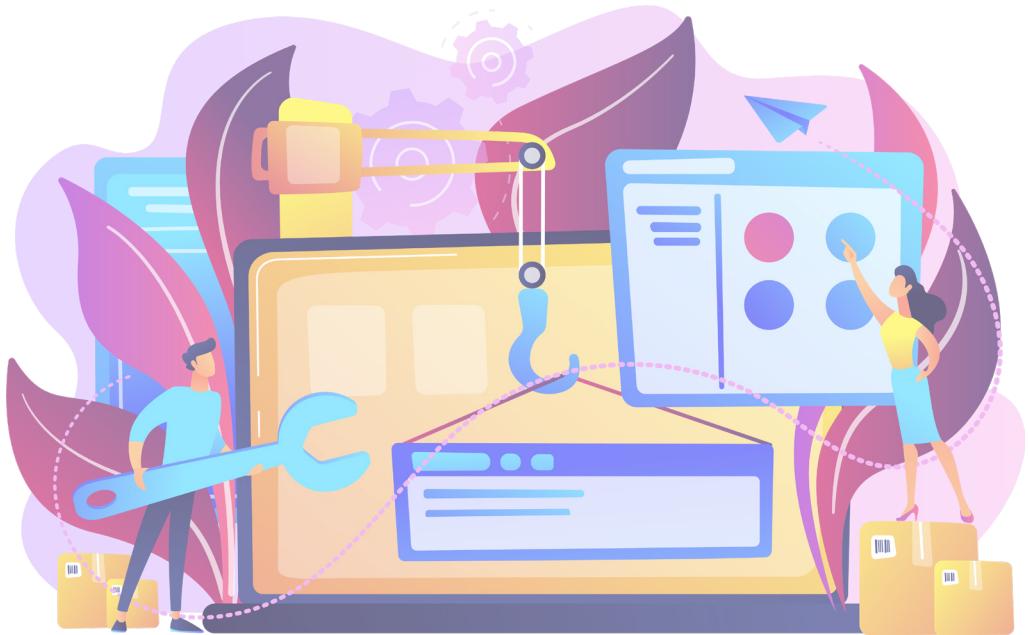




# *Semana 3* **INTRODUCCIÓN A CSS**

*Qué es CSS | Origen | Selectores | Unidades de medida | Colores | Textos | Modelos de cajas | Flexbox | Grid | Metodología BEM.*

3



# Índice

---

¿Qué es CSS? .....	2
¿Cómo incluir CSS en un documento HTML? .....	2
Selectores .....	4
Unidades de medida .....	9
Colores (ej. background, text, etc) .....	11
Texto .....	17
Modelo de cajas .....	18
Flexbox .....	23
Grid .....	34
Metodología BEM .....	45

## ¿Qué es CSS?

**CSS mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes.**

**CSS es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos HTML y XHTML.** CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas. Separar la definición de los contenidos y la definición de su aspecto presenta numerosas ventajas, ya que obliga a crear documentos HTML/XHTML bien definidos y con significado completo (también llamados "documentos semánticos").

Al crear una página web, se utiliza en primer lugar el lenguaje HTML/XHTML para marcar los **contenidos**, es decir, para designar la función de cada elemento dentro de la página: párrafo, titular, texto destacado, tabla, lista de elementos, etc. Una vez creados los contenidos, se utiliza el lenguaje CSS para definir el aspecto de cada elemento: color, tamaño y tipo de letra del texto, separación horizontal y vertical entre elementos, posición de cada elemento dentro de la página, etc.

### Especificación oficial

La especificación o norma oficial que se utiliza actualmente para diseñar páginas web con CSS se puede consultar libremente en <http://www.w3.org/> El sitio web del organismo W3C dispone de una sección en la que se detalla el trabajo que el W3C está desarrollando actualmente en relación a CSS y también dispone de un blog en el que se publican todas las novedades relacionadas con CSS (<http://www.w3.org/blog/CSS>).

**CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.**

## ¿Cómo incluir CSS en un documento HTML?

**Se pueden crear todos los archivos CSS que sean necesarios y cada página HTML puede enlazar tantos archivos CSS como necesite.**

Si se quieren incluir los estilos del ejemplo anterior en un archivo CSS externo, se deben seguir los siguientes pasos:

1. Se crea un archivo de texto y se le añade solamente el siguiente contenido:  
`p { color: blue; font-family: Verdana; }.`
2. Se crea un archivo de texto y se le añade

solamente el siguiente contenido:  
`p { color: blue; font-family: Verdana; }.`

3. En la página HTML se enlaza el archivo CSS externo mediante la etiqueta:

```

1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Informatorio | Etapa 1</title>
8      <link rel="stylesheet" type="text/css" href="style.css" media="screen">
9  </head>
10
11 <body>
12
13     <h1 class="titulo_uno">Bienvenidos al Informatorio Etapa 1</h1>
14
15 </body>
16
17 </html>
```

Cuando el navegador carga la página HTML anterior, antes de mostrar sus contenidos también descarga los archivos CSS externos enlazados mediante la etiqueta y aplica los estilos a los contenidos de la página.

**Normalmente, la etiqueta incluye cuatro atributos cuando enlaza un archivo CSS:**

**rel:** indica el **tipo de relación que existe entre el recurso enlazado** (en este caso, el archivo CSS) **y la página HTML**. Para los archivos CSS, siempre se utiliza el valor `stylesheet`.

**type:** indica el **tipo de recurso enlazado**. Sus valores están estandarizados y para los archivos CSS su valor siempre es `text/css`.

