
```

```
}
```

```
a[href="www.github.com"] {  
    color: purple;  
}
```

En este ejemplo le estamos aplicando una regla diferente a los diferentes anchor's especificando su atributo href y su distinto valor, pudiendo variar el color dependiendo del valor del atributo.

[Google](#) [Twitter](#) [GitHub](#)

Reto

Dado el siguiente HTML:

```
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <meta charset="UTF-8" />  
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />  
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
    <link rel="stylesheet" href="style.css" />  
    <title>Selectores</title>  
  </head>  
  <body>  
    <main>  
      <p>  Haz click aquí para ver más fotos de <a href="#">gatetes</a>. </p>  
        
  
      <section>  
        <h3>  Likes gatunos:</h3>  
        <ul>  
          <li class="like">Hierba gatuna</li>  
          <li class="like">Punteros laser</li>  
          <li class="like">Lasaña</li>  
        </ul>  
        <h3>  Top 3 dislikes gatunos:</h3>  
        <ol>  
          <li>Agua</li>  
          <li class="thunder">Tormentas</li>  
          <li>Otros gatos</li>  
        </ol>  
      </section>  
    </main>
```

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

Aplicar los siguientes estilos haciendo un uso correcto de selectores:

- Todos los elementos del documento tienen que tener un padding 0 y un margin 0.
- Todos los elementos del documento tienen que tener un font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif.

El cuerpo del documento debe de tener un color de fondo "lightcyan".

- La etiqueta main tiene que tener un padding-top y un padding-left de 1 rem y el color del texto "black".
- Tanto el ul como el ol deben tener un padding-left de 2rem.
- Los elementos con la clase like deben de tener un color "green".
- Los li de la ol deben de tener el color "crimson".
- Particularmente, el elemento con la clase thunder debe de tener el color "purple".
- El anchor tiene que tener un color "plum" y un font-style: italic.
- Los h3 deben de tener también el font-style: italic.
- La imagen debe de tener un border-radius de 1 rem.

Combinadores CSS

Los combinadores son unos tipos de selectores que nos proporcionan una relación entre selectores dependiendo de la ubicación, el tipo y los atributos de los elementos del documento.

Gracias a estos selectores podremos realizar relaciones en cadena para definir ciertos elementos de documento en un orden o descendencia/ascendencia concreta.

Veamos los diferentes combinadores que podemos utilizar:

Combinador de hermanos adyacentes

Este combinador, cuya sintaxis es el carácter "+", nos permite seleccionar hermanos adyacentes, es decir, un elemento que sigue directamente al primero y que comparte un elemento padre común.

El elemento que se seleccionará será el segundo indicado, utilizando el primero como referencia.

Os dejamos el HTML:

```
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <meta charset="UTF-8" />  
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />  
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
    <link rel="stylesheet" href="style.css" />  
    <title>Combinadores</title>  
  </head>
```

```

<body>
  <main>
    <ul>
      <li>La Comunidad del Anillo</li>
      <li>Las Dos Torres</li>
      <li>El Retorno del Rey</li>
    </ul>
  </main>
</body>
</html>

```

Y el CSS:

```

li:first-of-type + li {
  font-style: italic;
  color: orange;
}

```

En este ejemplo estamos utilizando el elemento “li” que viene continuación del primer “li”, recuperado a través de la pseudo-clase “first-of-type” (pronto la descubriremos).

- La Comunidad del Anillo
- *Las Dos Torres*
- El Retorno del Rey

Combinador general de hermanos

Este combinador, cuya sintaxis es el carácter “~”, nos permite seleccionar a todos los elementos que sigan al primero (sin necesidad de estar directamente uno detrás del otro) y que comparten el mismo elemento padre.

Es muy parecido al combinador de hermanos adyacentes pero nos permitirá recuperar todos los elementos que coincidan con la selección sin necesidad de estar juntos.

Os dejo el HTML:

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
    <title>Combinadores</title>
  </head>
  <body>
    <main>
      <span>La Comunidad del Anillo:</span>
      <p>
        La obra comienza con la noticia de la celebración del 111º cumpleaños de
      </p>
    </main>
  </body>
</html>

```

Bilbo Bolsón en la Comarca. Sin embargo, para Bilbo esta gran fiesta tenía como motivo principal su partida hacia su último viaje, producto del deseo de terminar sus días en paz y tranquilidad. El mago Gandalf, amigo de Bilbo y quien estaba informado de la decisión del hobbit, también acudió a la fiesta. Tras el discurso pronunciado por Bilbo, este se puso su anillo mágico y desapareció ante los sorprendidos hobbits. Gandalf, que sabía bien lo que acababa de hacer Bilbo, le encontró en Bolsón Cerrado y allí tuvo una pequeña discusión con él, ya que se negaba a dejar el anillo junto con el resto de la herencia a su sobrino Frodo; sin embargo, el mago acabó convenciéndole y Bilbo al fin partió. Entonces, debido a las dudas que le estaba ocasionando el anillo, Gandalf parte en busca de información sobre él, no sin antes informar a Frodo de que lo guarde y no lo toque.

</p>

<hr/>

Continuará...

</main>

</body>

</html>

Y el CSS:

```
p ~ span {  
font-style: italic;  
color: orange;  
}
```

En este ejemplo vemos como solo se han aplicado los estilos en el segundo span, ya que es el único precedido por un elemento p. En este caso se ha saltado la etiqueta "hr", ya que con este combinador no hay necesidad de que esté precedido directamente el elemento para poder recuperarlo.

La Comunidad del Anillo:

La obra comienza con la noticia de la celebración del 111º cumpleaños de Bilbo Bolsón en la Comarca. Sin embargo, para Bilbo esta gran fiesta tenía como motivo principal su partida hacia su último viaje, producto del deseo de terminar sus días en paz y tranquilidad. El mago Gandalf, amigo de Bilbo y quien estaba informado de la decisión del hobbit, también acudió a la fiesta. Tras el discurso pronunciado por Bilbo, este se puso su anillo mágico y desapareció ante los sorprendidos hobbits. Gandalf, que sabía bien lo que acababa de hacer Bilbo, le encontró en Bolsón Cerrado y allí tuvo una pequeña discusión con él, ya que se negaba a dejar el anillo junto con el resto de la herencia a su sobrino Frodo; sin embargo, el mago acabó convenciéndole y Bilbo al fin partió. Entonces, debido a las dudas que le estaba ocasionando el anillo, Gandalf parte en busca de información sobre él, no sin antes informar a Frodo de que lo guarde y no lo toque.

Continuará...

Combinador de hijo

Este combinador, cuya sintaxis es el carácter “>”, selecciona los elementos que son hijos directos del primer elemento.

Es de los más utilizados. En este caso, recuperará todos los coincidentes que comparten el mismo parente.

Os dejo el HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Combinadores</title>
</head>
<body>
  <main>
    <ul class="lotr">
      <li>La Comunidad del Anillo</li>
      <li>Las Dos Torres</li>
      <li>El Retorno del Rey</li>
    </ul>
    <ul class="hob">
      <li>Un Viaje Inesperado</li>
      <li>La Desolación de Smaug</li>
      <li>La Batalla de los Cinco Ejercitos</li>
    </ul>
  </main>
</body>
</html>
```

Y el CSS:

```
.lotr > li {
  color: orange;
  font-weight: bold;
}

.hob > li {
  color: goldenrod;
}
```

En este caso le estamos indicando con el combinador de hijo que le aplique ciertas reglas a todos los “li” descendientes de las clases .lotr y .hob.

De esta manera podemos atacar a todos los elementos de dos padres distintos.

- **La Comunidad del Anillo**
- **Las Dos Torres**
- **El Retorno del Rey**
- **Un Viaje Inesperado**
- **La Desolación de Smaug**
- **La Batalla de los Cinco Ejercitos**

Combinador de descendientes

Este combinador, cuya sintaxis es mediante el “**espacio**”, nos permite seleccionar todos los elementos que son descendientes de un elemento padre común, sin necesidad de ser descendiente directo. También es de los más utilizados.

Os dejamos el HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css" />
<title>Combinadores</title>
</head>
<body>
<main>
<section class="two-towers">
<p>
Tras internarse en Emyn Muil, <span>Frodo Bolsón</span> y
<span>Sam Gamayi</span> se encuentran con la criatura Gollum, que
intenta quitarles el Anillo por la fuerza pero, al verse vencido,
promete a los hobbits guiarlos hasta <span>Mordor</span>. Tras
sacarlos de Emyn Muil y atravesar la Ciénaga de los Muertos, llegan al
Morannon, la «Puerta Negra» de Mordor. No obstante, la gran protección
que tiene les imposibilita entrar por ahí y Gollum les propone tomar
el camino secreto de Cirith Ungol. Durante el viaje, se encuentran con
una tropa avanzada de montaraces de Gondor dirigida por Faramir, hijo
del senescal de Gondor y hermano de Boromir, quien los toma como
prisioneros y descubre que portan el <span>Anillo Único</span>.
Faramir cuenta a los hobbits que Boromir ha muerto y que fue
encontrado en un bote élfico. Esto hace pensar a Frodo, pues creía que
Boromir solo quería el Anillo, y por su hermano descubre que murió
protegiendo a la Compañía, tal y como habían jurado todos sus miembros
al salir de Rivendel. Faramir decidió llevarlos ante su padre, pero, a
su paso por la derruida ciudad de Osgiliath, los soldados gondorianos
se encuentran combatiendo a las fuerzas de
<span>Sauron</span> conducidas por algunos Nazgûl. Al darse cuenta del
maligno poder del Anillo sobre Frodo, al que casi capture uno de los
```

Nazgûl, Faramir decide liberarlos para que completen su misión.

```
</p>
</section>
</main>
</body>
</html>
```

Y el CSS:

```
.two-towers span {
color: crimson;
font-weight: bold;
}
```

En este ejemplo estamos seleccionando cualquier elemento span dentro de la clase .two-towers, sin necesidad que sea directo o adyacente, pero concretando el scope dentro de la clase indicada.

Tras internarse en Emyn Muil, **Frodo Bolsón** y **Sam Gamayi** se encuentran con la criatura Gollum, que intenta quitarles el Anillo por la fuerza pero, al verse vencido, promete a los hobbits guiarlos hasta **Mordor**. Tras sacarlos de Emyn Muil y atravesar la Ciénaga de los Muertos, llegan al Morannon, la «Puerta Negra» de Mordor. No obstante, la gran protección que tiene les imposibilita entrar por ahí y Gollum les propone tomar el camino secreto de Cirith Ungol. Durante el viaje, se encuentran con una tropa avanzada de montaraces de Gondor dirigida por Faramir, hijo del senescal de Gondor y hermano de Boromir, quien los toma como prisioneros y descubre que portan el **Anillo Único**. Faramir cuenta a los hobbits que Boromir ha muerto y que fue encontrado en un bote élfico. Esto hace pensar a Frodo, pues creía que Boromir solo quería el Anillo, y por su hermano descubre que murió protegiendo a la Compañía, tal y como habían jurado todos sus miembros al salir de Rivendel. Faramir decidió llevarlos ante su padre, pero, a su paso por la derruida ciudad de Osgiliath, los soldados gondorianos se encuentran combatiendo a las fuerzas de **Sauron** conducidas por algunos Nazgûl. Al darse cuenta del maligno poder del Anillo sobre Frodo, al que casi capture uno de los Nazgûl, Faramir decide liberarlos para que completen su misión.

Reto

Dado el siguiente HTML y haciendo uso del combinador más apropiado, cambiar a color “grey” y font-style:italic todos los elementos “p” descendientes de un elemento “section”.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css" />
<title>Selectores</title>
</head>
<body>
```

```
<main>
<p>
  Su historia se desarrolla en la Tercera Edad del Sol de la Tierra Media,
  un lugar ficticio poblado por hombres y otras razas antropomorfas como
  los hobbits, los elfos o los enanos, así como por muchas otras criaturas
  reales y fantásticas.
</p>
<p>
  La novela narra el viaje del protagonista principal, Frodo Bolsón,
  hobbit de la Comarca, para destruir el Anillo Único y la consiguiente
  guerra que provocará el enemigo para recuperarlo, ya que es la principal
  fuente de poder de su creador, el Señor oscuro Sauron.
</p>
<section>
<p>
  Tres Anillos para los Reyes Elfos bajo el cielo. Siete para los
  Señores Enanos en casas de piedra. Nueve para los Hombres Mortales
  condenados a morir. Uno para el Señor Oscuro, sobre el trono oscuro en
  la Tierra de Mordor donde se extienden las Sombras. Un Anillo para
  gobernarlos a todos. Un Anillo para encontrarlos, un Anillo para
  atraerlos a todos y atarlos en las tinieblas en la Tierra de Mordor
  donde se extienden las Sombras.
</p>
<p>J. R. R. Tolkien</p>
<span>El Señor de los Anillos</span>
</section>
</main>
</body>
</html>
```

Pseudoclases y Pseudoelementos

Los **pseudoelementos** y **pseudoclases** en CSS son herramientas que nos permiten aplicar estilos a partes específicas de un elemento HTML, ya sea en respuesta a interacciones del usuario o a la estructura del documento.

Pseudoelementos: Son elementos virtuales que se generan por el navegador y que permiten acceder a partes de un elemento que no están representadas directamente en el HTML. Se utilizan para añadir contenido o estilos a ciertas partes de un elemento sin necesidad de modificar directamente el HTML.

Pseudoclases: Son estados específicos de un elemento que pueden ser seleccionados para aplicar estilos. Estos estados pueden ser generados por la interacción del usuario, como pasar el ratón por encima de un elemento, o pueden estar relacionados con la estructura del documento, como un enlace que ha sido visitado.

Pseudoclases:active

La pseudoclase **:active** nos permite recuperar un elemento que está siendo activado por el usuario (normalmente, durante el click de un ratón hasta soltarlo). Esta pseudoclase se utiliza mucho en los elementos anchor y button.

Os dejamos un ejemplo.

El HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Pseudoclases</title>
</head>
<body>
  <main>
    <button>Haz click aquí</button>
  </main>
</body>
</html>
```

Y el CSS.

De esta manera el botón al estar pulsado convertirá las letras en naranja.

```
button:active {
  color: orange;
  font-weight: bold;
}
```



Pseudoclases: any-link

La pseudoclase **:any-link** nos permite recuperar un elemento link que haya sido visitado o no, siempre y cuando tenga un valor en su href.

Os dejamos un ejemplo.

El fichero HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Pseudoclases</title>
</head>
<body>
<main>
  <a href="www.google.com">Google</a>
  <a href="www.twitter.com">Twitter</a>
  <a>No hay link</a>
</main>
</body>
</html>
```

Y el CSS:

```
a:any-link {
  color: crimson;
  font-weight: bold;
}
```

[Google](www.google.com) [Twitter](www.twitter.com) No hay link

Pseudoclases: checked

La pseudoclase **:checked** nos permite recuperar cualquier elemento checkbox, radio u option que esté marcado o con el atributo checked activo.

El HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css" />
<title>Pseudoclases</title>
</head>
<body>
<main>
<div>
<input type="radio" name="my-input" id="yes" />
<label for="yes">Yes</label>
<input type="radio" name="my-input" id="no" checked />
<label for="no">No</label>
</div>
</main>
</body>
</html>
```

Y el CSS:

```
input:checked + label {
color: crimson;
}
```

Yes No

Pseudoclases:default

La pseudoclase **:default** nos permite recuperar cualquier elemento que esté en su estado predeterminado dentro de un formulario.

Esta pseudoclase se utiliza mucho en elementos button, checkbox, radio y option que tengan una opción marcada desde el principio.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css" />
<title>Pseudoclases</title>
</head>
<body>
<main>
<input type="radio" name="hero" id="spiderman" checked />
```

```
<label for="spiderman">Spiderman</label>
<input type="radio" name="hero" id="ironman" />
<label for="ironman">Iron Man</label>
<input type="radio" name="hero" id="daredevil" />
<label for="daredevil">Daredevil</label>
<input type="radio" name="hero" id="hulk" />
<label for="hulk">Hulk</label>
</main>
</body>
</html>
```

Y el CSS:

```
input:default + label {
  color: green;
}
```

Spiderman Iron Man Daredevil Hulk

Pseudoclases:disabled

La pseudoclase **:disabled** nos permite recuperar un elemento deshabilitado o con el atributo disabled activo.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Pseudoclases</title>
</head>
<body>
  <main>
    <button>Activado</button>
    <button disabled>Desactivado</button>
  </main>
</body>
</html>
```

Y el CSS:

```
button:disabled {
  color: red;
}
```

Activado Desactivado

Pseudoclases:enabled

La pseudoclase **:enabled** nos permite recuperar cualquier elemento habilitado en el documento, diferenciando así a los deshabilitados.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css" />
<title>Pseudoclases</title>
</head>
<body>
<main>
<button>Activado</button>
<button disabled>Desactivado</button>
</main>
</body>
</html>
```

Y el CSS:

```
button:disabled {
color: red;
}

button:enabled {
color: green;
}
```

Activado

Desactivado

Pseudoclases:first-child

La pseudoclase **:first-child** nos devuelve el primer elemento de un grupo de elementos hermanos con el mismo padre.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css" />
<title>Pseudoclases</title>
</head>
<body>
<main>
```

```
<ul>
<li>Primero</li>
<li>Segundo</li>
<li>Tercero</li>
</ul>
</main>
</body>
</html>
```

Y el CSS:

```
li:first-child {
  color: crimson;
}
```

- 
- Primero
 - Segundo
 - Tercero

Pseudoclases: focus

La pseudoclase **:focus** representa los elementos que han activado su foco al usuario interactuar con ellos o al haber entrado en uno de los elementos.

Normalmente se le aplican a los elementos input.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Pseudoclases</title>
</head>
<body>
  <main>
    <input type="text" name="username" placeholder="username" />
    <input type="text" name="email" placeholder="email@email.com" />
  </main>
</body>
</html>
```

Y el CSS:

```
input:focus {
  background-color: burlywood;
}
```



Pseudoclases: hover

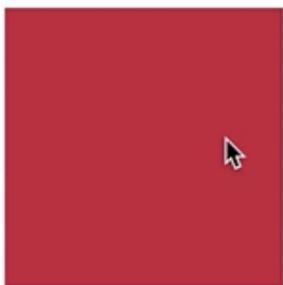
La pseudoclase **:hover** nos permite recuperar un elemento que ha sido activado por el usuario durante el desplazamiento del cursor sobre él.

No es necesario que el usuario active el elemento para que esta pseudoclase sea activada.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css" />
<title>Pseudoclases</title>
</head>
<body>
<main>
<div></div>
</main>
</body>
</html>
```

Y el CSS:

```
div {
  width: 100px;
  height: 100px;
  background-color: blueviolet;
}
div:hover {
  background-color: crimson;
}
```



Pseudoclases: invalid

La pseudoclase **:invalid** nos permitirá recuperar un elemento que haya tenido un error de campo a la hora de validarla dentro de un formulario.

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css" />
<title>Pseudoclases</title>
</head>
<body>
<main>
<form>
<label for="name">Ingresa tu nombre:</label>
<input type="text" id="name" />
<label for="email">Introduzca una dirección de e-mail:</label>
<input type="email" id="email" required />
</form>
</main>
</body>
</html>

```

Y el CSS:

```

input:invalid {
background-color: crimson;
}

```

Introduzca una dirección de e-mail:

Pseudoclases:last-child

La pseudoclase **:last-child** nos devuelve el último elemento de un grupo de elementos hermanos con el mismo padre.

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css" />
<title>Pseudoclases</title>
</head>
<body>
<main>
<ul>
<li>Primero</li>
<li>Segundo</li>
<li>Tercero</li>
</ul>

```

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

Y el CSS:

```
li:last-child {
  color: crimson;
}
```

- **Primero**
- **Segundo**
- **Tercero**

Pseudoclases:link

La pseudoclase **:link** nos devuelve los elementos que aún no han sido visitados y que incluyan el atributo href, normalmente utilizado con las etiquetas anchor, area o link.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Pseudoclases</title>
</head>
<body>
  <main>
    <ul>
      <li>
        <a href="http://www.google.com">Visitado</a>
      </li>
      <li>
        <a href="http://www.google.com">Visitado</a>
      </li>
      <li>
        <a href="http://www.twitter.com">No Visitado</a>
      </li>
    </ul>
  </main>
</body>
</html>
```

Y el CSS:

```
a:link {
  color: red;
}
```

- Visitado
- Visitado
- No Visitado

Pseudoclases:not

La pseudoclase `:not` nos devuelve los elementos que no coincidan con el selector indicado, es decir, negamos los elementos.

Se le conoce también como la pseudoclase de negación.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Pseudoclases</title>
</head>
<body>
  <main>
    <h1>Soy un h1</h1>
    <h2>Soy un h2</h2>
    <h2>Soy otro h2</h2>
  </main>
</body>
</html>
```

Y el CSS:

```
main :not(h1) {
  color: orange;
}
```

Soy un h1

Soy un h2

Soy otro h2

Pseudoclases:nth-child

La pseudoclase `:nth-child` nos devuelve los elementos que coincidan con la posición indicada entre una lista de elementos hermanos.

Esta posición se le indicará mediante el número, odd (impar) o even (par).

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
```

```

<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css"/>
<title>Pseudoclases</title>
</head>
<body>
<main>
<ul>
<li>Primero</li>
<li>Segundo</li>
<li>Tercero</li>
<li>Cuarto</li>
</ul>
</main>
</body>
</html>

```

Y el CSS:

```

li:nth-child(3) {
  color: orange;
}

```

- Primero
- Segundo
- Tercero
- Cuarto

Pseudoclases:nth-last-child

La pseudoclase **:nth-last-child** nos devuelve los elementos que coincidan con la posición indicada entre una lista de elementos hermanos contando desde el final hasta el comienzo.

Esta posición se le indicará mediante el número deseado.

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css"/>
<title>Pseudoclases</title>
</head>
<body>
<main>
<ul>
<li>Primero</li>
<li>Segundo</li>
<li>Tercero</li>
<li>Cuarto</li>
</ul>

```

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

Y el CSS:

```
li:nth-last-child(3) {
  color: orange;
}
```

- Primero
- Segundo
- Tercero
- Cuarto

Pseudoclases:only-child

La pseudo-clase **:only-child** nos devuelve los elementos que no tengan hermanos, es decir, encuentra a los hijos de los elementos que sean únicos sin formar parte de una lista de elementos paralelos.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Pseudoclases</title>
</head>
<body>
  <main>
    <ul>
      <li>Primero</li>
      <li>Segundo</li>
      <li>Tercero</li>
      <li>Cuarto</li>
    </ul>
    <ul>
      <li>Esta lista solo tiene un elemento</li>
    </ul>
  </main>
</body>
</html>
```

Y el CSS:

```
ul :only-child {
  color: orange;
}
```

- Primero
- Segundo
- Tercero
- Cuarto
- Esta lista solo tiene un elemento

Pseudoclases:read-only

La pseudo-clase **:read-only** nos devuelve los elementos que no son editables por el usuario, normalmente son aquellos inputs que incluyen el atributo “read-only” o etiquetas normales no editables.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Pseudoclases</title>
</head>
<body>
  <main>
    <input type="text" value="Este campo es editable." />
    <input type="text" value="Este campo es solo de lectura" readonly />
  </main>
</body>
</html>
```

Y el CSS:

```
input:read-only {
  color: crimson;
}

input {
  width: 100%;
}
```

Este campo es editable.

Este campo es solo de lectura

Pseudoclases:read-write

La pseudo-clase **:read-write** nos devuelve los elementos editables por el usuario, normalmente suele aplicarse a inputs o etiquetas que incluyan el atributo “contenteditable”.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
```

```

<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css"/>
<title>Pseudoclases</title>
</head>
<body>
<main>
<input type="text" value="Este campo es editable." />
<input type="text" value="Este campo es solo de lectura" readonly />
</main>
</body>
</html>

```

Y el CSS:

```

input:read-write {
  color: crimson;
}

```

```

input {
  width: 100%;
}

```

Pseudoclases:required

La pseudo-clase **:required** nos devuelve los elementos que contengan el atributo “required”, normalmente se aplica a las etiquetas “input”.

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="style.css"/>
<title>Pseudoclases</title>
</head>
<body>
<main>
<form>
<input type="text" placeholder="Username" required />
<input type="text" placeholder="Position" />
<input type="password" placeholder="Password" required />
</form>
</main>
</body>
</html>

```

Y el CSS:

```

input:required::placeholder {
  color: crimson;
}

```

```

}

input {
  width: 100%;
}



Username  

  Position  

  Password


```

Pseudoclases:root

La pseudo-clase **:root** nos devuelve el elemento raíz del documento para tener una mayor especificidad incluso que la etiqueta html.

Se utiliza mucho para declarar variables en nuestra hoja de estilos.

```

:root{
  font-size: 16px;
  font-family: "Roboto", sans-serif;
  --primary: crimson;
  --secondary: red;
  --background: grey;
  --border-radius: 1.2rem;
}

```

Pseudoelementos::after

::after crea un pseudoelemento, por defecto in-line, después del elemento a través de la propiedad “content”.

Estos pseudoelementos serán contenidos por la caja del elemento.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Pseudoelementos</title>
</head>
<body>
  <h2>Tareas completadas</h2>
  <ul>
    <li>Programar</li>
    <li>Hacer la comida</li>
    <li>Programar</li>
    <li>Limpiar la habitación</li>
    <li>Programar</li>
    <li>Programar más</li>
  </ul>

```

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

Y el CSS:

```
li::after {  
    content: "☒";  
    padding-left: 3px;  
}
```

Tareas completadas

- Programar ✓
- Hacer la comida ✓
- Programar ✓
- Limpiar la habitación ✓
- Programar ✓
- Programar más ✓

Pseudoelementos::before

::before crea un pseudoelemento, por defecto in-line, antes del elemento a través de la propiedad “content”.

Estos pseudoelementos serán contenidos por la caja del elemento.

```
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <meta charset="UTF-8" />  
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />  
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
    <link rel="stylesheet" href="style.css" />  
    <title>Pseudoelementos</title>  
  </head>  
  <body>  
    <h2>X-Men</h2>  
    <ul>  
      <li>Profesor X</li>  
      <li>Lobezno</li>  
      <li>Tormenta</li>  
      <li>Ciclope</li>  
      <li>Jean Grey</li>  
      <li>Bestia</li>  
    </ul>  
  </body>  
</html>
```

Y el CSS:

```

li::before {
  content: "X";
  padding-right: 4px;
  color: orange;
}

li {
  list-style-type: none;
}

```

X-Men

-  Profesor X
-  Lobezno
-  Tormenta
-  Ciclope
-  Jean Grey
-  Bestia

Pseudoelementos::first-letter

::first-letter aplica estilos a la primera letra de la primera linea del elemento siempre y cuando no sea precedida por una imagen, tabla o similares.

Pseudoelementos::first-line

::first-line aplica estilos a la primera linea de un bloque de texto, sabiendo que puede depender de el ancho del bloque, el documento o la propia fuente del texto en cuestión.

Pseudoelementos::placeholder

::placeholder nos permite modificar los estilos por defecto de los elementos provisionales de los elementos “input” o “textarea”.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <title>Pseudoelementos</title>
</head>
<body>
  <input type="text" placeholder="Username" />
</body>
</html>

```

Y el CSS:

```

input::placeholder {
  color: orange;
}

```

```
font-weight: bold;  
}
```

Username

Reto

Dado el siguiente HTML:

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Aventuras de los X-Men</title>  
<link rel="stylesheet" href="style.css" />  
</head>  
<body>  
<header>  
<h1>Aventuras de los X-Men</h1>  
</header>  
<nav>  
<ul>  
<li><a href="#">Inicio</a></li>  
<li><a href="#">Aventuras</a></li>  
<li><a href="#">Personajes</a></li>  
<li><a href="#">Galería</a></li>  
</ul>  
</nav>  
<main>  
<section>  
<h2>Aventuras destacadas</h2>  
<ul>  
<li><a href="#">La saga de Phoenix</a></li>  
<li><a href="#">El arco de Dark Phoenix</a></li>  
<li><a href="#">La historia de Days of Future Past</a></li>  
<li><a href="#">La batalla contra Apocalypse</a></li>  
<li><a href="#">El regreso de Cyclops</a></li>  
</ul>  
</section>  
</main>  
<footer>  
<p>  
X-Men es una serie animada de Marvel Comics. Todos los derechos  
reservados.  
</p>  
</footer>  
</body>  
</html>
```

Este es el documento CSS base:

```
* {  
    box-sizing: border-box;  
    margin: 0;  
    padding: 0;  
}  
  
body {  
    font-family: Arial, sans-serif;  
}  
  
header {  
    background-image: url(<https://lumiere-a.akamaihd.net/v1/images/pp_xmen1992_herobanner_21369_2ead90c7.jpeg?region=0,0,2048,878>);  
    background-size: cover;  
    height: 300px;  
    display: flex;  
    align-items: center;  
    justify-content: center;  
}  
  
header h1 {  
    color: white;  
    text-align: center;  
    font-size: 2em;  
}  
  
nav {  
    background-color: #333;  
}  
  
nav ul {  
    display: flex;  
    list-style: none;  
}  
  
nav li {  
    flex: 1;  
    text-align: center;  
}  
  
nav a {  
    display: block;  
    padding: 20px 0;  
    color: white;  
    text-decoration: none;  
}  
  
/*HOVER Y FOCUS AL NAV */
```

```

main {
  padding: 20px;
}

section {
  margin-bottom: 20px;
}

section h2 {
  text-align: center;
  margin-bottom: 20px;
}

section ul {
  display: flex;
  justify-content: space-around;
}

section li {
  margin: 0 10px;
  list-style-type: none;
}

/* SUSITUIR EL PUNTO DE LA LISTA POR LA X */

section a {
  color: #333;
  text-decoration: none;
  font-weight: bold;
  font-size: 1.2em;
}
/* AÑADIR CORCHETES A LOS ENLACES */
/* AÑADIR HOVER A LOS ENLACES */
/* AÑADIR FOCUS A LOS ENLACES */

footer {
  background-color: #333;
  color: white;
  padding: 20px;
  text-align: center;
}

footer p {
  margin: 0;
  font-size: 0.8em;
}

```

El resultado debería parecerse bastante a esta previsualización.



Colores

Los **colores** son elementos fundamentales utilizados para definir la apariencia visual de los elementos HTML en una aplicación web.

Los colores en CSS pueden ser especificados de varias maneras, incluyendo nombres de color predefinidos, códigos hexadecimales, valores RGB (Red, Green, Blue), valores RGBA (Red, Green, Blue, Alpha) y valores HSL (Hue, Saturation, Lightness).

Contenido y fondo

Los elementos principales a los que podemos modificar su color son los elementos que forman parte de contenido (el texto) y el fondo de la aplicación o los propios elementos.

Para modificar dichos colores aplicaremos las propiedades **color** y **background-color** respectivamente.

Pruébalo en un div con la clase "box" que contenga un párrafo con el texto "box".

```
.box {  
width: 200px;  
height: 200px;  
background-color: crimson;  
color: black;  
}
```



Métodos para declarar colores

Palabras claves

CSS proporciona palabras **clave** predefinidas para algunos colores comunes, como "red" para rojo o "blue" para azul. Ejemplo:

```
color: yellow; /* Amarillo */  
background-color: crimson; /* Carmesí */
```

Notación hexadecimal

La notación **hexadecimal** es la forma más común de definir colores en CSS.

Se utiliza una combinación de seis caracteres que representan los valores de rojo, verde y azul (en ese orden), cada uno en un rango de 00 a FF.

Por ejemplo:

```
color: #FF0000; /* Rojo */  
background-color: #00FF00; /* Verde */
```

Notación RGB

La notación **RGB** define un color en términos de sus componentes rojo, verde y azul.

Cada componente se especifica con un valor entre 0 y 255.

Por ejemplo:

```
color: rgb(255, 0, 0); /* Rojo */  
background-color: rgb(0, 255, 0); /* Verde */
```

Notación RGBA

La notación **RGBA** es similar a RGB, pero con un cuarto valor que representa la transparencia (alfa) del color.

El valor alfa va de 0 (completamente transparente) a 1 (completamente opaco).

Por ejemplo:

```
color: rgba(255, 0, 0, 0.5); /* Rojo semi-transparente */  
background-color: rgba(0, 255, 0, 0.2); /* Verde semi-transparente */
```

Notación HSL (Hue, Saturation, Lightness)

La notación **HSL** (Hue, Saturation, Lightness) proporciona una forma intuitiva de definir colores.

Representa un color en términos de matiz, saturación y luminosidad.

Por ejemplo:

```
color: hsl(0, 100%, 50%); /* Rojo de luminosidad media */  
background-color: hsl(0, 100%, 25%); /* Rojo más oscuro */
```

Notación HSLA (HSL con alfa)

La notación **HSLA** es similar a HSL, pero agrega un cuarto valor que representa la transparencia (alfa) del color, variando de 0 a 1.

Por ejemplo:

```
color: hsla(0, 100%, 50%, 0.5); /* Rojo semi-transparente */
background-color: hsla(120, 100%, 50%, 0.2); /* Verde semi-transparente */
```

Gradientes lineales

Los **gradientes lineales** permiten transiciones suaves entre dos o más colores a lo largo de una dirección específica.

En muchos casos se aplicará a la propiedad **background**.

Por ejemplo:

```
background: linear-gradient(to right, red, yellow); /* Degrado de rojo a amarillo */
```

Gradientes radiales

Los **gradientes radiales** permiten transiciones suaves entre dos o más colores desde un punto central hacia afuera en todas las direcciones.

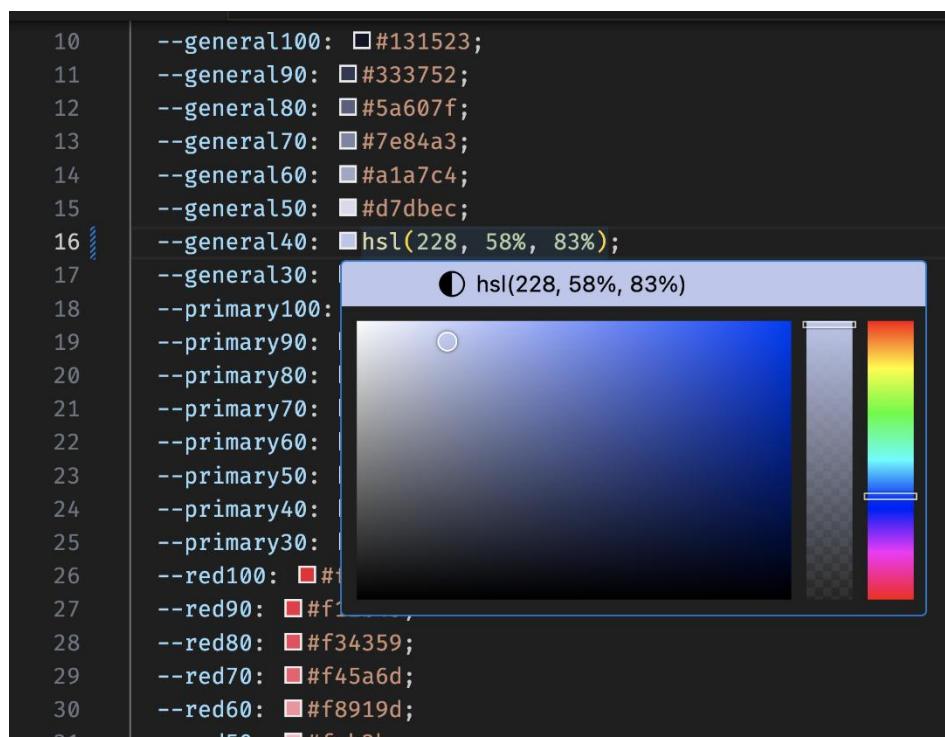
Por ejemplo:

```
background-image: radial-gradient(circle, red, yellow); /* Degrado radial de rojo a amarillo */
```

Colores en Visual Studio Code

Dentro de un fichero CSS, al pulsar en la previsualización de un color podrás modificar su valor en una interfaz similar a la que nos da un input de tipo color.

En ella se puede alterar el color, la transparencia y hasta el formato de color.



No olvides consultar el apartado de **Recursos** al final del módulo donde encontrarás aplicaciones que te ayuden a generar paletas de colores, gradientes y códigos de color únicos para tus aplicaciones.

Referencia MDN

Os dejamos la referencia de MDN por si queréis consultar en más profundidad los diferentes métodos para generar y modificar colores:

https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

Modelo Caja y Position

El modelo de caja es un sistema del navegador para interpretar estos elementos “caja”, que serán elementos HTML con unas características concretas.

La representación básica del **modelo de cajas** se basa en:

- El **borde**, que será el límite que separa el interior del exterior del elemento.
- El **márgen (margin)**, se representa en color naranja y es la parte exterior del elemento, por fuera del borde.
- El **relleno (padding)**, se representa en color verde y es la parte interior del elemento, entre el contenido y el borde.
- El **contenido(content)**, se representa en color azul y es la parte interior del elemento, excluyendo el relleno.

Tamaño y contenido

Cuando queremos controlar los elementos que definen el tamaño de nuestra caja y el comportamiento de su contenido usamos las propiedades de Contenido como: box-sizing, overflow, overflow-x y overflow-y y propiedades de Tamaño como height, width, max-height, max-width, min-height y min-width.

Algunas de las propiedades más importantes serán width y height, relativas al ancho y alto respectivamente. En el caso de indicar el valor auto a estas propiedades, el navegador se encargará de darle el tamaño, siempre dependiendo de su contenido. Es el valor por defecto, por lo que no es necesario especificarlo. *Importante: el tamaño auto por defecto dependerá del elemento HTML al que se lo indiquemos, no es lo mismo un elemento de bloque que de línea.*

```
<div id="lightblue" class="height-example">
  <span>80px de alto</span>
</div>
<div id="blueviolet" class="height-example">
  <span>40px de alto</span>
</div>
<div id="parent" class="height-example">
  <div id="child" class="height-example">
    <span>Soy la mitad de altura de mi padre</span>
  </div>
</div>

.height-example {
  width: 250px;
  margin-bottom: 5px;
  border: 3px solid #999999;
}
#lightblue {
  height: 80px;
  background-color: lightblue;
}

#blueviolet {
```

```

height: 40px;
background-color: blueviolet;
}

#parent {
height: 100px;
background-color: crimson;
}

#child {
height: 50%;
width: 75%;
background-color: goldenrod;
}

```

A través de estas dos propiedades **width** y **height** indicamos al navegador que queremos que el ancho y alto sea el valor introducido y no el “auto” por defecto.



Pero además existen algunas variaciones de estas dos propiedades:

Propiedad	Descripción
max-width	Ancho máximo que puede ocupar un elemento.
min-width	Ancho mínimo que puede ocupar un elemento.
max-height	Alto máximo que puede ocupar un elemento.
min-height	Alto mínimo que puede ocupar un elemento.

Veamos su uso (Así, indicaremos unos máximos y mínimos donde el ancho y el alto podrá variar pero no superar por arriba o por abajo los valores indicados).

```

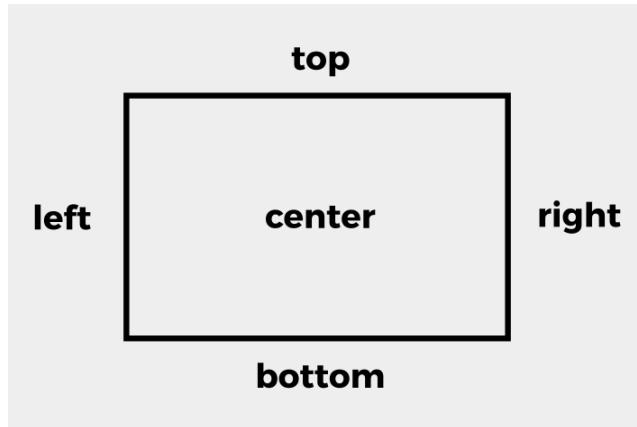
div {
width: 300px;

```

```
height: 100px;  
background: whitesmoke;  
max-width: 500px;  
}
```

Otra de las utilidades que os vendrá bien saber es la relativa a las zonas de un elemento, que son propiedades que hacen referencia a una zona concreta del elemento, estas serán **top**, **bottom**, **right**, **left** y **center**.

Su propio nombre indica la zona a la que hacen referencia, por lo que serán bastante sencillas de entender.



También existen propiedades que hacen referencia al propio contenido que insertamos. Estas serán muy útiles cuando tengamos cajas con un ancho definido pero el contenido sobrepase este ancho... lo que hará que el contenido sea más grande que la caja y tenga que salir por algún lado.

La propiedad más importante aquí será **overflow**, que además tendrá variantes x o y para indicar el comportamiento de cada eje por separado.

Veamos un ejemplo:

Overflow

Los valores que puede tomar esta propiedad serán:

- **visible**: se mostrará el contenido que sobresalga, por lo que se desbordará.
- **hidden**: todo el contenido que no entre en la caja se ocultará.
- **scroll**: se introducirán barras de scroll verticales y horizontales para que todo el contenido esté visible. No hay desbordamiento y todo el contenido se podrá ver.
- **auto**: se colocarán barras de scroll cuando sea necesario.

```
<p class="overflow-visible">
```

Valor visible. Lorem ipsum dolor sit amet, consectetur adipisicing elit.

*Tempora, expedita? Tempora aut tempore est nemo sint porro sequi optio vel
ex amet. Nihil impedit ipsa cupiditate unde asperiores, dignis*

```
</p>
```

```
<br />
```

```
<p class="overflow-hidden">
```

overflow-hidden. Lorem ipsum dolor sit, amet consectetur adipisicing elit.

Doloremque architecto quis neque, ea vel distinctio at numquam, sit, fuga

```

asperiores aut? Reprehenderit ea hic est ducimus deleniti iure ex i
</p>
<br />
<p class="overflow-scroll">
Se muestra un scroll cuando el contenido se desborde. Lorem ipsum dolor
sit amet consectetur adipisicing elit. Vero cupiditate ea doloribus
perspiciatis, preferendis enim id sit accusamus dicta dolor maxime
eligendi odit harum opti
</p>

<br />
<p class="overflow-auto">
Overflow auto. Lorem ipsum dolor sit amet consectetur adipisicing elit.
Neque accusantium sunt laudantium ullam cupiditate esse, quas aperiam
saepe cumque reprehenderit consequatur culpa libero placeat provident eum
quibusdam, dicta veritatis ipsam. Lorem ipsum dolor sit amet, consectetur
adipisicing elit
</p>

.overflow-visible {
height: 2em;
width: 12em;
border: 3px dotted orange;
margin-bottom: 100px;
overflow: visible; /* dibuja barras si es necesario */
}

.overflow-hidden {
width: 12em;
height: 2em;
border: 3px dotted orange;
overflow: hidden; /* No Scroll */
}

.overflow-scroll {
width: 12em;
height: 2em;
border: 3px dotted orange;
overflow: scroll; /* Scroll */
}

.overflow-auto {
width: 12em;
height: 2em;
border: 3px dotted orange;
overflow: auto; /* auto */
}

```

Como podemos ver en el resultado, el comportamiento dependerá del valor de esta propiedad overflow.

Valor visible. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Tempora, expedita? Tempora aut tempore est nemo sint porro sequi optio vel ex amet. Nihil impedit ipsa cupiditate unde asperiores, dignis

overflow-hidden. Lorem ipsum dolor sit, amet.

Se muestra un scroll

Overflow auto. Lorem ipsum dolor sit amet

Con todo lo que hemos ido contando se puede entender de forma sencilla el modelo de caja, pero tenemos que entender que cuando trabajemos con una aplicación muy grande puede ocurrir que tengamos numerosas cajas y se pueden dar comportamientos que a priori no esperábamos, ¡por eso hay que estar preparados!

Por defecto, cuando añadimos un borde, padding, etc, estas dimensiones se añadirán a las del contenido. Es decir, por defecto cuando asignamos un width y height a un elemento se lo asignamos en realidad al contenido, y todo lo “extra” le irá añadiendo tamaño.

Esto puede resultar frustrante cuando habíamos asignado, por ejemplo, 200px de width y al meter padding vemos que el elemento acaba ocupando mucho más.

Para evitar estos cálculos mentales existe la propiedad **box-sizing**, que por defecto tendrá un valor de **content-box**, que significa lo que comentábamos, las propiedades width y height se referirán únicamente al contenido.

El otro valor que puede tener esta propiedad es **border-box**, que sí que incluirá en el width y height que asignemos espacio para propiedades como border, padding, etc.

Vamos a poner un ejemplo de código:

Box-sizing

```
<div class="content-box">Contenido de la caja</div>
<br>
<div class="border-box">Borde de la caja</div>

div {
  width: 250px;
```

```

height: 75px;
padding: 25px;
border: 5px solid black;
background: lightblue;
}

.content-box {
  box-sizing: content-box;
  /* Ancho total: 250px + (2 * 25px) + (2 * 5px) = 310px
   Altura total: 75px + (2 * 25px) + (2 * 5px) = 135px
   Ancho de la caja de contenido: 250px
   Altura de la caja de contenido: 75px */
}

.border-box {
  box-sizing: border-box;
  /* Ancho total: 250px
   Altura total: 75px
   Ancho de la caja de contenido: 250px - (2 * 25px) - (2 * 5px) = 190px
   Altura de la caja de contenido: 75px - (2 * 25px) - (2 * 5px) = 10px */
}

```

Lo que estamos haciendo aquí es, en el <div> con el box-sizing en **content-box**, indicar que el width del **CONTENIDO** será 250 y a esto se le tendrá que añadir el padding y el border.

Por otro lado, en el <div> con el box-sizing en **border-box** estamos indicando que el ancho total de la **CAJA** será 250px, por lo que el espacio que tendrá el contenido serán esos 250px menos lo asignado al padding y border.

Margin y padding

En el modelo de cajas, los **márgenes** (*margin*) son los espacios exteriores de un elemento. Es decir, el espacio que hay entre el borde de un elemento y su exterior. El **relleno** (*padding*) será el espacio entre el borde del elemento y el contenido.

Para empezar debemos mencionar, recalcar y recordar siempre que los margin NO son lo mismo que los paddings.

Los márgenes o **margin** implican espacio entre los bordes del elemento HTML con el que estemos trabajando y los bordes de otros elementos, por lo que hacen referencia a espaciados exteriores, mientras que el **padding** o relleno será espacio entre los bordes del elemento y el contenido del elemento, lo que supone espacio interior.

En relación a los márgenes, se podrán alterar de forma general o de forma específica en cada lado del elemento (arriba, derecha, abajo e izquierda).

Estas propiedades serán:

Propiedad	Descripción
margin-top	Establece un tamaño de margen superior.

margin-left	Establece un tamaño de margen a la izquierda.
margin-right	Establece un tamaño de margen a la derecha.
margin-bottom	Establece un tamaño de margen inferior.

Si queremos **centrar horizontalmente un elemento en pantalla** fácilmente existen tips, como por ejemplo aplicar un ancho fijo al contenedor (por ejemplo `width: 200px`) y luego aplicar un `margin: auto`.

Así, el navegador sabrá que ese elemento tiene un tamaño concreto repartirá los márgenes de manera igual en izquierda y derecha, y centraremos el elemento.

```
<div class="margin-example">Este es un ejemplo de centrar un div</div>
```

```
.margin-example{
    width: 800px;
    margin: auto;
    height: 100;
    background-color: cornflowerblue;
    color: whitesmoke;
}
```

En relación al relleno o padding, tendrá las mismas propiedades que el margin:

Propiedad	Descripción
padding-top	Aplica un relleno interior en el espacio superior de un elemento.
padding-left	Aplica un relleno interior en el espacio izquierdo de un elemento.
padding-right	Aplica un relleno interior en el espacio derecho de un elemento.
padding-bottom	Aplica un relleno interior en el espacio inferior de un elemento.

Por defecto el relleno es 0, deberá modificarse para ver el padding aplicado.

```
<div class="margin-example">Este es un ejemplo de centrar un div con padding</div>
```

```
.margin-example{
    width: 800px;
    margin: auto;
    height: 100;
    padding: 20px;
    background-color: cornflowerblue;
    color: whitesmoke;
}
```

Como hemos comentado, existen 4 lados (top, right, bottom y left) en los que aplicar margin/padding, pero no existen atajos o maneras sencillas para no tener que definir los 4 lados.

Veámoslo en una tabla:

Propiedad	Descripción
margin o padding	1 parámetro. Aplica el mismo margen a todos los lados.
margin o padding	2 parámetros. Aplica margen top/bottom y left/right.
margin o padding	3 parámetros. Aplica margen top, left/right y bottom.
margin o padding	4 parámetros. Aplica margen top, right, bottom e left.

Separados por // encontraremos los diferentes valores posibles de margin/padding dependiendo de lo que queramos conseguir.

```
<div class="margin-example">  
Este es un ejemplo de centrar un div con todas las posibilidades de padding  
</div>
```

```
.margin-example {  
width: 200px;  
margin: auto;  
padding: 10px // 10px 5px // 10px 5px 8px // 10px 5px 8px 2px;;  
background-color: cornflowerblue;  
color: whitesmoke;  
}
```

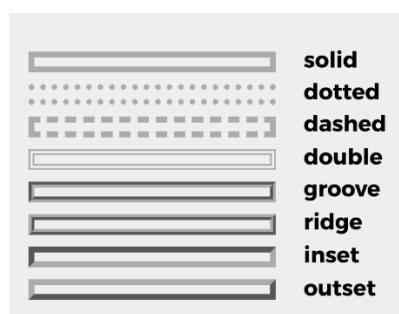
Propiedades de bordes

En CSS es posible especificar el aspecto que tendrán los **bordes** de cualquier elemento HTML, pudiendo incluso, dar diferentes características a zonas particulares del borde.

Los bordes son una parte esencial del modelo de caja. La mayoría de webs que usáis en vuestro día a día contienen estos bordes que “encierran” el contenido y le dan un toque a nuestras cajas. Tendremos estas propiedades a la hora de modificar los bordes:

- **border-color**: especifica el color que se usará en el borde.
- **border-width**: especificará un tamaño para el grosor del borde.
- **border-style**: define el estilo a usar en dicho borde.

Dentro de esta propiedad tendremos distintas variantes:



```
<div>Este es un ejemplo de aplicación de bordes</div>
```

```
div {  
    border-color: gray;  
    border-width: 1px;  
    border-style: dotted;  
}
```

Este es un ejemplo de aplicación de bordes

Una vez visto esto, podemos avanzar a los atajos, que nos vienen a solucionar la vida.

Principalmente se crearon para hacer este tipo de inserciones más sencillas y que no tengamos que escribir muchas líneas de código para diseñar nuestros bordes.

Existe así la propiedad border a la que podremos aplicarle en una sola línea todos los valores.

El ejemplo anterior se podía condensar en:

```
div {  
    border: 1px solid gray;  
}
```

Por último, debemos mencionar que se pueden aplicar bordes a los distintos lados de nuestra caja, con las propiedades border-top, border-right, border-bottom y border-left respectivamente.

Y, si lo que queremos es algo más avanzado... podemos incluso redondear las esquinas para dar una sensación mucho más vistosa. Este tipo de redondeces se hacen sobre todo a la hora de crear botones, y para ello podemos usar la propiedad border-radius, que tendrá valores numéricos o en porcentaje.