**href:** indica la **URL del archivo CSS que contiene los estilos**. La URL indicada puede ser relativa o absoluta y puede apuntar a un recurso interno o externo al sitio web.

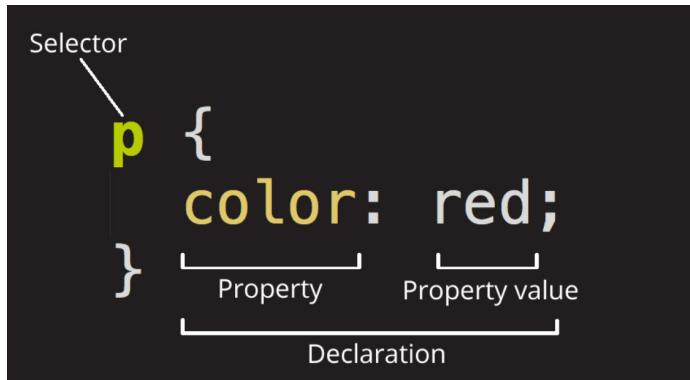
**media:** indica el **medio en el que se van a aplicar los estilos del archivo CSS**. Más adelante se explican en detalle los medios CSS y su funcionamiento.

De todas las formas de incluir CSS en las páginas HTML, esta es la más utilizada con mucha diferencia. La principal ventaja es que se puede incluir un mismo archivo CSS en multitud de páginas HTML, por lo que se garantiza la aplicación homogénea de los mismos estilos a todas las páginas que forman un sitio web.

Con este método, el mantenimiento del sitio web se simplifica al máximo, ya que un solo cambio en un solo archivo CSS permite variar de forma instantánea los estilos de todas las páginas HTML que enlazan ese archivo.

## Glosario

CSS define una serie de términos que permiten describir cada una de las partes que componen los estilos CSS. El siguiente esquema muestra las partes que forman un estilo CSS muy básico:



**Selector:** indica el elemento o elementos HTML a los que se aplica la regla CSS.  
**Declaration:** especifica los estilos que se aplican a los elementos. Está compuesta por una o más propiedades CSS.

**Property:** característica que se modifica en el elemento seleccionado, como por ejemplo su tamaño de letra, su color de fondo, etc.

**Property Value:** establece el nuevo valor de la característica modificada en el elemento.

Los diferentes términos se definen a continuación:

## Selectores

Para crear diseños web profesionales, es imprescindible conocer y dominar los selectores de CSS. Una regla de CSS está formada por una parte llamada "selector" y otra parte llamada "declaración" o "declaration".

En simples palabras:

**DECLARACIÓN** ..... ➔ "QUÉ HAY QUE HACER"

INDICA

**SELECTOR** ..... ➔ "A QUIÉN HAY QUE HACÉRSELO"

Por lo tanto, los selectores son imprescindibles para aplicar de forma correcta los estilos CSS en una página.

A un mismo elemento HTML se le pueden asignar infinitas reglas CSS y cada regla CSS puede aplicarse a un número infinito de elementos. En otras palabras, una misma regla puede aplicarse sobre varios selectores y un mismo selector se puede utilizar en varias reglas.

El estándar de CSS3 incluye una docena de tipos diferentes de selectores, que permiten seleccionar de forma muy precisa elementos individuales o conjuntos de elementos dentro de una página web.

No obstante, la mayoría de páginas de los sitios web se pueden diseñar utilizando solamente los cinco selectores básicos.

## Selectores básicos

### 1 Selector universal:

Se utiliza para seleccionar todos los elementos de la página. El siguiente ejemplo elimina el margen y el relleno de todos los elementos HTML (por ahora no es importante fijarse en la parte de la declaración de la regla CSS):

```

1   * {
2     margin: 0;
3     padding: 0;
4 }
```

El selector universal se indica mediante un asterisco (\*). A pesar de su sencillez, no se utiliza habitualmente, ya que es difícil que un mismo estilo se pueda aplicar a todos los elementos de una página.

No obstante, sí que se suele combinar con otros selectores y además, forma parte de algunos hacks muy utilizados, como se verá más adelante.

### 2 Selector de tipo o etiqueta:

Selecciona todos los elementos de la página cuya etiqueta HTML coincide con el valor del selector. El siguiente ejemplo selecciona todos los párrafos de la página:

```

1   p {
2     /* Declaraciones */
3 }
```

Para utilizar este selector, solamente es necesario indicar el nombre de una etiqueta HTML (sin los caracteres < y >) correspondiente a los elementos que se quieren seleccionar.

El siguiente ejemplo aplica diferentes estilos a los titulares y a los párrafos de una página HTML (ver *Imagen 5*).

```

1   h1 {
2     color: red;
3 }
4   h2 {
5     color: blue;
6 }
7   h3 {
8     color: black;
9 }
```

*Imagen 5*

Si se quiere aplicar los mismos estilos a dos etiquetas diferentes, se pueden encadenar los selectores.

En el siguiente ejemplo, los títulos de sección h1, h2 y h3 comparten los mismos estilos (ver *Imagen 6*):

```

1  h1 {
2      color: red;
3      font-weight: normal;
4      font-family: Arial, Helvetica, sans-serif;
5  }
6
7  h2 {
8      color: red;
9      font-weight: normal;
10     font-family: Arial, Helvetica, sans-serif;
11  }
12
13 h3 {
14     color: red;
15     font-weight: normal;
16     font-family: Arial, Helvetica, sans-serif;
17  }
18

```

Imagen 6

En este caso, CSS permite agrupar todas las reglas individuales en una sola regla con un selector múltiple. Para ello, se incluyen todos los selectores separados por una coma (,) y el resultado es que la siguiente regla CSS es equivalente a las tres reglas anteriores.

```

1  h1, h2, h3 {
2      color: red;
3      font-weight: normal;
4      font-family: Arial, Helvetica, sans-serif;
5  }
6

```

En las hojas de estilo complejas, es habitual agrupar las propiedades comunes de varios elementos en una única regla CSS y posteriormente definir las propiedades específicas de esos mismos elementos.

```

1  h1, h2, h3 {
2      color: red;
3      font-weight: normal;
4      font-family: Arial, Helvetica, sans-serif;
5  }
6
7  h1 {
8      font-size: 18px;
9  }
10
11 h2 {
12     font-size: 16px;
13 }
14
15 h3 {
16     font-size: 14px;
17 }
18

```

### 3 Selector de clase:

Si se considera el siguiente código HTML de ejemplo:

```

11 <body>
12     <p>Lorem ipsum dolor sit adipisicing elit.</p>
13     <p>Natus inventore labore tempore eveniet totam</p>
14     <p>autem cumque nostrum quam.</p>
15 </body>

```

**¿Cómo se pueden aplicar estilos CSS sólo al primer párrafo?**

El selector universal (\*) no se puede utilizar porque seleccionaría todos los párrafos.

El selector de tipo o etiqueta (p) tampoco se puede utilizar porque seleccionaría todos los párrafos.

Una de las soluciones más sencillas para apli-

car estilos a un solo elemento de la página consiste en utilizar el atributo class de HTML sobre ese elemento para indicar directamente la regla CSS que se le debe aplicar:

```

11  <body>
12    <p class="destacado">Lorem ipsum dolor sit adipisicing elit.</p>
13    <p>Natus inventore labore tempore eveniet totam</p>
14    <p>autem cumque nostrum quam.</p>
15  </body>
```

A continuación, se crea en el archivo CSS una nueva regla llamada destacado con todos los estilos que se van a aplicar al elemento. Para que el navegador no confunda este selector con los otros tipos de selectores, se prefija el valor del atributo class con un punto (.) tal y como muestra el siguiente ejemplo:

```

1  .destacado {
2    color: red;
3 }
```

*Imagen 11*

El selector **.destacado** (ver *Imagen 11*) se interpreta como “cualquier elemento de la página cuyo atributo class sea igual a destacado”, por lo que solamente el primer párrafo cumple esa condición.

Este tipo de selectores se llaman **selectores de clase y son los más utilizados junto con los selectores de ID que se verán a continuación**. La principal característica de este selector es que en una misma página HTML varios elementos diferentes pueden utilizar el mismo valor en el atributo class (ver *Imagen 12*):

```

11  <body>
12    <p class="destacado">Lorem ipsum dolor sit adipisicing elit.</p>
13    <p>Natus inventore <a href="#" class="destacado">labore tempore</a> eveniet totam</p>
14    <p>autem cumque <em class="destacado">nostrum</em> quam.</p>
15  </body>
```

*Imagen 12*

***Los selectores de clase son imprescindibles para diseñar páginas web complejas, ya que permiten disponer de una precisión total al seleccionar los elementos. Además, estos selectores permiten reutilizar los mismos estilos para varios elementos diferentes.***

#### 4 Selectores de ID:

En ocasiones, es necesario aplicar estilos CSS a un único elemento de la página. Aunque puede utilizarse un selector de clase para aplicar estilos a un único elemento, existe otro selector más eficiente en este caso.

El selector de ID **permite seleccionar un elemento de la página a través del valor de su atributo id**. Este tipo de selectores sólo seleccionan un elemento de la página porque el valor del atributo id no se puede repetir en dos elementos diferentes de una misma página.

La **sintaxis** de los selectores de ID es muy parecida a la de los selectores de clase, salvo que se utiliza el **símbolo de numeral (#)** en vez del punto (.) como prefijo del nombre de la regla CSS:

```
11 <body>
12   <div id="contenido">
13     <p>Informatorio</p>
14   </div>
15 </body>
```

```
1 #contenido {
2   width: 200px;
3   height: 200px;
4   border: 2px dotted;
5   background-color: red;
6 }
```

En el ejemplo anterior, el selector `#contenido` da estructura a la etiqueta div (cuyo atributo id es igual a contenido).

La principal diferencia entre este tipo de selector y el selector de clase tiene que ver con HTML y no con CSS. Como se sabe, en una misma página, el valor del atributo id debe ser único, de forma que dos elementos diferentes no pueden tener el mismo valor de id. Sin embargo, el atributo class no es obligatorio que sea único, de forma que muchos elementos HTML diferentes pueden compartir el mismo valor para su atributo class.

De esta forma, la recomendación general es la de utilizar el selector de ID cuando se quiere aplicar un estilo a un solo elemento específico de la página y utilizar el selector de clase cuando se quiere aplicar un estilo a varios elementos diferentes de la página HTML.

*La sintaxis de los selectores de ID es muy parecida a la de los selectores de clase, salvo que se utiliza el símbolo de numeral (#) en vez del punto (.) como prefijo del nombre de la regla CSS...*

#### 3 Combinación de selectores básicos

CSS permite la combinación de uno o más tipos de selectores para restringir el alcance de las reglas CSS. A continuación se muestran algunos ejemplos habituales de combinación de selectores.

```
1 .aviso .especial {
2   /* declaraciones */
3 }
```

El anterior selector solamente selecciona aquellos elementos con un class="especial" que se encuentren dentro de cualquier elemento con un class="aviso".