```
<div>Este es un ejemplo de aplicación de bordes redondeados</div>
```

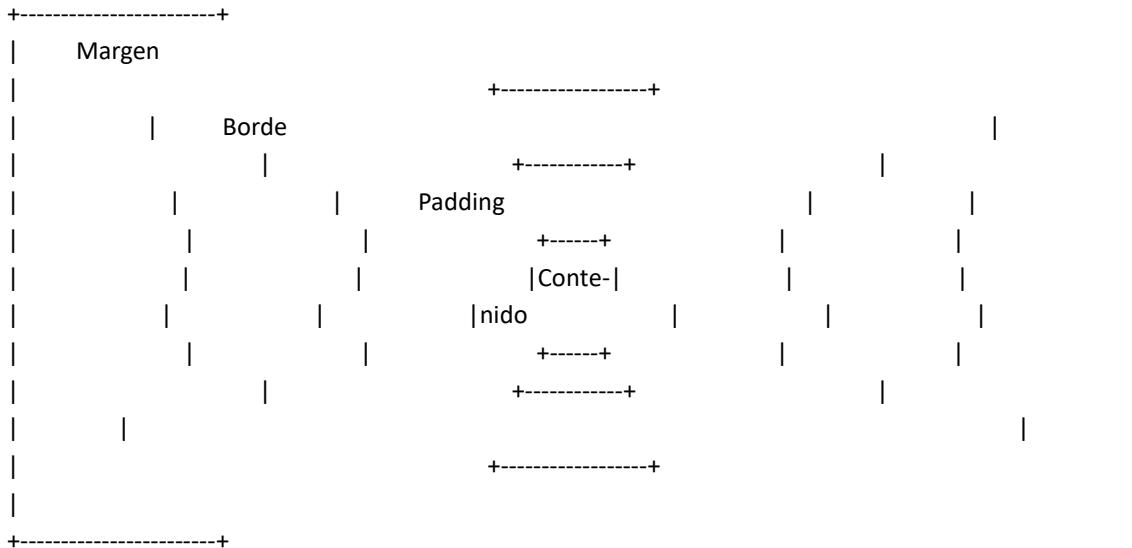
```
div {  
    border: 2px solid blue;  
    border-radius: 8px;  
}
```

Este es un ejemplo de aplicación de bordes redondeados

Resumen

El modelo de cajas (o "box model" en inglés) es un concepto fundamental en CSS que describe cómo se relacionan entre sí el contenido, el padding (relleno), el borde y el margen de un elemento.

Imagina que cada elemento en tu página web es una caja. Esta caja tiene varias capas:



Reto

Crea una tarjeta de perfil que contenga:

- Una imagen de perfil (puedes usar cualquier imagen de placeholder).
- Un nombre.
- Una breve descripción.

Requisitos:

- La tarjeta debe tener un margen de 50px para separarla de los bordes de la ventana del navegador.
- La tarjeta debe tener un borde de 5px de grosor y de color a tu elección.
- La imagen de perfil debe estar centrada en la tarjeta y tener un padding de 20px.
- El nombre debe tener un padding superior e inferior de 10px y un padding izquierdo y derecho de 20px.
- La descripción debe tener un padding de 20px.

Position



En líneas generales, cuando hablamos de elementos en línea nos referimos a elementos que en nuestro navegador aparecerán apilados unos al lado de otros de izquierda a derecha.

Por otro lado, los elementos en bloque ocuparán todo el ancho disponible por lo que aparecerán apilados uno encima de otro.

La mezcla de estos elementos compondrán nuestra página web.

A la hora de cambiar el funcionamiento por defecto de la posición de los elementos, podremos darle a la propiedad position los siguientes valores:

- **static**: es el posicionamiento por defecto y no será necesario especificarlo en nuestros elementos.
- **relative**: los elementos se colocan igual que con el valor static pero la diferencia será que se nos activarán las propiedades top, bottom, left y right para posicionar los elementos a nuestro antojo.
- **absolute**. Con este valor los elementos se posicionarán tomando como referencia el primer contenedor que tenga posicionamiento diferente a static. Este posicionamiento es el más complejo de usar y entender. Por ejemplo, si el contenedor padre tiene posicionamiento estático, pasamos a mirar el posicionamiento del padre del contenedor padre, y así sucesivamente hasta encontrar un contenedor con posicionamiento no estático o llegar a la etiqueta <body>, en el caso que se comportaría como el ejemplo anterior.
- **fixed**: es parecido al absolute, salvo por el detalle de que el posicionamiento fixed el elemento se mostrará en una posición fija dependiendo de la región visual del navegador. Por lo tanto, con ese tipo de posicionamiento, aunque hagamos scroll el elemento seguirá en la misma posición.
- **sticky**. Se denomina también “pegado”. Se asemeja al relativo, y se suele usar para fijar los <nav> de las páginas de forma que aunque hagamos scroll no perderemos el <nav> de vista.

Una vez desactivamos el comportamiento por defecto (es decir, establecemos la propiedad position en algún valor distinto de static), se nos activarán las propiedades **top**, **bottom**, **left** y **right** que hará que los elementos se desplacen en la dirección que marquemos. Estas propiedades serán fundamentales.

Os dejamos un ejemplo de código para que probéis y practiquéis con esta propiedad:

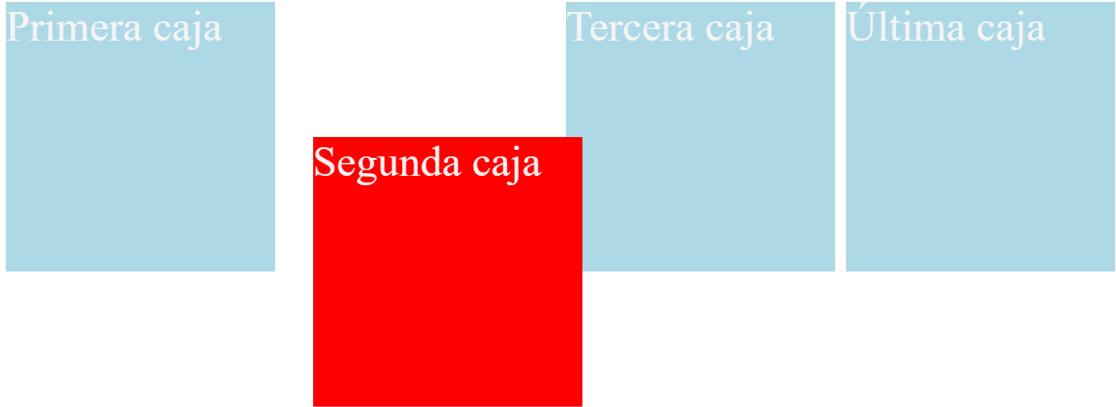
```
<div class="box" id="one">Primera caja</div>
<div class="box" id="two">Segunda caja</div>
<div class="box" id="three">Tercera caja</div>
<div class="box" id="four">Última caja</div>

* {
  box-sizing: border-box;
}

.box {
  display: inline-block;
  width: 100px;
  height: 100px;
  background: lightblue;
  color: whitesmoke;
}

#two {
```

```
position: relative;  
top: 50px;  
left: 10px;  
background: red;  
}
```



Un concepto interesante también es el de la profundidad o niveles en un eje z, que define la profundidad de un elemento.

Esta profundidad de la que hablamos se cambia con la propiedad **z-index**, y su funcionamiento es que habrá que indicar el valor que queremos darle a la profundidad del elemento.

Tened en cuenta que el elemento que tenga el z-index más alto será el que mayor profundidad tenga.

Esto nos será útil a la hora de traer al fondo-frente algunos elementos y apilarlos. Esta propiedad requerirá un valor de la propiedad position diferente de static.

Os dejamos un ejemplo de código y una imagen para que os aclare el funcionamiento.

```
<div class="rojo">  
Rojo  
<div class="verde">Verde</div>  
</div>  
<div class="gris">Gris</div>  
<div class="azul">Azul</div>  
  
.amarillo {  
z-index: 0;  
}  
.rojo {  
z-index: 1;  
width: 100px;  
height: 20px;  
background-color: red;  
}  
.gris {  
z-index: 2;  
}  
.verde {  
z-index: 3;
```

```
position: relative;  
bottom: 15px;  
left: 40px;  
width: 100px;  
height: 20px;  
background-color: green;  
}  
.azul {  
z-index: 100;  
}
```



Si queremos más control aún a la hora de posicionar nuestros elementos tenemos una propiedad float disponible. Esta propiedad como su nombre indica hará que el elemento “flote” a izquierda o derecha dependiendo del valor que le demos.

Aún así, siempre será mejor usar la propiedad display en lugar de float, ya que el resultado será más limpio con display y float es más complejo de usar.

Si por lo que sea lo que queremos es impedir elementos flotantes tenemos la propiedad clear.

Resumen

Vamos a explicar las propiedades de position en CSS con ejemplos:

1. position: static

Es el valor por defecto. Los elementos se posicionan en el flujo normal del documento, uno después del otro, según aparecen en el código.

Es el valor por defecto. Los elementos se posicionan en el flujo normal del documento, uno después del otro, según aparecen en el código.

2. position: relative

El elemento se posiciona en relación a su posición original. Puedes usar top, right, bottom y left para mover el elemento desde donde normalmente estaría.

```
<div style="position: relative; left: 10px; top: 5px;">  
Este elemento se ha movido 10px a la derecha y 5px hacia abajo de su posición  
original.  
</div>
```

3. position: absolute

El elemento se posiciona en relación al ancestro posicionado más cercano (que no sea static). Si no hay ancestros con posición, se posiciona en relación al <html>.

```
<div  
style="position: relative; height: 100px; width: 200px; border: 1px solid black;">
```

```
>  
Contenedor con position: relative  
<div style="position: absolute; top: 0; right: 0;">  
Este elemento está en la esquina superior derecha del contenedor.  
</div>  
</div>
```

4. position: fixed

El elemento se posiciona en relación a la ventana del navegador, lo que significa que permanece en la misma posición incluso si se desplaza la página.

```
<div style="position: fixed; top: 10px; right: 10px;">  
Este elemento siempre estará en la esquina superior derecha, sin importar cómo se desplace la página.  
</div>
```

5. position: sticky

Es una mezcla entre relative y fixed. El elemento se desplaza con el flujo normal del contenido hasta que alcanza un punto de anclaje definido, como top: 0, y luego se comporta como fixed.

```
<div style="position: sticky; top: 0;">  
Esta barra se "pegará" en la parte superior de la ventana al desplazarse.  
</div>  
<div style="height: 2000px;">  
Contenido largo para permitir el desplazamiento.  
</div>
```

Reto

Crea un párrafo de texto y, cuando pases el cursor sobre una palabra específica, muestra un tooltip (pequeña caja de texto) con una definición o explicación adicional.

Requisitos:

- La palabra que activará el tooltip debe estar envuelta en un elemento, por ejemplo, un ``.
- El tooltip debe tener `position: absolute` y estar oculto por defecto.
- Al pasar el cursor sobre la palabra, el tooltip debe aparecer justo encima de ella.
- El elemento contenedor del párrafo debe tener `position: relative` para que el tooltip se posicione en relación a él.

Variables

Las **variables** en CSS son como contenedores que te permiten almacenar y reutilizar valores en diferentes partes de tu hoja de estilo.

Esto no solo mejora la organización de tu código, sino que también facilita enormemente la actualización y el mantenimiento del diseño de tu aplicación web.

Básicamente es como si memorizáramos ciertas palabras claves con valores guardados en ellas y que al modificar dichas variables se modifique cualquier propiedad donde se usen a modo de "centralita", ahorrando mucho tiempo al modificar decenas de propiedades una a una.

Imagina que el color de una marca cambia y tenéis que modificar una hoja de estilos entera. Si tenéis un color que utilizáis en 30 reglas deberíais ir una por una modificándolo.

Sin embargo, si en estas reglas hemos utilizado una variable, con cambiar el valor de dicha variable se modificará todo en un solo movimiento.

Declaración de variables

Para definir una **variable** en CSS, se utiliza la pseudo-clase **:root** para establecer variables globales que pueden ser accesibles desde cualquier parte del documento.

Aquí tienes un ejemplo de cómo se define un conjunto de variables:

```
:root {  
  --primary-color: #007bff; /* Define una variable para el color primario */  
  --text-size: 16px; /* Define una variable para el tamaño de texto */  
  --main-font: Arial, sans-serif; /* Define una variable para la fuente principal */  
}
```

Uso de variables

Una vez que has definido tus variables, puedes usarlas en cualquier parte de tu hoja de estilo utilizando la función **var()** junto con el nombre de la variable.

Aquí hay un ejemplo de cómo usar las variables definidas anteriormente:

```
.elemento {  
  color: var(--primary-color); /* Usa la variable de color primario */  
  font-size: var(--text-size); /* Usa la variable de tamaño de texto */  
  font-family: var(--main-font); /* Usa la variable de fuente principal */  
}
```

Ahora, vamos a ampliar nuestros ejemplos y definir un conjunto más amplio de variables para diferentes propósitos, como colores adicionales, tamaños de texto específicos y fuentes secundarias:

```
:root {  
  --color-primary: #007bff; /* Azul */  
  --color-secondary: #6c757d; /* Gris */  
  --color-accent: #ff4500; /* Naranja */  
  --text-size-small: 14px; /* Pequeño */  
  --text-size-medium: 16px; /* Mediano */  
  --text-size-large: 20px; /* Grande */
```

```
--primary-font: Arial, sans-serif; /* Fuente principal */
--secondary-font: "Roboto", sans-serif; /* Fuente secundaria */
--gap: 20px; /* Hueco reusable para paddings, margenes o gap */
}

.title {
  color: var(--color-primary);
  font-size: var(--text-size-large);
  font-family: var(--primary-font);
}

.paragraph {
  color: var(--color-secondary);
  font-size: var(--text-size-medium);
  font-family: var(--secondary-font);
  padding: var(--gap);
}
```

Fondos e imágenes

El dominio de la manipulación de fondos e imágenes en CSS es crucial para crear experiencias visuales envolventes y estéticamente agradables.

Veamos detalladamente cómo aplicar fondos de pantalla, las propiedades asociadas y cómo tratar las imágenes en CSS, incluyendo su colocación, tamaño, relación de aspecto, y más.



Fondos de pantalla

- Propiedad **background-image**: Esta propiedad permite incrustar una imagen como fondo de un elemento HTML mediante su url (local o dirección web).

```
.container {  
    background-image: url('imagen.jpg');  
}
```

Propiedad **background-repeat**: Controla si y cómo se repite la imagen de fondo.

```
.container {  
    background-repeat: no-repeat; /* Para evitar la repetición */  
}
```

Propiedad **background-size**: Permite controlar el tamaño de la imagen de fondo.

```
.container {  
    background-size: cover; /* Para cubrir todo el contenedor */  
}
```

Tratamiento de imágenes

- Propiedad **object-fit**: Controla cómo se adapta la imagen dentro de su contenedor manteniendo su relación de aspecto. Con el valor cover nunca se nos deformarán las imágenes.

```
img {  
    width: 100%;  
    height: 100%;  
    object-fit: cover; /* Ajusta la imagen sin distorsión */  
}
```

Propiedad **object-position**: Controla la posición de la imagen dentro de su contenedor.

Si hemos aplicado **object-fit: cover** podremos controlar qué trozo de la imagen perdemos o no al "recortarla".

```
img {  
    object-position: center; /* Centra la imagen */  
}
```

Propiedad max-width: Limita el tamaño máximo de una imagen para evitar que se distorsione en dispositivos más pequeños.

```
img {  
    max-width: 100%; /* Se ajustará al tamaño del contenedor */  
}
```

Propiedad aspect-ratio: Establece la relación de aspecto de un elemento, garantizando que se mantenga incluso si la dimensión del contenedor cambia. Es de los métodos más modernos para respetar la proporción de nuestras imágenes.

```
img {  
    aspect-ratio: 16/9; /* Establece una relación de aspecto de 16:9 */  
}
```

Prueba con el siguiente ejemplo

```
<section class="ejemplo">  
    <div class="width ratio2-1">  
        <div class="contenido">Aspect ratio 2:1</div>  
    </div>  
    <div class="width ratio16-9">  
        <div class="contenido">Aspect ratio 16:9</div>  
    </div>  
    <div class="width ratio4-3">  
        <div class="contenido">Aspect ratio 4:3</div>  
    </div>  
    <div class="width ratio1-1">  
        <div class="contenido">Aspect ratio 1:1</div>  
    </div>  
</section>  
  
@import url(<https://fonts.googleapis.com/css?family=Dosis:300>);  
* {  
    margin: 0;  
    padding: 0;  
    border: 0 none;  
    position: relative;  
}  
  
html, body {  
    background: #164C88;  
    font-family: Dosis;  
    font-size: 1.1rem;  
    line-height: 1.6;  
    color: #F6FAFD;  
}  
body {padding: 1rem;}  
.ejemplo {
```

```
text-align: center;
}
.width {
    width: 45%;
    background: #F9D237;
    position: relative;
    display: inline-block;
    margin: 1%;
    vertical-align: top;
    box-shadow: 0 0 4px rgba(0,0,0,.3);
}
.width:before {
    content: '';
    display: block;
}
.ratio1-1:before {padding-top: 100%;}
.ratio2-1:before {padding-top: 50%;}
.ratio4-3:before {padding-top: 75%;}
.ratio16-9:before {padding-top: 56.25%;}
.contenido {
    position: absolute;
    top: 0; left: 0; bottom: 0; right: 0;
    font-size: 1.5rem;
    color: #444;
}
a {color: #F9D237}
```

Fuentes tipográficas

Las **fuentes tipográficas** desempeñan un papel crucial en el diseño web, ya que influyen en la legibilidad, la estética y la experiencia del usuario.

En CSS, tienes varias opciones para controlar y personalizar las fuentes.

Veamos cómo trabajar con ellas.

Familias de Fuentes

La propiedad **font-family** en CSS te permite especificar qué fuentes se utilizarán para mostrar el texto.

Puedes definir una lista de fuentes preferidas, separadas por comas.

Si una fuente no está disponible, se utilizará la siguiente en la lista. Esto servirá para tener fuentes similares siempre por si alguna falla.

```
font-family: Arial, Helvetica, sans-serif;
```

Fuentes Genéricas

CSS proporciona nombres genéricos de fuentes que se aplican según el tipo de contenido.

Los nombres genéricos comunes son **serif**, **sans-serif**, **monospace**, **cursive** y **fantasy**.

Estos nombres representan categorías de fuentes, permitiendo que el navegador elija la más adecuada dentro de esa categoría según su configuración y preferencias del usuario.

```
font-family: serif; /* Fuentes con remates */  
font-family: sans-serif; /* Fuentes sin remates */  
font-family: monospace; /* Fuentes de ancho fijo */
```



Fuentes Personalizadas

Además de las fuentes genéricas, puedes utilizar fuentes personalizadas proporcionadas por el usuario mediante **@font-face**.

Esto te permite incorporar fuentes personalizadas desde tu servidor, una vez las hayamos descargado y guardado en la carpeta de nuestro proyecto (por ejemplo, en una carpeta dentro de assets llamada **fonts**).

Primero, define la fuente usando **@font-face**, dale un nombre y luego úsala como cualquier otra fuente.

```
@font-face {  
    font-family: MiFuentePersonalizada;  
    src: url('ruta/a/miFuente.woff2') format('woff2'),  
         url('ruta/a/miFuente.woff') format('woff');  
}  
  
body {  
    font-family: MiFuentePersonalizada, Arial, sans-serif;  
}
```

Estilo de la Fuente

La propiedad **font-style** define si el texto es normal, cursiva o inclinado.

Utiliza **normal** para el texto normal, **italic** para cursiva y **oblique** para inclinado.

```
font-style: italic; /* Cursiva */
```

Peso de la Fuente

La propiedad **font-weight** define el grosor de la fuente.

Puedes especificar valores numéricos (**100** a **900**) o palabras clave (**normal**, **bold**, etc.).

normal es equivalente a 400 y **bold** a 700.

```
font-weight: 900; /* Negrita */
```

Tamaño de la Fuente

La propiedad **font-size** controla el tamaño de la fuente.

Puedes especificarlo en píxeles, em, rem, porcentajes, etc.

Además, puedes utilizar valores relativos para ajustar el tamaño de la fuente con respecto al texto circundante.