La combinación de selectores puede llegar a ser todo lo compleja que sea necesario:

```

1  ul#menuPrincipal li.destacado a#inicio {
2      /* declaraciones */
3 }
```

*Imagen 16*

El selector de la *Imagen 16* hace referencia al enlace con un atributo id igual a inicio que se encuentra dentro de un elemento de tipo **<li>** con un atributo class igual a destacado, que forma parte de una lista **<ul>** con un atributo id igual a menuPrincipal.

## Unidades de medida

**Muchas de las propiedades de CSS que se ven en las próximas unidades permiten indicar medidas en sus valores. Además, CSS es tan flexible que permite indicar las medidas de muchas formas diferentes.**

Por este motivo, se presentan a continuación todas las alternativas disponibles en CSS para indicar las medidas.

Las medidas en CSS se emplean, entre otras, para definir la altura, anchura y márgenes de los elementos y para establecer el tamaño de letra del texto. Todas las medidas se indican como un valor numérico entero o decimal seguido de una unidad de medida (sin ningún espacio en blanco entre el número y la unidad de medida).

CSS divide las unidades de medida en dos grupos: **absolutas y relativas**.

*Las medidas relativas definen su valor en relación con otra medida, por lo que para obtener su valor real, se debe realizar alguna operación con el valor indicado.*

*Las unidades absolutas establecen de forma completa el valor de una medida, por lo que su valor real es directamente el valor indicado.*

Si el valor es 0, la unidad de medida es opcional. Si el valor es distinto a 0 y no se indica ninguna unidad, la medida se ignora completamente, lo que suele ser una fuente habitual de errores para los diseñadores que empiezan con CSS.

Algunas propiedades permiten indicar medidas negativas, aunque habitualmente sus valores son positivos.

```

1  body {
2      font-size: 1em;
3  }
4
5  h1 {
6      font-size: 200%;
7  }

```

Imagen 17

## Unidades relativas

Son más flexibles que las unidades absolutas porque se adaptan más fácilmente a los diferentes medios. A continuación se muestra la lista de unidades de medida relativas y la referencia que se toma para determinar su valor real:

**em**, (no confundir con la etiqueta <em> de HTML) relativa respecto del tamaño de letra empleado. Aunque no es una definición exacta, el valor de 1em se puede aproximar por la anchura de la letra M ("eme mayúscula") del tipo y tamaño de letra que se esté utilizando.

**ex**, relativa respecto de la altura de la letra x ("equis minúscula") del tipo y tamaño de letra que se esté utilizando.

**px**, (píxel) relativa respecto de la resolución de la pantalla del usuario.

## Unidades absolutas

Definen las medidas de forma completa, ya que sus valores reales no se calculan a partir de otro valor de referencia, sino que son directamente los valores indicados.

A continuación se muestra la lista completa de unidades absolutas definidas por CSS y su significado:

**in**, del inglés "inches", pulgadas (1 pulgada son 2.54 centímetros)

**cm**, centímetros

**mm**, milímetros

**pt**, puntos (1 punto equivale a 1 pulgada/72, es decir, unos 0.35 milímetros)

**pc**, picas (1 pica equivale a 12 puntos, es decir, unos 4.23 milímetros)

## Porcentajes

CSS define otra unidad de medida relativa basada en los porcentajes. Un porcentaje está formado por un valor numérico seguido del símbolo % y siempre está referenciado a otra medida.

Cada una de las propiedades de CSS que permiten indicar como valor un porcentaje, define el valor al que hace referencia ese porcentaje. Los porcentajes se pueden utilizar por ejemplo para establecer el valor del tamaño de letra de los elementos (ver *Imagen 17, arriba*):

# Colores

Los colores en CSS se pueden indicar de cinco formas diferentes: palabras clave, colores del sistema, RGB hexadecimal, RGB numérico y RGB porcentual.

Los métodos más habituales son las Palabras claves y el RGB hexadecimal, por eso desarrollaremos estas alternativas.

## Palabras clave

CSS define 17 palabras clave para referirse a los colores básicos. Las palabras se corresponden con el nombre en inglés de cada color:

<b>maroon</b> #800000	<b>red</b> #ff0000	<b>orange</b> #ffa500	<b>yellow</b> #ffff00	<b>olive</b> #808000
<b>purple</b> #800080	<b>fuchsia</b> #ff00ff	<b>white</b> #ffffff	<b>lime</b> #00ff00	<b>green</b> #008000
<b>navy</b> #000080	<b>blue</b> #0000ff	<b>aqua</b> #00ffff	<b>teal</b> #008080	
<b>black</b> #000000	<b>silver</b> #c0c0c0	<b>gray</b> #808080		

La imagen anterior ha sido extraída de la sección sobre colores de la especificación oficial de CSS.

Aunque es una forma muy sencilla de referirse a los colores básicos, este método prácticamente no se utiliza en las hojas de estilos de los sitios web reales, ya que se trata de una gama de colores muy limitada.

Además de la lista básica, los navegadores modernos soportan muchos otros nombres de

de colores. La lista completa se puede ver en [https://en.wikipedia.org/wiki/Web\\_colors#Web-safe\\_colors](https://en.wikipedia.org/wiki/Web_colors#Web-safe_colors).