```
font-size: 16px; /* Tamaño en píxeles */  
font-size: 1.2em; /* Tamaño en relación al tamaño del texto padre */
```

Espaciado de Texto

Las propiedades **letter-spacing** y **word-spacing** controlan el espacio entre letras y palabras respectivamente.

Puedes especificar valores positivos o negativos para ajustar el espaciado según sea necesario.

```
letter-spacing: 1px; /* Espacio entre letras */  
word-spacing: 2px; /* Espacio entre palabras */
```

Google Fonts

Google fonts es un proveedor gratuito de fuentes, además es cómodo y rápido, por lo que es una opción muy recomendada a la hora de buscar fuentes para vuestra página web.

Para ello entraremos en la fuente que deseamos y clicaremos en Get font y Get Embed Code.

Os dejamos la fuente utilizada:

[Google Fonts: Roboto](#)

Veamos de que dos formas podemos incluir una fuente "on-line" sin necesidad de descargarla en el servidor.

Inclusión por CSS

En caso de no querer descargar la fuente podemos usar **@import** en CSS y usarlo.

Todas estas importaciones tienen que ir al principio de nuestro fichero CSS. Recuerda que este código está en la página anteriormente comenzada, solo tienes que pegarlo.

```
@import url("<https://fonts.googleapis.com/css2?family=Roboto:wght@100&display=swap>");
```

Y para usarlo indicamos que nuestro documento usará dicha fuente con el selector general o en el elemento deseado para indicar qué contenido queremos que se muestre con esa tipografía.

Por ejemplo, en este caso todo el documento se mostrará con esta tipografía:

```
@import url("<https://fonts.googleapis.com/css2?family=Roboto:wght@100&display=swap>");
```

```
* {  
    font-family: "Roboto", sans-serif;  
}
```

Inclusión por HTML

Es la menos recomendada por ello la dejamos para el final, simplemente tienes que añadir dentro del head una serie de links que nos proporciona la página anterior.

```
<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:wght@700&display=swap>"  
rel="stylesheet">
```

Por último os dejamos otras webs alternativas donde podéis encontrar más fuentes:

- [Adobe Fonts](#) ofrece miles de fuentes con cualquier suscripción a Creative Cloud. Con otros servicios (que no se basan en fuentes de código abierto como Google Fonts, al menos), a menudo hay que preocuparse por las licencias.

- [Fonts.com](#) tiene una amplia selección de fuentes tanto para el escritorio como para la web. Te proporcionan el código que necesitas para ponerlas en tu sitio. El problema es que hay varias licencias diferentes, y puede ser un poco confuso.
- [TypeNetwork](#) ofrece fuentes de alta calidad para proyectos serios con una variedad de opciones de licencia.
- Antes de que existiera Google Fonts, [Font Squirrel](#) era el lugar donde encontrar fuentes gratuitas con licencia comercial para usar en cualquier proyecto. Tiene una gran selección, pero lamentablemente no hay opción de alojamiento de fuentes. Tendrás que descargar las fuentes y subirlas a tu sitio manualmente. Tampoco todas las fuentes están optimizadas para la web, pero puedes probar el [generador de fuentes web](#).

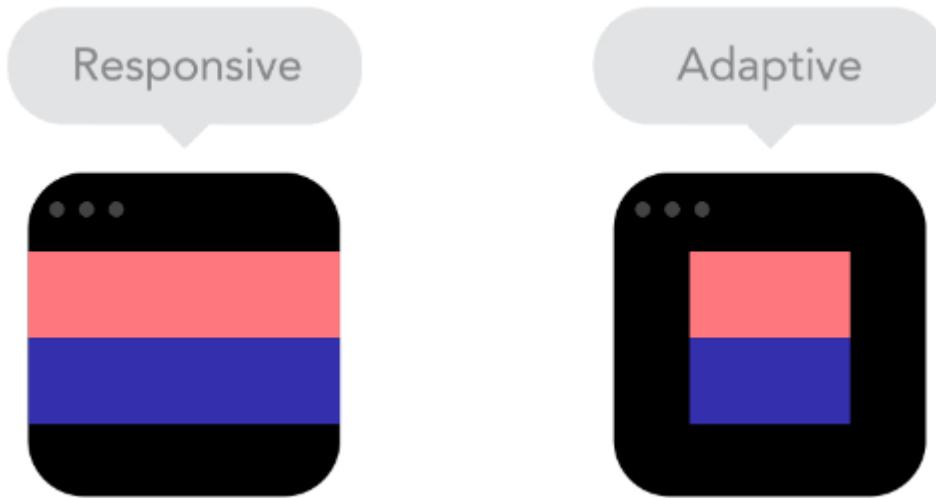
Recuerda visitar el la sección de **Recursos** para ver más utilidades web donde generar fuentes, pares de fuentes y tipografías fluidas.

Web responsive

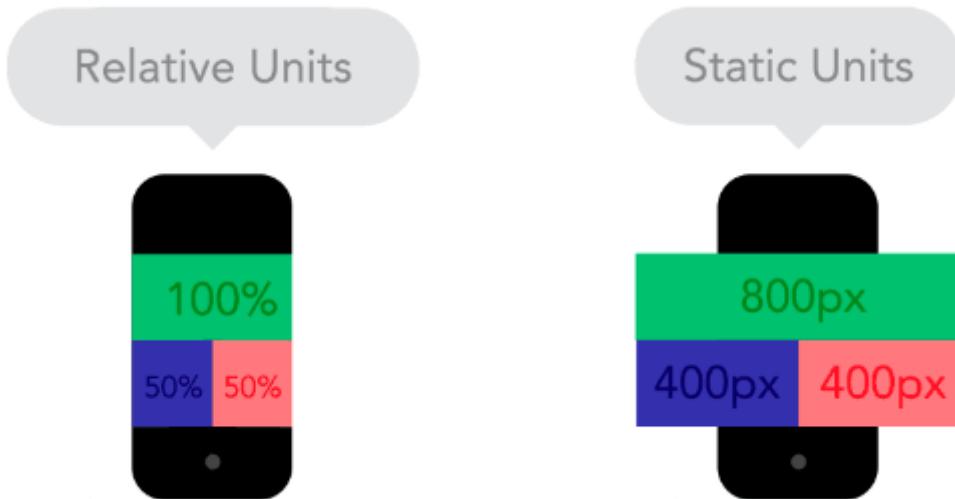
En **diseño responsive** tendremos que hacer hincapié en varios conceptos.

El primero de ellos es la diferencia entre **diseño responsive** y **diseño adaptativo**.

Como se puede ver en la imagen a continuación, un diseño **responsive** responde en todo momento a las dimensiones del dispositivo, mientras que un diseño adaptable es aquel que se adapta, pero no necesariamente responde en todo momento.

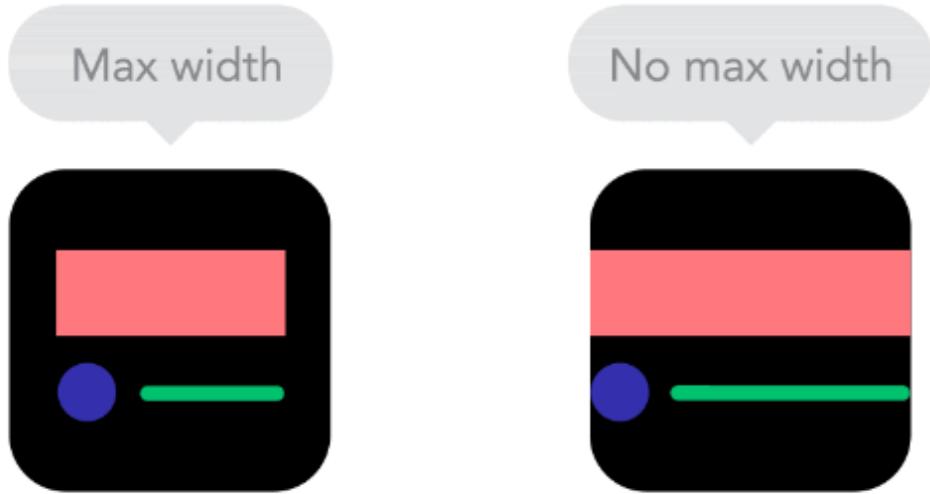


Por otro lado, para trabajar correctamente en diseños **responsive** hay que tener en cuenta que debemos trabajar con unidades relativas e intentar evitar las unidades fijas o estáticas, las cuales no responden a la adaptación de nuestros diseños flexibles.



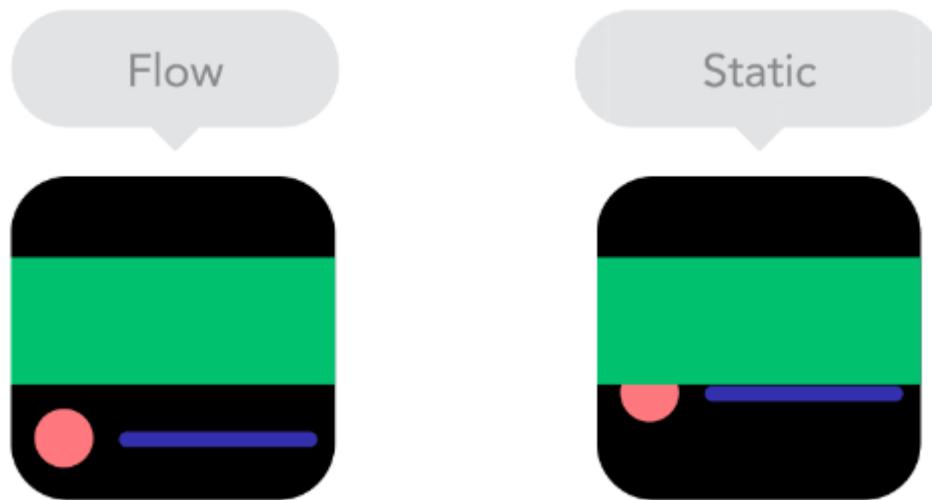
Otra forma interesante de trabajar esa respuesta de los diseños **responsive** es utilizar propiedades como **min-width** o **max-width**, donde definimos tamaños mínimos o máximos, para que los elementos de nuestra página puedan ampliar o reducirse según sea necesario dependiendo de la pantalla del dispositivo utilizado.

Con estas propiedades podemos crear diseños que aprovechen al máximo toda la pantalla de dispositivos pequeños (*como móviles o tablets*), mientras que establecemos unos máximos en pantallas de dispositivos grandes, para crear unos espacios visuales que hacen que el diseño sea más agradable.



Otro concepto, que a la misma vez es una característica muy atractiva en nuestros diseños responsive es la de mantener el flujo de los elementos cuando cambian de tamaño y evitar que estos se solapen unos con otros.

Si estamos habituados a trabajar en diseños más estáticos que no están preparados para móviles, suele ser duro hacer ese cambio. Sin embargo, una vez lo conseguimos, todo resulta mucho más fácil y conseguiremos transmitir una buena respuesta y fluidez visual.

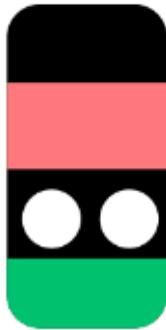


Esto último va muy de la mano del sistema habitual de recolocación de elementos que se suele seguir en los diseños **Responsive Design**.

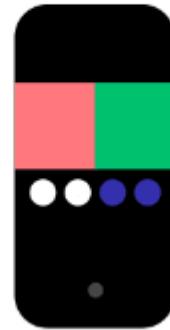
Como se puede ver en la siguiente imagen, en un diseño responsive se utilizan ciertos «puntos de control».

Por ejemplo, se suele pensar que en una resolución de escritorio queremos mostrar la información dentro de una cuadrícula (*grid*) de 4 ó 5 celdas de ancho, mientras que en la versión de tablet será sólo de 3 celdas de ancho (*el resto se desplazará a la siguiente fila*) y en móviles será una sola celda de ancho, mostrándose el resto de celdas haciendo scroll hacia abajo:

With Breakpoints



Without Breakpoints



Esta forma de trabajar nos proporciona múltiples ventajas:

- Es mucho más sencillo mostrar la misma información desde diseños de pantalla grande.
- Ayuda a evitar la mala práctica de ocultar bloques de información en dispositivos móviles.
- Incentiva a diseñar siguiendo buenas prácticas para facilitar la creación responsive.

Estrategias de diseño

Por último, es aconsejable decidirse por una estrategia de diseño antes de comenzar. Aunque existen otras estrategias, las dos vertientes principales son las siguientes.

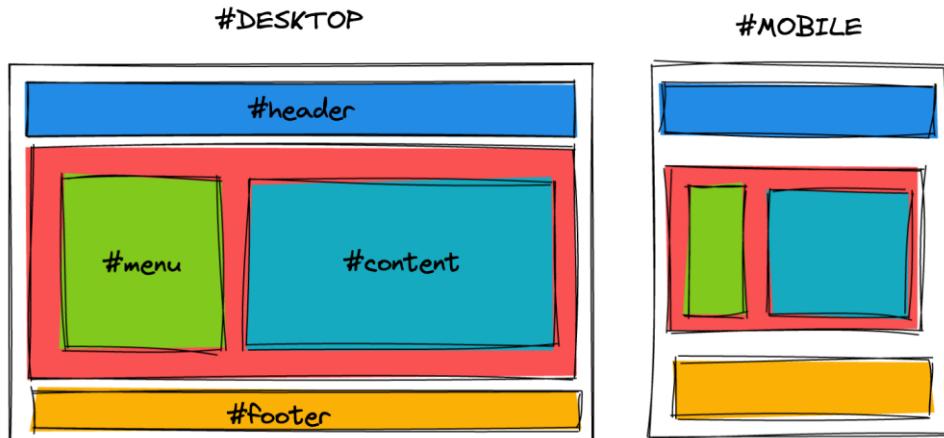
- **Mobile first** - Primero nos enfocamos en dispositivos móviles y luego pensamos en otros.
- **Desktop first** - Primero nos enfocamos en dispositivos de escritorio, y luego pensamos en otros.

Bases del Responsive Design

Hay ciertos conceptos que hay que tener claros antes de comenzar con el **Responsive Design**. En esta sección vamos a ver como llevarlos a la práctica con código.

El primer paso para crear un diseño que se adapte correctamente, es comenzar a familiarizarse con un tipo de unidades relativas: **los porcentajes**. Recordemos que los porcentajes son relativos al contenedor padre, por lo que si especificamos un porcentaje a un elemento, el navegador va a tomar dicho porcentaje del contenedor.

Podemos comenzar usando porcentajes con las propiedades **width** en un ejemplo sencillo. Si establecemos por un lado un ancho de **100%** a los elementos **azules**, **rojos** y **amarillos**, y por otro lado al elemento **verde** un **30%** y al turquesa un **70%** podemos tener un diseño responsive.



Prueba

```
<div id="header"></div>
<div id="page">
  <div id="menu"></div>
  <div id="content"></div>
</div>
<div id="footer"></div>
```

Sin embargo, utilizar porcentajes no nos garantiza un diseño adaptativo de calidad, hay que comprender otros detalles.

El primer problema que encontraremos será que si sumamos el tamaño de los elementos (70% + 30%) junto a los bordes (*2px por cada lado*), la suma es superior al 100% del contenedor padre, por lo que **no cabe en su interior** y el segundo elemento se desplaza a la zona inferior, descuadrando todo el diseño.

Lo mismo puede ocurrir si intentamos añadir **margin** o **padding**.

Esto es algo muy habitual en CSS.

Y frustrante al principio.

Hay varias formas de solucionar esto:

- Eliminar los bordes y reducir los porcentajes hasta que quepan en el 100% del parente.
- Usar **box-sizing: border-box** para cambiar el modo en el que se gestionan los tamaños.
- Utilizar un sistema moderno como Flexbox o Grid (*recomendado*).

Una forma simple de solucionar el problema en el ejemplo anterior, es hacer los siguientes cambios en el CSS del documento.

```
/* Eliminamos este bloque */
#menu, #content {
  display: inline-block;
}
```

```
/* Añadimos este */
#page {
```

```
    display: flex;  
}
```

De esta forma, conseguimos que nuestro diseño se adapte de forma adecuada a la página, sin necesidad de tener que ajustar los márgenes, rellenos, bordes o tamaño de los contenidos.

Más adelante ahondaremos en Flex y Grid.

min-width y max-width

Si buscamos un cierto grado de control aún mayor, podríamos recurrir a las propiedades **max-width** y **min-width**, con las que podemos indicar el ancho de un elemento como máximo y el ancho de un elemento como mínimo respectivamente, consiguiendo así garantizar cierto control del diseño.

```
.picture {  
    min-height: 200px; /* Por defecto, height es 0 */  
    background: grey; /* Simplemente, para verlo visualmente */  
  
    max-width: 1024px;  
    min-width: 800px;  
}
```

En este caso, el elemento tiene un tamaño máximo de 1024 píxeles, y un tamaño mínimo de 800 píxeles, por lo que si ajustamos el ancho de la ventana del navegador, dicho elemento iría variando en un rango de 800 a 1024 píxeles, nunca haciéndose más pequeño de 800 o más grande de 1024.

Con las imágenes, videos y contenidos multimedia, se puede hacer lo mismo, consiguiendo así que las imágenes se escalen y adapten al formato especificado o incluso al tamaño de pantalla de los diferentes dispositivos utilizados.

```
img,  
video,  
object,  
embed {  
    max-width: 100%;  
    height: auto;  
}
```

meta

Con esta etiqueta <meta>, estamos estableciendo unos parámetros de comportamiento para el **viewport** del navegador. Veamos qué significan y cuáles más existen.

Propiedad	Valor	Significado
width	device-width	Indica un ancho para el viewport.
height	device-height	Indica un alto para el viewport.
initial-scale	1	Escala inicial con la que se visualiza la página web.

minimum-scale	0.1	Escala mínima a la que se puede reducir al hacer zoom.
maximum-scale	10	Escala máxima a la que se puede aumentar al hacer zoom.
user-scalable	no/fixed	yes/zoom

Las propiedades **initial-scale**, **minimum-scale** y **maximum-scale** permiten valores desde el **0.1** al **10**, aunque ciertos valores se traducen automáticamente a ciertos números determinados:

- yes = 1
- no = 0.1
- device-width = 10
- device-height = 10

Por otra parte, **user-scalable** permite definir si es posible que el usuario pueda «pellizcar» la pantalla para ampliar o reducir el zoom.

Es una mala práctica prohibir el zoom al usuario, ya que muchos usuarios lo necesitan para ver el contenido.

MediasQueries

Una vez nos adentramos en el mundo del **Responsive Design**, nos damos cuenta en que hay situaciones en las que determinados aspectos o componentes visuales deben aparecer en un tipo de dispositivos, o deben existir ciertas diferencias.

Para ello, utilizaremos un concepto denominado **media queries**, con los que podemos hacer esas excepciones para que sólo se apliquen a un tipo de diseño concreto.

Las reglas **media queries** (*también denominadas MQ a veces*) son un tipo de reglas de CSS que permiten crear un bloque de código que sólo se procesará en los dispositivos que cumplan los criterios especificados como condición.

```
@media screen and (*condición*) {
/* reglas CSS */
/* reglas CSS */
}
```

```
@media screen and not (*condición*) {
/* reglas CSS */
/* reglas CSS */
}
```

Con este método, especificamos qué queremos aplicar los estilos CSS para tipos de medios concretos (**screen**: sólo en pantallas, en este caso) que cumplan las condiciones especificadas entre paréntesis.

Ejemplos Medias Queries

Veamos un ejemplo clásico de **media queries** en el que definimos diferentes estilos dependiendo del dispositivo que estamos utilizando.

Observad que en el código existen 3 bloques **@media** donde se definen estilos CSS para cada uno de esos tipos de dispositivos.

```
@media screen and (max-width: 640px) {  
    .menu {  
        background: blue;  
    }  
}  
  
@media screen and (min-width: 640px) and (max-width: 1280px) {  
    .menu {  
        background: red;  
    }  
}  
  
@media screen and (min-width: 1280px) {  
    .menu {  
        background: green;  
    }  
}
```

El ejemplo anterior muestra un elemento (con clase menu) con un color de fondo concreto, dependiendo del tipo de medio con el que se visualice la página:

- **Azul** para resoluciones menores a **640 píxeles** de ancho (*móviles*).
- **Rojo** para resoluciones entre **640 píxeles** y **1280 píxeles** de ancho (*tablets*).
- **Verde** para resoluciones mayores a **1280 píxeles** (*desktop*).

El número de bloques de reglas **@media** que se utilicen depende del desarrollador web, ya que no es obligatorio utilizar un número concreto. Se pueden utilizar desde un sólo media query, hasta múltiples de ellos a lo largo de todo el documento CSS.

Display básico

Es importante que entiendas que cada elemento HTML en una pagina web es una caja rectangular, esta es la forma en que se representan todos los elementos, no existen elementos triangulares, redondos, poligonales etc. Todos los elementos en HTML por defecto son rectangulares ya que internamente el navegador dibuja un rectángulo.

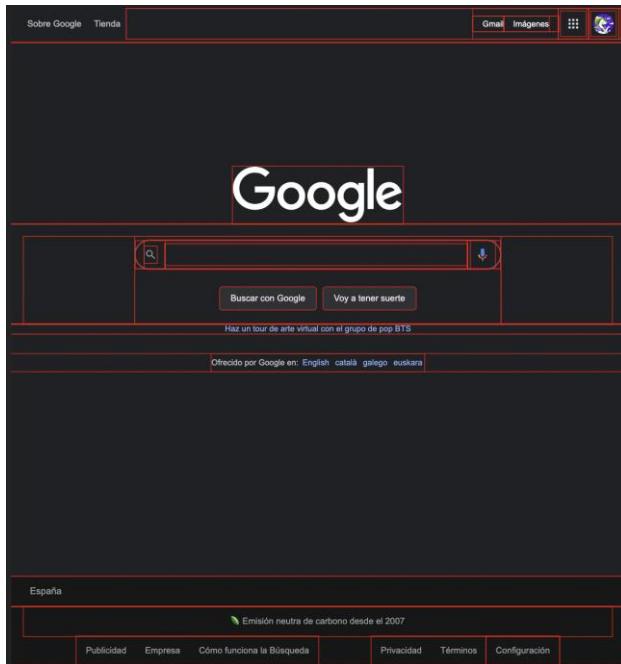
Inline serán los elementos en línea, que se colocan uno a continuación de otro de forma horizontal. Ocuparán el ancho necesario y no todo el ancho disponible como lo harán los elementos block o de bloque. Estos elementos ocuparán la totalidad del ancho y se llaman así porque se entienden como un bloque de contenido.



En el siguiente ejemplo, con ayuda de la propiedad **outline**, podemos ver todos los elementos de la pagina de inicio de Google de forma rectangular y esto lo puedes hacer con cualquier otro sitio web.

Para ello en el navegador - click izquierdo inspeccionar y meter el estilo por la consola. En nuestro caso sería atacar a todas las etiquetas **div** y meter el **outline**.

```
*{  
    outline: 1px solid red;  
}
```



Os dejamos un ejemplo de los comportamientos principales de la mayoría de elementos HTML, que serán inline y block:

```
<div>Este es un ejemplo de comportamiento de bloque</div>
<span>Este es un ejemplo de comportamiento en línea</span>
```

```
div {
    background-color: blueviolet;
}
span {
    background-color: steelblue;
}
```

Como vemos en el ejemplo, un div será un ejemplo de elemento que se comporta como un bloque y un span será un ejemplo de comportamiento en línea.

Este es un ejemplo de comportamiento de bloque

Este es un ejemplo de comportamiento en línea

Ahora bien, el comportamiento por defecto de los elementos se puede modificar con la propiedad `display`, que dependiendo del valor que debemos modificará de una forma u otra el comportamiento del elemento.

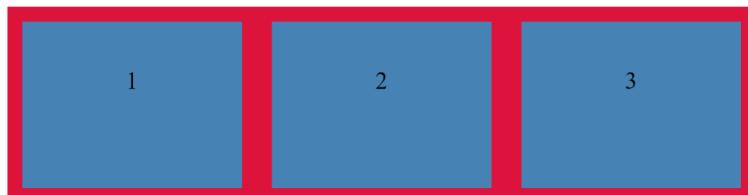
Vamos a ver los tipos:

- **block:** hace que el elemento ocupe la totalidad del ancho, por lo que los elementos se apilarán en vertical y en una línea solo podrá existir un elemento.
- **inline:** adapta el ancho del elemento al contenido del mismo. Como veímos en el ejemplo anterior de código, el `` ocupa únicamente lo que ocupa su contenido. Estos elementos no tienen en cuenta los valores de `width` o `height` que les añadamos.

- **inline-block**: Combina los dos anteriores, ya que permite que creamos bloques pero del width y height que queramos.
- **flex**: utiliza un modelo de cajas flexibles que será una sección en sí misma debido a su riqueza de contenido.
- **grid**: al igual que flex, será una sección en sí misma, pero lo importante es que usa cuadrículas o “grillas” en su modelo para repartir el contenido.
- **contents**: necesitaremos nociones de flex y grid para usarlo - corretamente. Será de ayuda a la hora de mantener la relación padre-hijo que es necesaria en los elementos con display flex/grid. Este valor se le suele dar a la propiedad display por necesidades de javascript.
- **none**: ocultará el elemento y nos será muy útil cuando lo mezclamos con javascript, para renderizados condicionales o en los que en algunos casos debamos mostrar unos elementos y en otros casos otros elementos distintos. Es un comportamiento similar a la propiedad visibility con el valor de hidden, solo que visibility además hace “desaparecer” el elemento, por lo que no se verá en nuestro árbol HTML o DOM.

```
div{
  margin: 0;
  padding: 0;
  display: block;
  background-color: crimson;
}
span {
  background-color: steelblue;
  width: 100px;
  height: 50px;
  padding: 30px;
  list-style-type: none;
  text-align: center;
  margin: 10px 10px;
}
```

Lo único que tendrás que hacer es cambiar el valor de la propiedad display de block a las que os hemos ido explicando para ver los resultados y que os hagáis una mejor idea.



Display: Flex

Organizar o posicionar elementos en nuestra página es algo muy importante que nos permitirá realizar cualquier diseño que nos propongamos.

Display: flex Es una propiedad de CSS que se utiliza para establecer un contenedor como flexible o "flexbox". Tiene bastante contenido y es importante por eso que aprendamos un poco de la terminología y el concepto que podemos utilizar, ¡vamos a ello!

Terminología básica de flex

Es importante que utilicemos los mismos nombres de los términos sobre todo en la fase de aprendizaje para no liarnos, ya que con tantos nombres de diferentes cosas que estamos aprendiendo a veces perdemos el foco de lo que realmente es, así que aunque no lo anotéis en ningún sitio es importante que estos términos os suenen y sepáis a qué se refieren.

Contenedor (Container)

Un contenedor Flex puede contener otros elementos, como cajas, texto, imágenes, etc.

Los contenedores son utilizados para organizar y estructurar la disposición de los elementos en una aplicación.

Elemento (Item o Element)

Un elemento se refiere a un elemento como su propio nombre indica, individual dentro de un contenedor.

Puede ser cualquier cosa, desde texto hasta gráficos, botones, campos de entrada, etc.

Caja (Box)

Una caja es un tipo de contenedor que se utiliza para agrupar elementos de manera horizontal o vertical. Puede contener varios elementos y controlar su distribución en el espacio disponible.

Espacio disponible (Available Space)

Es el espacio dentro de un contenedor que está disponible para colocar elementos. La gestión de este espacio es fundamental en Flex, ya que los elementos se ajustan automáticamente según el espacio disponible.

Dirección principal (Main Axis)

Se refiere a la dirección principal de distribución en un contenedor Flex.

En una distribución **horizontal**, la dirección principal es horizontal, en una **vertical**, la dirección principal es vertical.

Dirección secundaria (Cross Axis)

Se comporta de manera contraria a la dirección principal, es decir, en una distribución horizontal, la dirección secundaria es vertical y viceversa.

Propiedades de contenedor Flex

Como toda propiedad, flex tiene sus características únicas, es decir, hay ciertas propiedades que se le pueden dar a un contenedor o a sus elementos hijos directos únicamente cuando un contenedor tiene la propiedad: "**display: flex**", en este caso, nos centraremos en las propiedades que se le pueden añadir al contenedor (padre).

Flex-direction

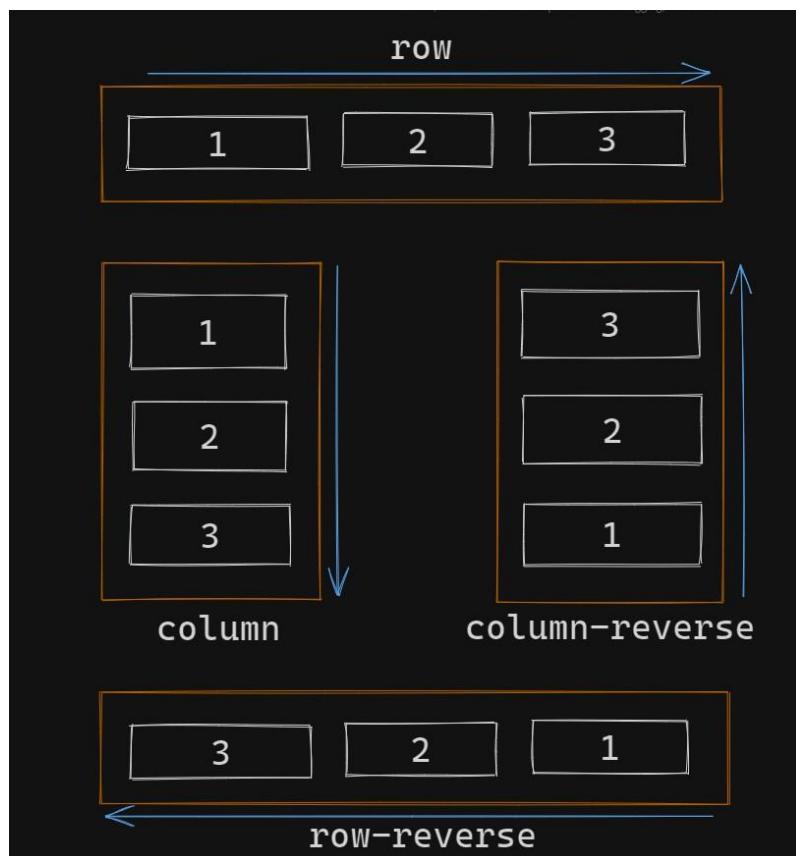
La propiedad `flex-direction` tiene los siguientes valores. **flex-direction** es una propiedad que afecta a los elementos hijos directos.

1 **row**: en línea (**valor por defecto**)

2 **column**: en columna

3 **row-reverse**: en línea del revés

4 **column-reverse**: en columna del revés



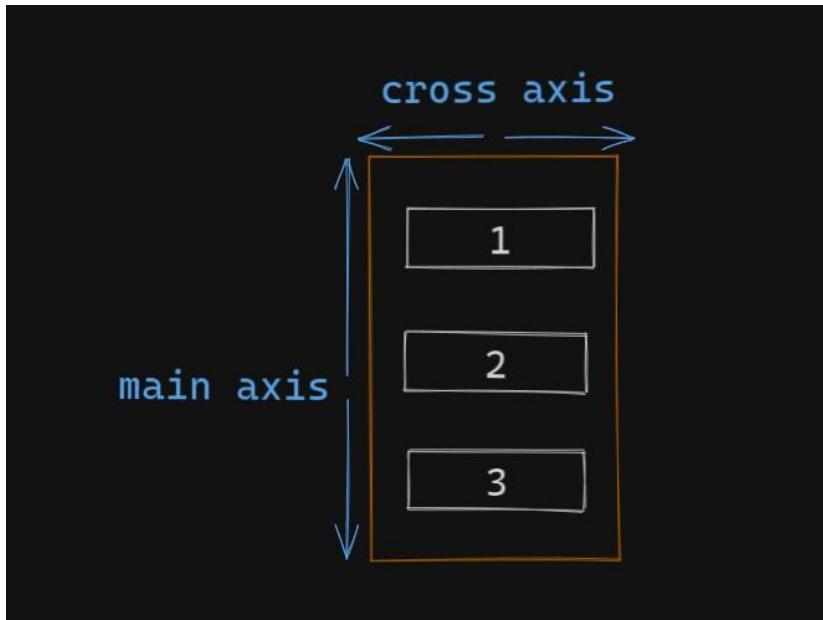
Eje principal

Como comentamos anteriormente el eje principal se define mediante las anteriores propiedades.

Ejes en row o row-reverse



Ejes en column o column-reverse



Flex-wrap

La propiedad `flex-wrap` nos permitirá "envolver" nuestros elementos, el valor por defecto de esta propiedad es: "**flex-wrap: no-wrap**"

¿Para que sirve "envolver" nuestros elementos? sirve por si queremos que se pasen hacia abajo o hacia algún lado los elementos interiores cuando no quepan dentro del contenedor padre, por ejemplo:



Valores de flex-wrap

wrap: envuelve para que si no caben se pasen a la siguiente línea o columna

wrap-reverse: hace lo mismo que wrap pero del revés

no-wrap: no pasa los elementos a la siguiente línea o columna aunque no quepan en el contenedor padre

Abreviación Flex-flow

Es una abreviación de las dos propiedades vistas anteriormente, sirve para hacer procesos más rápidamente pero no hace nada especial o distinto que no hayamos visto previamente.

Como podemos ver la sintaxis combina tanto el `flex-direction` con cualquiera de sus valores y la propiedad `flex-wrap` en cualquiera de sus valores, os invitamos a combinarlo como en el ejemplo.