## RGB Hexadecimal

Aunque es el método más complicado para indicar los colores, se trata del método más utilizado con mucha diferencia. De hecho, prácticamente todos los sitios web reales utilizan exclusivamente este método.

Para entender el modelo RGB hexadecimal, en primer lugar es preciso introducir un concepto matemático llamado sistema numérico hexadecimal. Cuando realizamos operaciones matemáticas, siempre utilizamos 10 símbolos para representar los números (del 0 al 9). Por este motivo, se dice que utilizamos un sistema numérico decimal.

No obstante, el sistema decimal es solamente uno de los muchos sistemas numéricos que se han definido. Entre los sistemas numéricos alternativos más utilizados se encuentra el sistema hexadecimal, que utiliza 16 símbolos para representar sus números.

Como sólo conocemos 10 símbolos numéricos, el sistema hexadecimal utiliza también seis letras (de la A a la F) para representar los números. De esta forma, en el sistema hexadecimal, después del 9 no va el 10, sino la A. La letra B equivale al número 11, la C al 12, la D al 13, la E al 14 y la F al número 15.

Definir un color en CSS con el método RGB hexadecimal requiere realizar los siguientes pasos:

- 1 Determinar las componentes RGB decimales del color original, por ejemplo: R = 71, G = 98, B = 176

**2** Transformar el valor decimal de cada componente al sistema numérico hexadecimal. Se trata de una operación exclusivamente matemática, por lo que puedes utilizar una calculadora. En el ejemplo anterior, el valor hexadecimal de cada componente es: R = 47, G = 62, B = B0

**3** Para obtener el color completo en formato RGB hexadecimal, se concatenan los valores hexadecimales de las componentes RGB en ese orden y se les añade el prefijo #. De esta forma, el color del ejemplo anterior es #4762B0 en formato RGB hexadecimal.

Siguiendo el mismo ejemplo de las secciones anteriores, el color del párrafo se indica de la siguiente forma utilizando el formato RGB hexadecimal:

```
2 < p {  
3   |   color: #4762B0;  
4 }
```

Recuerda que aunque es el método más complicado para definir un color, se trata del método que utilizan la inmensa mayoría de sitios web, por lo que es imprescindible dominarlo. Afortunadamente, todos los programas de diseño gráfico convierten de forma automática los valores RGB decimales a sus valores RGB hexadecimales, por lo que no tienes que hacer ninguna operación matemática.

## Colores RGBA

Como ya fuimos viendo en unidades anteriores, los colores en HTML se expresan en valores RGB, igual que en CSS, que admite diversas notaciones para definir el color, a través de números en hexadecimal.

La notación RGBA es una manera de especificar colores en la que se definen cuatro valores. Los tres primeros son los bien conocidos canales RGB (rojo, verde y azul) y el cuarto parámetro es el canal Alpha, que no es más que el grado de transparencia u opacidad del color.

El canal Alpha es un valor entre cero y uno, siendo 0 totalmente transparente y 1 totalmente opaco. Por medio de los colores en RGBA en CSS3, podremos aplicar nuevas transparencias a los colores que especificamos con CSS, abriendo nuevas posibilidades a los diseñadores sin necesidad de complicarse con pequeños trucos como el uso de imágenes de fondo semitransparentes en PNG, etc.

Alpha	hsla(H,S,L,A)	Color
0.0	rgba(255, 0, 0, 0.0)	fully transparent
0.2	rgba(255, 0, 0, 0.2)	
0.4	rgba(255, 0, 0, 0.4)	
0.6	rgba(255, 0, 0, 0.6)	
0.8	rgba(255, 0, 0, 0.8)	
1.0	rgba(255, 0, 0, 1.0)	fully opaque 

Además, como los colores RGBA se pueden aplicar a cualquier elemento que soporte asignación de color, las aplicaciones aumentan todavía más.

**Para definir un color RGBA, se deben especificar cuatro valores, de la siguiente manera:**

```
background-color: rgba(255, 130, 10, 0.5);
```

Los tres primeros valores son números en sistema decimal, que corresponden con los valores de rojo, verde y azul. Siempre tienen que ser números entre 0 y 255.

El cuarto valor es un número entre 0 y 1. Por ejemplo 0 sería totalmente transparente, 1 sería totalmente opaco y 0.5 sería una transparencia al 50%, es decir, mitad opaco mitad transparente.

## Colores HSL y HSLA

CSS3 añade además un nuevo modelo de color conocido como HSL. Estas siglas provienen del inglés Hue, Saturation, Ligthness o lo que es lo mismo, tono, saturación y brillo.



La sintaxis es la siguiente:

```
/* propiedad: hsl(tono, saturación, brillo) */
background-color: hsl(360, 100%, 20%);
```

Teniendo en cuenta que el valor del tono puede tomar valores del 0 al 360 donde:

- 0, sería el rojo.
- 120, sería el verde.
- 240, sería el azul.
- 360, volvería a ser rojo

Además, como en el caso del modelo RGB, al HSL se le puede añadir un canal alpha para definir la transparencia del color.

El resultado en código sería el siguiente, dando resultados análogos a RGBA:

```
/* propiedad: hsla(tono, saturación, brillo, alpha) */
background-color: hsla(300, 130%, 65%, 10%);
```

## Degrados

Siempre que debíamos realizar un fondo degradado para una página web, lo más común era generar una imagen que utilizábamos como patrón, que pudiese repetirse horizontal o verticalmente.

Vamos a ir haciéndolo paso a paso para que puedan ir siguiéndolo desde su editor de html.