```
flex-flow: row wrap;
```

Align-items

La propiedad align-items, nos permitirá "alinear" nuestros elementos hijos en el **cross axis** que ya hemos visto que depende de la **flex-direction**.

Valores de align-items

1 **stretch**: los elementos se ajustan por defecto a la altura de aquel más alto (**valor por defecto**)

2 **flex-start o start**: los elementos se alinean al comienzo del contenedor flex

3 **flex-end o end**: los elementos se alinean al final del contenedor flex

4 **center**: los elementos se alinean al centro del contenedor flex

Como siempre os invitamos a probarlo para que comprobéis su funcionamiento.

Align-content

Esta propiedad se aplica a contenedores que tienen múltiples filas o columnas de elementos, y ajusta el espacio entre esas filas o columnas en función de su contenido.

Los valores son exactamente las mismas que en justify-content.

Propiedades de un elemento flex

Como ya sabemos, un elemento sería un item dentro de un contenedor flex.

Estos items **heredan** la posibilidad de tener propiedades de flex, vamos a ver unas cuantas.

Align-self

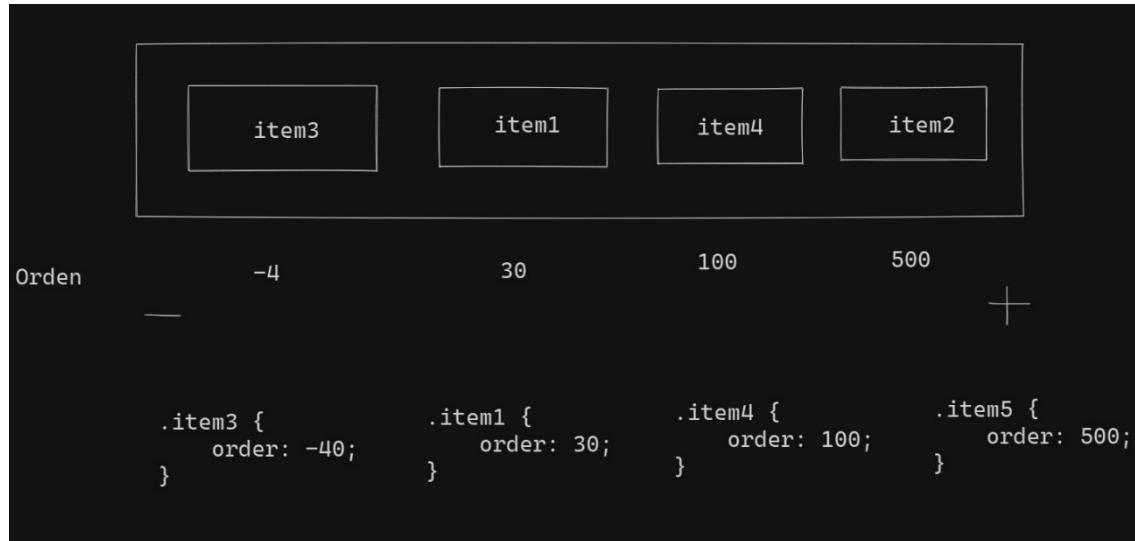
Align-self será como el align items pero para un único elemento, es decir, esta propiedad se pondrá en el elemento que queramos alinear en concreto, por ejemplo:

```
.item1 {  
    align-self: flex-start;  
}  
  
.item2 {  
    align-self: center;  
}  
  
.item3 {  
    align-self: flex-end;  
}
```

Order

Es una propiedad bastante curiosa, **su valor por defecto es: 0**

Según los valores que le demos a los diferentes items se posicionarán de arriba a abajo o izquierda a derecha de mayor a menor, además, no hace falta que los números o valores que le pongamos a la propiedad vayan en orden, es decir, puede ser **cualquier valor numérico**, simplemente se ordenarán de menor a mayor valor, por ejemplo:



flex-basis

Es una propiedad bastante similar al `width` que me permite manejar el ancho de mi elemento y admite cualquier unidad de medida, pero únicamente la podremos utilizar en elementos afectados por `display: flex`, por ejemplo:



flex-grow

La propiedad `flex-grow` permite recibir cualquier número.

Esta propiedad es un **producto de crecimiento**, es decir, cuando nuestra página **crezca**, los elementos que tengan valor en `flex-grow` irán creciendo, cuanto más número más espacio crecen respecto al resto.

El valor inicial de esta propiedad es 0, esto quiere decir que por defecto no crece, para este no hay imagen puesto que lo veremos en el video mucho mejor ya que podremos jugar con el ancho de la pantalla a nuestro gusto

flex-shrink

La propiedad flex-shrink, será lo contrario de flex-grow, es decir, esta propiedad es un **producto de decrecimiento**, por lo tanto, cuando nuestra página **decrezca**, los elementos que tengan valor en flex-shrink irán decreciendo, cuanto más número más espacio decrecen respecto al resto.

El valor inicial de esta propiedad es 1, esto quiere decir que por defecto decrecerán todos los elementos por igual, para este tampoco hay imagen puesto que lo veremos en el video mucho mejor ya que podremos jugar con el ancho de la pantalla a nuestro gusto.

flex

Es una propiedad que mezcla las 3 vistas anteriormente, es decir, es una abreviación de estas tres

¿Cómo funciona?: `flex: <flex-grow> <flex-shrink> <flex-basis>`

Por lo tanto...

Podríamos usarla así: `flex: 1 2 200px;`

Herramientas para dominar flexbox

En este video podéis ver las diferentes herramientas que nos vendrán genial para poner en práctica todo lo explicado, os recomendamos hacerlas todas para aprender de una forma más amena.

 [Flex-Froggy](#)

 [FlexBox-Defense](#)

 [FlexBox-Adventure](#)

Accesibilidad

Hemos visto diversas formas de darle la vuelta a una columna, a una fila o desordenar los elementos. Pero es muy importante indicar que es solo a nivel visual, ya que HTML seguirá en el mismo orden.

Por lo tanto, si queremos ordenar algo por orden alfabético, invertir el orden de unos datos para leerlos del final al principio o algo similar, tendrá que ser a nivel lógico (JavaScript), y no a nivel visual (CSS).

Ya que alguien puede ver el resultado, pero un lector de pantalla lo va a leer tal y como está en HTML.

Podéis hacer la prueba con un column-reverse o row-reverse inspeccionando como el estilo y el HTML no concuerdan.

Estructuras Flex

Ahora no hay mejor manera de ver cómo aplicar lo visto de Flex que con ejemplos.

Así que vamos a ver ejemplos muy comunes donde podemos utilizar "`display: flex`" en nuestro día a día.

Barra de navegación

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Navbar Flex</title>
  <link rel="stylesheet" href="style.css">
  <link rel="shortcut icon" href="./assets/logo.png" type="image/x-icon">
</head>
<body>
  <header>
    
    <nav>
      <ul class="nav-list">
        <li>
          <a href="#">Home</a>
        </li>
        <li>
          <a href="#">About</a>
        </li>
        <li>
          <a href="#">Gallery</a>
        </li>
        <li>
          <a href="#">Contact Us</a>
        </li>
      </ul>
    </nav>
    
  </header>
</body>
</html>

header {
  display: flex;
  padding: 40px;
  align-items: center;
  justify-content: space-between;
}

header > img {
  width: 50px;
  height: 50px;
}

header > nav {
  flex-grow: 1;
  display: flex;
```

```

justify-content: center;
}

.nav-list {
  display: flex;
  justify-content: space-evenly;
  width: 100%;
  list-style: none;
}

a {
  color: black;
  font-weight: 600;
  text-decoration: none;
}

.menu {
  display: none;
  width: 30px;
  height: 30px;
}

@media (max-width: 500px) {
  header > nav {
    display: none;
  }
  .menu {
    display: block;
  }
}

Hero

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Hero</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <main>
    <section id="hero">
      <div class="info">
        <h1>Experienced <span>mobile and web</span> applications and website builders measuring.</h1>
        <p>KODEX TECHNOLOGY (PVT) LTD is a team of experienced mobile and web applications and website builders measuring dozens of completed projects. We build and develop mobile applications for several top platforms, including Android & IOS. </p>
      </div>
    </section>
  </main>
</body>

```

```
<div class="buttons">
  <button>Contact Us</button>
  <button>View More</button>
</div>
</div>

</section>
</main>
</body>
</html>

@import
url('https://fonts.googleapis.com/css2?family=Roboto:wght@100;300;400;500;700;900&display=swap');

* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: 'Roboto', sans-serif;
}

:root {
  --rtc-primary-color: #1090CB;
  --rtc-secondary-color: #000000;
  --rtc-tertiary-color: #FFFFFF;
  --rtc-background-color: #1090cb2b;
}

#hero {
  background-color: var(--rtc-background-color);
  display: flex;
  align-items: center;
  min-height: 100svh;
  justify-content: space-around;
  flex-wrap: wrap;
  padding: 50px;
}

/* izquierda */
.info {
  width: 40%;
  min-width: 516px;
}

/* derecha */
#hero > img {
  width: 30%;
  max-width: 500px;
  min-width: 300px;
```

```
}

.info > h1 {
    font-size: 40px;
}

.info > h1 > span {
    color: var(--rtc-primary-color);
}

.buttons {
    display: flex;
    margin-top: 30px;
    gap: 30px;
}

button {
    border-radius: 10px;
    padding: 15px 25px;
    cursor: pointer;
    border: none;
}

.buttons > button:first-child {
    background-color: var(--rtc-primary-color);
    color: var(--rtc-tertiary-color);
    border: 1px solid var(--rtc-primary-color);
}

.buttons > button:last-child {
    background-color: var(--rtc-tertiary-color);
    color: var(--rtc-primary-color);
    border: 1px solid var(--rtc-primary-color);
}

@media (max-width: 570px) {
    .info > h1 {
        font-size: 30px;
    }

    .info {
        width: 100%;
        min-width: 0px;
    }
}
```

Galeria

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Galería de fotos</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<ul>
<li>

</li>
</ul>
</body>
</html>

* {
  margin: 0;
  padding: 0;

```

```

    box-sizing: border-box;
}

ul {
    display: flex;
    flex-wrap: wrap;
    padding: 20px;
    list-style-type: none;
    gap: 20px;
}

ul > li {
    flex-basis: 350px;
    flex-grow: 1;
}

ul > li > img {
    width: 100%;
}

```

Formularios

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Formulario</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <h2>Registro</h2>
        <form action="#" method="post">
            <div class="form-group">
                <label for="nombre">Nombre:</label>
                <input type="text" id="nombre" name="nombre" required>
            </div>
            <div class="form-group">
                <label for="email">Correo Electrónico:</label>
                <input type="email" id="email" name="email" required>
            </div>
            <div class="form-group">
                <label for="password">Contaseña:</label>
                <input type="password" id="password" name="password" required>
            </div>
            <div class="form-group">
                <button type="submit">Registrarse</button>
            </div>
        </form>
    </div>

```

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

body {
    font-family: Arial, sans-serif;
    background-color: #f5f5f5;
    height: 100vh;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
}

.container {
    background-color: white;
    padding: 20px;
    border-radius: 5px;
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 10px;
    border: 2px solid black;
}

.form-group {
    width: 100%;
    display: flex;
    flex-direction: column;
    gap: 5px;
    padding-bottom: 5px;
}

.form-group > input {
    padding: 8px;
    border: 2px solid black;
    border-radius: 5px;
}

.form-group > button {
    padding: 8px;
    border: 2px solid black;
    border-radius: 5px;
    background-color: blue;
    color: white;
}
```

Footer

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Footer</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <footer>
        <ul>
            <li>Contact</li>
            <li>Gallery</li>
            <li>About</li>
            <li>Testimonials</li>
        </ul>
        <ul>
            <li>Contact</li>
            <li>Gallery</li>
            <li>About</li>
            <li>Testimonials</li>
        </ul>
        <ul>
            <li>Contact</li>
            <li>Gallery</li>
            <li>About</li>
            <li>Testimonials</li>
        </ul>
    </footer>
</body>
</html>

@import
url('https://fonts.googleapis.com/css2?family=Roboto:wght@100;300;400;500;700;900&display=swap');

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Roboto', sans-serif;
}

footer {
    position: fixed;
    bottom: 0;
    left: 0;
    width: 100%;
```

```
display: flex;
padding: 50px;
justify-content: space-around;
gap: 50px;
flex-wrap: wrap;
background-color: antiquewhite;
}

ul {
list-style: none;
width: 250px;
}

ul > li:first-child {
font-weight: bolder;
}



## Layout



<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Layout Flexbox</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<header>Header</header>
<div class="container">
<main>Main</main>
<nav>Nav</nav>
<aside>Aside</aside>
</div>
<footer>Footer</footer>
</body>
</html>

body {
margin: 0;
padding: 0;
font-family: Arial, sans-serif;
min-height: 100vh;
}

.container {
min-height: 80vh;
display: flex;
justify-content: space-between;
}
```

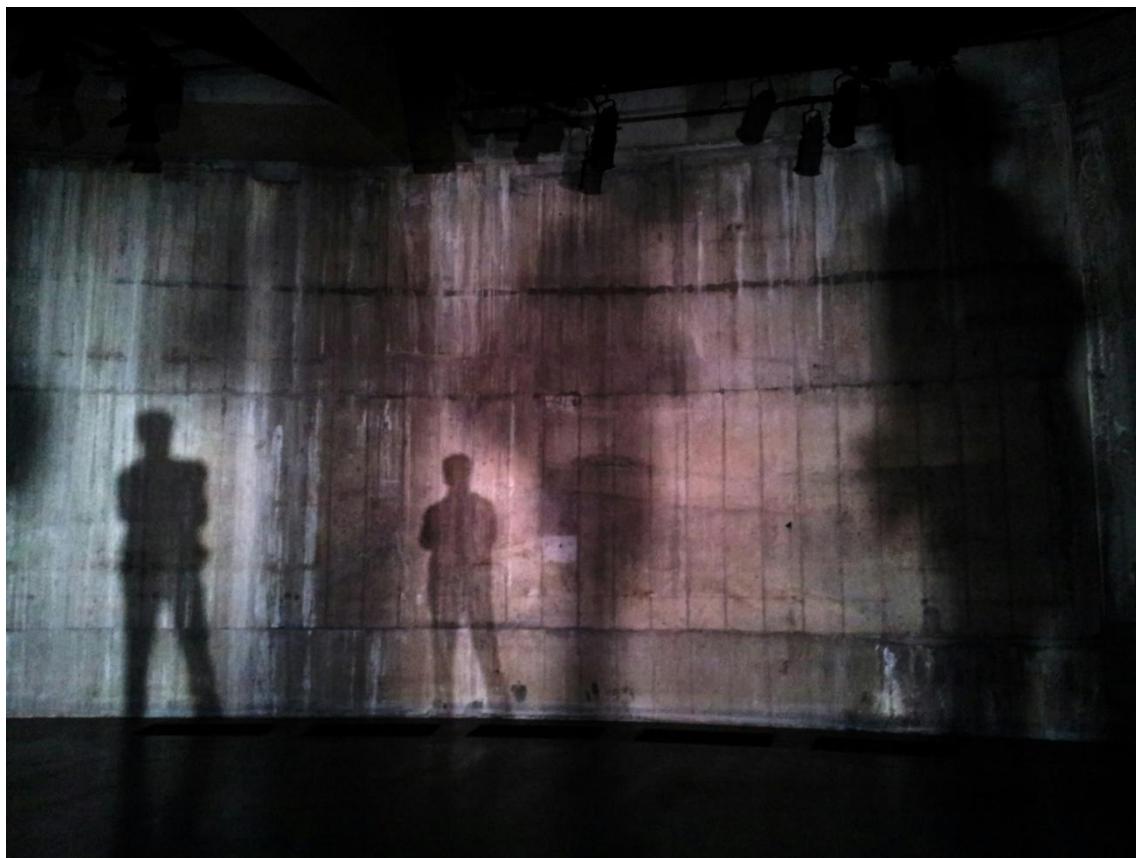
```
header, footer {  
    height: 10vh;  
    background-color: crimson;  
}  
  
main {  
    flex: 2;  
    background-color: green;  
}  
  
nav, aside {  
    flex: 1;  
}  
  
@media screen and (max-width: 700px) {  
    .container {  
        flex-direction: column;  
    }  
  
    main, nav, aside {  
        flex: auto;  
    }  
}
```

Sombras y animaciones

Propiedades de sombras

Se denominan **sombra sobre cajas** a las sombras en CSS que se pueden crear en una etiqueta o elemento HTML. Para ello, se utiliza la propiedad **box-shadow**.

Las sombras se pueden crear en una etiqueta o elemento HTML para darle efecto a nuestras cajas. Para ello tenemos la propiedad box-shadow, que por defecto están desactivadas o con valor none y con el color negro sin difuminar. La sombra se podrá extender o desplazar en horizontal y/o vertical. Además podremos difuminar, desenfocar, aplicar crecimiento... todo ello dependerá de cómo juguemos con la propiedad box-shadow.



Vamos a ver cada posibilidad:

- **Desplazamiento:** los dos primeros parámetros que son obligatorios en la propiedad box-shadow indicamos el desplazamiento de la sombra en el eje x e y. Si queremos desplazar la sombra a la derecha y abajo, podríamos hacer lo siguiente:

```
.element {  
  box-shadow: 10px 10px;  
}  
/* Desplazamientos izquierda y arriba, ponemos valores negativos*/  
.element {  
  box-shadow: -5px -5px;  
}
```

- **Desenfoque:** este será el tercer parámetro que podemos pasarle a la propiedad, e indica la cantidad de desenfoque o difuminado que queremos en nuestra sombra. Por defecto es 0, que quiere decir que no estará difuminada.

```
.element {
  box-shadow: 5px 5px 0; /* Sombra sin desenfoque */
  box-shadow: 5px 5px 2px; /* Sombra con ligero desenfoque */
  box-shadow: 5px 5px 10px; /* Sombra desenfocada */
  box-shadow: 5px 5px 40px; /* Sombra con un desenfoque casi disipado */
}
```

- **Factor de crecimiento:** es el cuarto parámetro que admite la propiedad y es opcional, y hará crecer la sombra en todos sus lados.

```
.element {
  box-shadow: 0 0 10px 5px;
}
```

- **Color:** es el último parámetro admitido y normalmente se suele usar el cuarto ya que el factor de crecimiento no es muy habitual.

Ahora bien, hasta ahora hemos hablado mucho de sombras pero no hemos mencionado que por defecto la sombra será exterior y se colocará por fuera del elemento. Si queremos una sombra interior e invertir los valores por defecto tenemos la palabra clave inset.

Al invertir los valores debemos tener en cuenta que ahora los valores que eran positivos en el eje x deberán ser negativos. Vamos a ver mejor un ejemplo de código:

```
.element {
  /* Sombra exterior que se desplaza hacia la zona inferior-derecha */
  box-shadow: 10px 10px 5px black;

  /* Sombra interior que se desplaza hacia la zona inferior-derecha */
  box-shadow: -10px -10px 5px black inset;
}
```

Vamos a mostráros un ejemplo completo de código para que veáis de lo que estamos hablando:

```
<div class="shadow-example">Este es un ejemplo de aplicación de sombras en un div. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Modi magni temporibus, eaque et praesentium quidem, iste veritatis, tenetur itaque similique voluptatem quae doloribus maxime dolores blanditiis ex ut illum aut?. Lorem ipsum dolor sit amet consectetur adipisicing elit. Ducimus quibusdam nam sint. Dolor repellendus dolorum reprehenderit voluptate ullam vel sed, accusantium similique quae atque cupiditate fuga, id quam eos nulla! Lorem ipsum, dolor sit amet consectetur adipisicing elit. Molestiae sint est aliquam perferendis aspernatur doloribus quo dolorem quos dicta molestias nemo possimus eius hic culpa modi, quia vero accusamus. Quis.</div>
```

```
.shadow-example {
  box-shadow: 10px 10px 5px black;
  width: 500px;
  margin: auto;
  height: 100%;
```

Como podéis ver se ha aplicado la sombra a nuestra caja. Podéis jugar con los valores de la propiedad box-shadow para ver cómo cambia la sombra!