Lo primero que faremos será crear el archivo .html que contendrá nuestro degradado.

Dentro de ese archivo crearemos una declaración de estilo para agregar la propiedad del degradado (veamos la siguiente, *imagen 25*).

```
#contenedor {
    width: 800px;
    height: 500px;
    margin: auto;
    background: -moz-linear-gradient(top, #c03939, #2c2c2c);
    background: -webkit-gradient(linear, 0 0, 0 100%, from(#c03939), to(#2c2c2c));
}
```

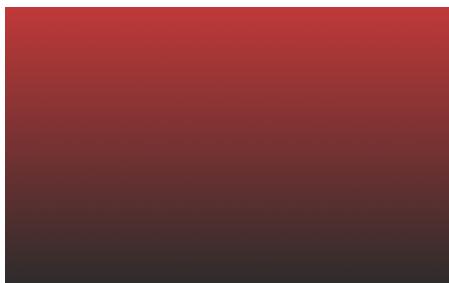
*Imagen 25*

Vamos a ver un poco más en detalle cada sentencia:

```
"background: -moz-linear-gradient(top,
#c03939 , #2c2c2c);"
```

Aunque uno de los objetivos tanto de HTML5 como de CSS3 es estandarizar la visualización en los navegadores, algunos atributos aún requieren una diferenciación de código para cada navegador.

Esta es la sentencia para navegadores Mozilla Firefox, es sencillo reconocerla por su abreviatura inicial **-moz**, el gradiente que acabamos de realizar es el siguiente:



Como podemos ver, es un degradado casi por default, este reparte los colores en un 50% a cada color.

Para modificar esto podemos declarar un porcentaje o tamaño a cada dolor, veamos entonces cómo explicarle al navegador desde donde empieza cada color:

```
background: -moz-linear-gradient(top, #c03939, #2c2c2c 100px);
```

Esta es la primera forma de delimitar los colores, aquí sentenciamos que el segundo color comenzara su llenado después de 100px desde el Top.

Se vería más o menos así:



Otro elemento que podemos controlar es la cantidad de colores que van a intervenir en el degradado:

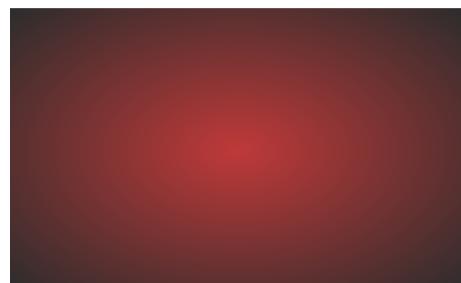
```
background: -moz-linear-gradient(top, #c03939, #2c2c2c 100px, #a2b4c3 80%);
```

```
(top, #c03939, #2c2c2c 100px, #a2b4c3 80%);
```



La propiedad **gradiente** también nos permite trabajar degradados radiales:

```
background: -moz-radial-gradient(50%, #c03939, #2c2c2c);
```



El atributo **WebKit**, es el que podemos visualizar desde Google Chrome, Safari, Microsoft Edge, Opera, Konqueror, Epiphany y Arora.

(Documentación: <https://es.wikipedia.org/wiki/WebKit>).

La sentencia que utilizamos es:

```
background: -webkit-gradient(linear, 0 0, 0
100%, from(#c03939), to(#2c2c2c));
```

Y se interpreta de la siguiente forma:

- **lineal:** indica el tipo de degradado. • 120, sería el verde.
- **O O:** coordenadas de inicio "eje x", "eje y" (O O me indica la esquina superior izquierda).
- **O 100%:** coordenadas de finalización (O 100% indican la esquina inferior derecha).
- **from (#c03939):** color inicial del degradado.
- **to (#2c2c2c):** color que finaliza el degradado.

También podemos agregarle más colores al degradado (al igual que en Firefox), con la propiedad color-stop:

```
background: -webkit-gradient(linear, O O, O 100%, from(#dedede), colorstop(8%, white), color-stop(20%, red));
```

## Sombras

Hasta ahora aplicar un efecto de sombra a cualquier elemento de nuestro html era un proceso en donde teníamos que cargar imágenes creadas previamente en nuestro programa de edición de imágenes favorito, recortarlas, ajustarlas, etc.

Con el atributo box-shadow podemos aplicar sombras a nuestras cajas con facilidad. La sintaxis de box-shadow es la siguiente:

```
box-shadow: distanciaX distanciaY difuminado color;
```

### Desplazamiento horizontal de la sombra

**(distancia X):** La sombra de un elemento suele estar un poco desplazada con respecto al elemento que la produce y su posición será en función del ángulo con el que llegue la luz. Si quisieramos que apareciera un poco hacia la izquierda del elemento original que la produce, pondríamos un valor negativo a este atributo. Cuanto más desplazamiento tenga una sombra, el elemento que la produce parecerá que está más separado del lienzo de la página.

### Desplazamiento vertical de la sombra

**(desplazamiento Y):** El segundo valor que colocamos en el atributo box-shadow es el desplazamiento vertical de la sombra con respecto a la posición del elemento que la produce. Este valor es similar al desplazamiento horizontal.

Valores positivos indican que la sombra aparecerá hacia abajo del elemento y valores negativos harán que la sombra aparezca desplazada un poco hacia arriba.

**Difuminado:** El tercer valor indica cuánto queremos que esté difuminado el borde de la sombra. Si el difuminado fuera cero, querría decir que la sombra no tiene ningún difuminado y aparece totalmente definida. Si el valor es mayor que cero, quiere decir que la sombra tendrá un difuminado de esa anchura.