Este es un ejemplo de aplicación de sombras en un div. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Modi magni temporibus, eaque et praesentium quidem, iste veritatis, tenetur itaque similique voluptatem quae doloribus maxime dolores blanditiis ex ut illum aut?. Lorem ipsum dolor sit amet consectetur adipisicing elit. Ducimus quibusdam nam sint. Dolor repellendus dolorum reprehenderit voluptate ullam vel sed, accusantium similique quae atque cupiditate fuga, id quam eos nulla! Lorem ipsum, dolor sit amet consectetur adipisicing elit. Molestiae sint est aliquam perferendis aspernatur doloribus quo dolorem quos dicta molestias nemo possimus eius hic culpa modi, quia vero accusamus. Quis.

Si queréis practicar más las sombras os dejamos esta web en la que podéis interactuar directamente para que podáis entenderlo mejor:

[Box Shadow](#)

Para lograr un efecto de sombra "**drop shadow**" podemos utilizar la propiedad filter en CSS. Esta propiedad permite agregar sombras a imágenes o elementos, creando un efecto similar al de la sombra natural proyectada.

Aquí podréis ver la diferencia entre box-shadow y drop-shadow:



```
img {  
width: 200px;  
}
```

```
#box {  
box-shadow: 10px 10px 10px black;  
}
```

```
#drop {  
    filter: drop-shadow(10px 10px 10px black);  
}
```

Animaciones

Las **animaciones** en CSS nos permiten crear transiciones entre estilos de un elemento de nuestra página web.

Estas transiciones se pueden crear utilizando la propiedad "animation" y sus sub-propiedades, y se pueden aplicar a cualquier elemento del DOM.

Definición de keyframes

Para crear una animación en CSS, primero debemos definir un conjunto de estilos para el elemento en diferentes puntos del tiempo, estos estilos se llaman "**keyframes**".

Para definir un conjunto de keyframes, utilizamos la regla "@keyframes". Por ejemplo, para definir una animación que cambie el color de fondo de un elemento de rojo a azul, podemos hacer lo siguiente:

```
@keyframes cambiarColor {  
    from {  
        background-color: red;  
    }  
    to {  
        background-color: blue;  
    }  
}
```

En este ejemplo, hemos definido un conjunto de keyframes llamado "cambiarColor" que cambia el color de fondo de rojo a azul.

El "from" indica el estado inicial del elemento y el "to" indica el estado final.

También podemos definir keyframes en porcentajes.

Por ejemplo, si queremos crear una animación que cambie el color de fondo de un elemento de rojo a azul en los primeros 50% y luego de azul a verde en los siguientes 50%, podemos hacer lo siguiente:

```
@keyframes cambiarColor {  
    0% {  
        background-color: red;  
    }  
    50% {  
        background-color: blue;  
    }  
    100% {  
        background-color: green;  
    }  
}
```

En este ejemplo, hemos definido un conjunto de keyframes llamado "cambiarColor" que cambia el color de fondo de rojo a azul en los primeros 50% y luego de azul a verde en los siguientes 50%.

Aplicación de la animación

Una vez que hemos definido nuestros keyframes, podemos aplicarlos a un elemento utilizando la propiedad "animation". Por ejemplo, para aplicar la animación "cambiarColor" a un elemento con la clase "mi-elemento" durante 2 segundos, podemos hacer lo siguiente:

```
.mi-elemento {  
    animation: cambiarColor 2s;  
}
```

En este ejemplo, hemos aplicado la animación "cambiarColor" al elemento con la clase "mi-elemento" durante 2 segundos.

Control de la animación

La propiedad "animation" tiene varias sub-propiedades que nos permiten controlar aspectos como la duración, el retraso, la cantidad de veces que se repite la animación, etc. Algunas de estas sub-propiedades son:

- "**animation-duration- "**animation-delay- "**animation-iteration-count- "**animation-timing-function********

Por ejemplo, podemos modificar nuestra animación anterior para que se repita indefinidamente y tenga un retraso de 1 segundo antes de comenzar de la siguiente manera:

```
.mi-elemento {  
    animation: cambiarColor 2s infinite;  
    animation-delay: 1s;  
}
```

En este ejemplo, hemos aplicado la animación "cambiarColor" al elemento con la clase "mi-elemento" durante 2 segundos, y hemos indicado que se repita indefinidamente y tenga un retraso de 1 segundo antes de comenzar.

Ejemplos prácticos

- **Cambio de color:** Podemos crear una animación para cambiar el color de fondo de un elemento de rojo a azul en 2 segundos de la siguiente manera:

```
@keyframes cambiarColor {  
    from {  
        background-color: red;  
    }  
    to {  
        background-color: blue;  
    }  
}
```

```
.mi-elemento {  
    animation: cambiarColor 2s;  
}
```

- **Desplazamiento:** Podemos crear una animación para desplazar un elemento hacia la derecha en 200 píxeles en 2 segundos de la siguiente manera:

```
@keyframes desplazar {  
    from {  
        margin-left: 0px;  
    }  
    to {  
        margin-left: 200px;  
    }  
}
```

```
.mi-elemento {  
    animation: desplazar 2s;  
}
```

- **Cambio de opacidad:** Podemos crear una animación para cambiar la opacidad de un elemento de 0 a 1 en 2 segundos de la siguiente manera:

```
@keyframes aparecer {  
    from {  
        opacity: 0;  
    }  
    to {  
        opacity: 1;  
    }  
}
```

```
.mi-elemento {  
    animation: aparecer 2s;  
}
```

Y recordad, somos desarrolladores, no animadores. Podemos coger inspiración de otros medios o webs, pero hay que tener en cuenta que en muchos casos menos es más.

Las animaciones son muy llamativas, pero abusar de ellas puede implicar que el usuario encuentre innavegable nuestra aplicación. Sed sutiles.

Filtros

Los **filtros** CSS te permiten transformar y mejorar la visualización de elementos en tu aplicación web.

Son como herramientas mágicas que te otorgan control sobre la apariencia de imágenes, textos y otros elementos, permitiéndote crear efectos sorprendentes como si un editor de fotos se tratase.



Se pueden usar para:

- **Cambiar el brillo, contraste y saturación:** Ajusta la intensidad de los colores y la claridad de la imagen.
- **Agregar desenfoque o nitidez:** Suaviza o resalta los detalles de la imagen.
- **Aplicar efectos de distorsión:** Crea efectos de inversión de colores, sombras, etc...
- **Colorear elementos:** Cambia el color de un elemento o aplica un tono sepia, por ejemplo.

Los filtros CSS se aplican a los elementos mediante la propiedad **filter** en la declaración de una regla.

La propiedad filter acepta una lista de funciones de filtro, cada una con sus propios parámetros. En este ejemplo, se aplica un filtro de desenfoque de 5px a la clase imagen.

```
.imagen {  
  filter:  
    blur(5px);  
}
```

Ventajas de usar filtros CSS

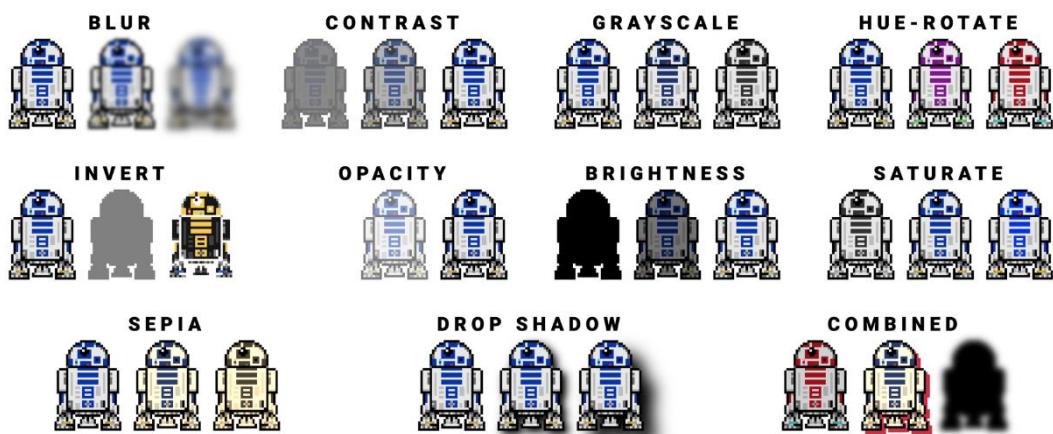
- **Mejora la experiencia del usuario:** Los filtros pueden usarse para crear efectos visuales llamativos y mejorar la accesibilidad.

- **Reduce el tamaño de archivo:** En algunos casos, usar filtros puede ser una alternativa más eficiente que usar imágenes con efectos predefinidos.
- **Flexibilidad y dinamismo:** Los filtros se pueden aplicar de forma dinámica, lo que permite crear efectos interactivos.

Desventajas de usar filtros CSS

- **Compatibilidad:** No todos los navegadores son compatibles con todos los filtros.
- **Rendimiento:** Los filtros complejos pueden afectar el rendimiento de la página.

Os dejamos con un "laboratorio" de filtros creado con HTML y CSS con el que podréis visualizar perfectamente todos los filtros, probar el código dentro del mismo y utilizarlos en vuestros proyectos tomándolo como referencia:



```

<main>
  <div>
    <h2>Blur</h2>
    <article>
      
      
      
    </article>
  </div>

```

```
</div>
<div>
  <h2>Contrast</h2>
  <article>
    <img
      src=<http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>">
      alt="R2D2"
      class="no-contrast"
    />
    <img
      src=<http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>">
      alt="R2D2"
      class="some-contrast"
    />
    <img
      src=<http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>">
      alt="R2D2"
      class="much-contrast"
    />
  </article>
</div>
<div>
  <h2>Grayscale</h2>
  <article>
    <img
      src=<http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>">
      alt="R2D2"
      class="no-grayscale"
    />
    <img
      src=<http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>">
      alt="R2D2"
      class="some-grayscale"
    />
    <img
      src=<http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>">
      alt="R2D2"
      class="much-grayscale"
    />
  </article>
</div>
<div>
  <h2>Hue-Rotate</h2>
```

```
<article>
<img
  src=<"http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png">
  alt="R2D2"
  class="no-hue"
/>
<img
  src=<"http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png">
  alt="R2D2"
  class="some-hue"
/>
<img
  src=<"http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png">
  alt="R2D2"
  class="much-hue"
/>
</article>
</div>
<div>
<h2>Invert</h2>
<article>
<img
  src=<"http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png">
  alt="R2D2"
  class="no-invert"
/>
<img
  src=<"http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png">
  alt="R2D2"
  class="some-invert"
/>
<img
  src=<"http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png">
  alt="R2D2"
  class="much-invert"
/>
</article>
</div>
<div>
<h2>Opacity</h2>
<article>
<img
  src=<"http://pixelartmaker-data-
```

78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>"
 alt="R2D2"
 class="no-opacity"
 />
 <img
 src="" alt="R2D2" class="some-opacity"/>
 />
 <img
 src="" alt="R2D2" class="much-opacity"/>
 />
 </article>
 </div>
 <div>
 <h2>Brightness</h2>
 <article>
 <img
 src="" alt="R2D2" class="no-brightness"/>
 />
 <img
 src="" alt="R2D2" class="some-brightness"/>
 />
 <img
 src="" alt="R2D2" class="much-brightness"/>
 />
 </article>
 </div>
 <div>
 <h2>Saturate</h2>
 <article>
 <img
 src="" alt="R2D2" class="no-saturate"/>

```
/>
<img
  src=<http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>
  alt="R2D2"
  class="some-saturate"
/>
<img
  src=<http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>
  alt="R2D2"
  class="much-saturate"
/>
</article>
</div>
<div>
<h2>Sepia</h2>
<article>
<img
  src=<http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>
  alt="R2D2"
  class="no-sepia"
/>
<img
  src=<http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>
  alt="R2D2"
  class="some-sepia"
/>
<img
  src=<http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>
  alt="R2D2"
  class="much-sepia"
/>
</article>
</div>
<div>
<h2>Drop Shadow</h2>
<article>
<img
  src=<http://pixelartmaker-data-
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>
  alt="R2D2"
  class="no-shadow"
/>
<img
  src=<http://pixelartmaker-data-
```

```
78746291193.nyc3.digitaloceanspaces.com/image/1c5de380c0d107a.png>"  
    alt="R2D2"  
    class="some-shadow"  
  />  
    
  </article>  
</div>  
<div>  
  <h2>Combined</h2>  
  <article>  
      
      
      
  </article>  
</div>  
</main>  
  
@import url("<a href="https://fonts.googleapis.com/css2?family=Roboto:wght@700;900&display=swap"></a>");  
  
* {  
  margin: 0;  
  padding: 0;  
  box-sizing: border-box;  
  font-family: "Roboto", sans-serif;  
}  
  
h2 {  
  text-transform: uppercase;  
  letter-spacing: 6px;  
  font-weight: 900;
```

```
}
```

```
main {  
  display: flex;  
  justify-content: space-around;  
  align-items: center;  
  flex-wrap: wrap;  
  padding: 1rem 2rem;  
  cursor: crosshair;  
}
```

```
div {  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  justify-content: center;  
  margin: 1rem;  
}
```

```
img {  
  width: 6rem;  
  transition: all 0.3s ease-in-out;  
}  
  
img:hover {
```

```
  transform: scale(1.1);  
}
```

```
/*FILTERS*/
```

```
/*BLUR*/  
.no-blur {  
  filter: blur(0px);  
}
```

```
.some-blur {  
  filter: blur(2px);  
}
```

```
.much-blur {  
  filter: blur(5px);  
}
```

```
/*CONTRAST*/
```

```
.no-contrast {  
  filter: contrast(10%);  
}
```

```
.some-contrast {
  filter: contrast(50%);
}

.much-contrast {
  filter: contrast(100%);
}

/*GRAYSCALE*/
.no-grayscale {
  filter: grayscale(0%);
}

.some-grayscale {
  filter: grayscale(50%);
}

.much-grayscale {
  filter: grayscale(100%);
}

/*HUE-ROTATE*/
.no-hue {
  filter: hue-rotate(0deg);
}

.some-hue {
  filter: hue-rotate(60deg);
}

.much-hue {
  filter: hue-rotate(120deg);
}

/*INVERT*/
.no-invert {
  filter: invert(0%);
}

.some-invert {
  filter: invert(50%);
}

.much-invert {
  filter: invert(100%);
}

/*OPACITY*/
.no-opacity {
```

```
filter: opacity(0%);  
}  
  
.some-opacity {  
filter: opacity(50%);  
}  
  
.much-opacity {  
filter: opacity(100%);  
}  
  
/*BRIGHTNESS*/  
.no-brightness {  
filter: brightness(0%);  
}  
  
.some-brightness {  
filter: brightness(50%);  
}  
  
.much-brightness {  
filter: brightness(100%);  
}  
  
/*SATURATE */  
.no-saturate {  
filter: saturate(0%);  
}  
  
.some-saturate {  
filter: saturate(100%);  
}  
  
.much-saturate {  
filter: saturate(150%);  
}  
  
/*SEPIA*/  
.no-sepia {  
filter: sepia(0%);  
}  
  
.some-sepia {  
filter: sepia(50%);  
}  
  
.much-sepia {  
filter: sepia(100%);  
}
```

```
/*DROP SHADOW*/
.no-shadow {
  filter: drop-shadow(0px 0px 0px black);
}

.some-shadow {
  filter: drop-shadow(5px 5px 5px black);
}

.much-shadow {
  filter: drop-shadow(16px 16px 10px black);
}

/*COMBINED*/
.first-combined {
  filter: brightness(90%) hue-rotate(120deg);
}

.second-combined {
  filter: sepia(50%) drop-shadow(5px 5px 0px crimson);
}

.third-combined {
  filter: brightness(0%) blur(4px);
}
```

Prueba a aplicar filtros a elementos en tu aplicación y modifica elementos visuales en vez de hacerlo de manera externa con editores de imágenes, texto, etc...

Recursos

Os dejamos de manera desglosada una serie de recursos para CSS3 que os pueden ayudar a la hora de maquetar todas las aplicaciones web.



Fuentes

- **Fluid Type Scale** → Gracias a la función clamp nos permite tener variables con tamaños ajustables a la resolución de la pantalla

[Fluid Type Scale - Generate responsive font-size variables](#)

- **Google Fonts** → Una librería de fuentes gratuitas de Google, también incluye iconos

[Fonts Knowledge - Google Fonts](#)

- **Font Joy** → Genera pares de fuentes

[Generate font pairing using neural nets](#)

Colores

- **Huemint** → Generador de paletas para cualquier tipo de aplicación

[Huemint - Generate a unique 4-color palette for your website](#)

- **HappyHues** → Generador de paletas de colores adecuadas a accesibilidad

[Happy Hues - Curated colors in context.](#)

- **CSS Gradient** → Generador de gradientes

[CSS Gradient - Generator, Maker, and Background](#)

- **RandomA11y** → Generador de paletas de colores accesibles

[RandomA11y](#)

Iconos

- **Flaticon**

[Iconos y stickers gratuitos - Millones de recursos para descargar](#)

- **Iconos8**

[Iconos Gratis, Ilustraciones Clipart, Fotos y Música](#)

- **Google Fonts**

[Material Symbols and Icons - Google Fonts](#)

Fondos

- **Transparent Textures** → Fondos transparentes

[Transparent Textures](#)

- **The Pattern Library** → Patrones de fondo

[Pattern Library](#)

Herramientas

- **Clippy** → Generador de clip-path

[Clippy - CSS clip-path maker](#)

- **CSSMatic** → Generador de sombras

[Box Shadow](#)

Flex

- **CSS Tricks** → Guía de Flexbox con ejemplos visuales

[A Complete Guide to Flexbox | CSS-Tricks](#)

- **Flex Cheatsheet** → Guia rápida de Flexbox

[Flex Cheatsheet](#)

- **Flexbox Froggy** → Juego de la rana para practicar Flexbox

[Flexbox Froggy](#)

- **Flexbox Adventure** → Juego del caballero medieval para practicar Flexbox

[Play Flex Box Adventure - CSS Game to Learn Flexbox](#)

Grid

- **CSS Garden** → Juego de jardinero para practicar Grid

[Grid Garden](#)

- **CSS Grid Attack** → Juego de lucha para practicar Grid

[Play Grid Attack - CSS Game to learn CSS Grid](#)