**Color de la sombra:** El último atributo que se indica en el atributo box-shadow es el color de la sombra. Generalmente las sombras en el mundo real tienen un color negro o grisáceo, pero con CSS3 podemos indicar cualquier gama de color para hacer la sombra, lo que nos dará bastante más versatilidad a los diseños gracias a la posible utilización de sombras en distintos colores, que puedan combinar mejor con nuestra paleta.

Podemos indicar la sombra en hexadecimal o en RGBA tal como vimos anteriormente para el color.

1 Aplicado de la siguiente forma:

```
#sombra {  
    box-shadow: 5px 10px 7px rgba(0,0,0,0.5);  
}
```

El resultado que obtendremos de la aplicación de ese estilo será algo más o menos así (ver imagen 35):

1

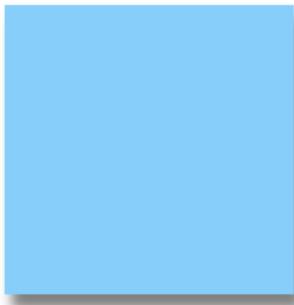


imagen 35

*Hasta ahora aplicar un efecto de sombra a cualquier elemento de nuestro HTML era un proceso en donde teníamos que cargar imágenes creadas previamente en nuestro programa de edición de imágenes (...)*

2

```
box-shadow: 5px 10px 7px □rgba(0, 0, 0, 0.5), -5px -10px 7px □rgba(0, 0, 0, 0.5);
```

imagen 36

2 Si quisiéramos añadir varias sombras, sólo tendríamos que añadir un nuevo valor al atributo usando una coma como separador (ver *imagen 36*):

3 Y el resultado que obtendríamos sería algo así (ver *imagen 37*):

3



imagen 37

## Compatibilidad de las sombras

CSS con navegadores Por el momento podemos utilizar box-shadow en las versiones más moderadas del navegador Opera. Por su parte, navegadores basados en Mozilla y WebKit tienen soporte a esta funcionalidad de CSS3, pero a través de unos atributos CSS con una ligera variación en su nombre.

Atributo box-shadow para navegadores basados en Mozilla, como Firefox: De manera temporal, Firefox es capaz de interpretar el atributo -moz-box-shadow, por ejemplo:

```
-moz-box-shadow: 1px 1px 0px □#090;
```

Atributo box-shadow para navegadores basa-

dos en WebKit, como Safari o Google Chrome: En estos momentos y de manera temporal, navegadores como Chrome o Safari entienden el atributo: -webkit-box-shadow, por ejemplo:

```
-webkit-box-shadow: 3px 3px 1px □#fc8;
```

Como podremos imaginar, si deseamos ampliar al máximo la compatibilidad con box-shadow, necesitaríamos indicar tanto el propio atributo box-shadow (que funciona en Opera y en el futuro funcionará en todos los navegadores), así como -moz-box-shadow y -webkit-box-shadow para que funcione en las versiones actuales de Firefox, Safari, Chrome, etc.

# Texto

**CSS define numerosas propiedades para modificar la apariencia del texto. A pesar de que no dispone de tantas posibilidades como los lenguajes y programas específicos para crear documentos impresos, CSS permite aplicar estilos complejos y muy variados al texto de las páginas web.**

La propiedad básica que define CSS relacionada con la tipografía se denomina color y se utiliza para establecer el color de la letra.

## Atributos

**color:** Establece el color de letra utilizado para el texto Ej: h1 { color: #B1251E; }

**font-family:** Establece el tipo de letra utilizado para el texto. El tipo de letra del texto se puede indicar de dos formas diferentes:

- **Mediante el nombre de una familia tipográfica:** en otras palabras, mediante el nombre del tipo de letra, como por ejemplo "Arial", "Verdana", "Garamond", etc.

- **Mediante el nombre genérico de una familia tipográfica:** los nombres genéricos no se refieren a ninguna fuente en concreto, sino que hacen referencia al estilo del tipo de letra.

**Las familias genéricas definidas son:**

- serif (tipo de letra similar a Times New Roman),
- sans-serif (tipo Arial),
- cursive (tipo Comic Sans),
- fantasy (tipo Impact)
- monospace (tipo Courier New).

Ej: p {font-family: Arial, Helvetica, sans-serif};

**font-size:** Establece el tamaño de letra utilizado para el texto.

Además de todas las unidades de medida re-

lativas y absolutas y el uso de porcentajes, CSS permite utilizar una serie de palabras clave para indicar el tamaño de letra del texto:

- **Tamaño absoluto:** indica el tamaño de letra de forma absoluta mediante alguna de las siguientes palabras clave: xx-small, x-small, small, medium, large, x-large, xx-large.

- **Tamaño relativo:** indica de forma relativa el tamaño de letra del texto mediante dos palabras clave (larger, smaller) que toman como referencia el tamaño de letra del elemento padre.

Ej: p {font-size: medium}

**font-style:** Establece el estilo de la letra utilizada para el texto.

- **Valores:** normal | italic | oblique | inherit.

Ej: h1 {font-style: italic}

## Apariencia

Además de las propiedades relativas a la tipografía del texto, CSS define numerosas propiedades que determinan la apariencia del texto en su conjunto. Estas propiedades adicionales permiten controlar al alineación del texto, el interlineado, la separación entre palabras, etc.

## Atributos

## Atributos

**text-align:** Establece la alineación del contenido del elemento

- \* Valores: left | right | center | justify | inherit

- \* Ej: `p { text-align: center }`

**line-height:** Permite establecer la altura de línea de los elementos

- \* Valores: normal | <numero> | <medida> | <porcentaje> | inherit

- \* Ej: `p { line-height: 120% }`

**text-decoration:** Establece la decoración del texto (subrayado, tachado, parpadeante, etc.)

- Valores: none | ( underline || overline || line-through || blink ) | inherit

**text-transform:** Transforma el texto original (lo transforma a mayúsculas, a minúsculas, etc.)

- \* Valores: capitalize | uppercase | lowercase | none | inherit

- \* Ej: `<div style="text-transform: none">`  
`<h1> Original </h1> Lorem ipsum sit amet...`  
`</div>`

**vertical-align:** Determina la alineación vertical de los contenidos de un elemento

- \* Valores: baseline | sub | super | top | text-top | middle | bottom | text-bottom | inherit | <porcentaje> | <medida>

**letter-spacing:** Permite establecer el espacio entre las letras que forman las palabras del texto

- \* Valores: normal || inherit

**word-spacing:** Permite establecer el espacio entre las palabras que forman el texto

- \* Valores: normal || inherit

*Estas propiedades adicionales permiten controlar al alineación del texto, el interlineado, la separación entre palabras, etc.*

## Box Model (Modelo de Cajas)

**El modelo de cajas o "box model" es seguramente la característica más importante del lenguaje de hojas de estilos CSS, ya que condiciona el diseño de todas las páginas web.**

El modelo de cajas es el comportamiento de CSS que hace que todos los elementos de las páginas se representen mediante cajas rectangulares.

Las cajas de una página se crean automáticamente. Cada vez que se inserta una etiqueta

HTML, se crea una nueva caja rectangular que encierra los contenidos de ese elemento.

La siguiente imagen muestra las tres cajas rectangulares que crean las tres etiquetas HTML que incluye la página (ver *Imagen 40*):

**<strong>**Párrafo de texto con **<strong>**algunas palabras**</strong>** resaltadas**</p>**

**<p>**Otro párrafo**</p>**

Párrafo de texto con **algunas palabras** resaltadas

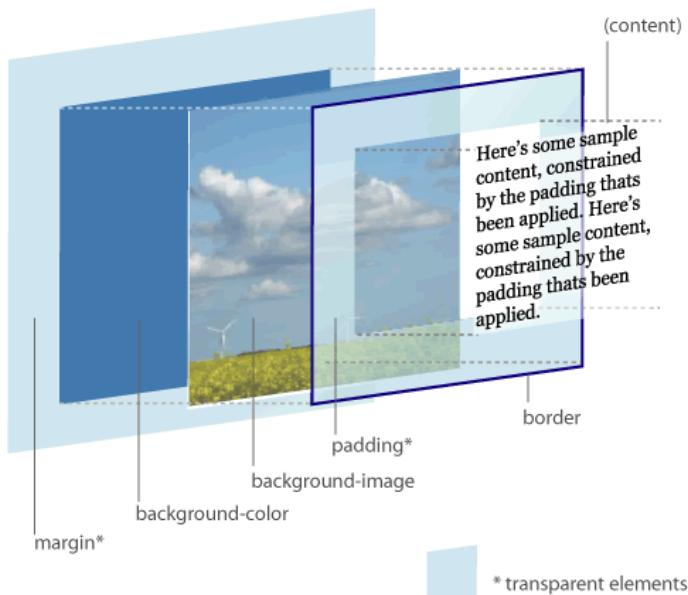
Otro párrafo

Imagen 40

Los navegadores crean y colocan las cajas de forma automática, pero CSS permite modificar todas sus características.

Cada una de las cajas está formada por diferentes partes, tal y como muestra la siguiente imagen:

#### THE CSS BOX MODEL HIERARCHY



Las partes que componen cada caja y su orden de visualización desde el punto de vista del usuario son las siguientes:

**Contenido (content):** se trata del contenido HTML del elemento (las palabras de un párrafo, una imagen, el texto de una lista de elementos, etc.)

**Relleno (padding):** espacio libre opcional exis-

tente entre el contenido y el borde.

**Borde (border):** línea que encierra completamente el contenido y su relleno.

**Imagen de fondo (background image):** imagen que se muestra por detrás del contenido y el espacio de relleno.

**Color de fondo (background color):** color que se muestra por detrás del contenido y el espacio de relleno.

**Margen (margin):** separación opcional existente entre la caja y el resto de cajas adyacentes.

El relleno y el margen son transparentes, por lo que en el espacio ocupado por el relleno o el margen se muestra el color o imagen de fondo (si están definidos).

Si una caja define tanto un color como una imagen de fondo, la imagen tiene más prioridad y es la que se visualiza. No obstante, si la imagen de fondo no cubre totalmente la caja del elemento o si la imagen tiene zonas transparentes, también se visualiza el color de fondo. Combinando imágenes transparentes y colores de fondo se pueden lograr efectos gráficos muy interesantes.

#### Anchura

La propiedad CSS que controla la anchura de los elementos se denomina **width**.

La propiedad **width** no admite valores negativos y los valores en porcentaje se calculan a

partir de la anchura de su elemento padre, es decir, del elemento que lo contiene.

El valor ***inherit*** indica que la anchura del elemento se hereda de su elemento padre.

El valor ***auto***, que es el que se utiliza si no se establece de forma explícita un valor a esta propiedad, indica que el navegador debe calcular automáticamente la anchura del elemento, teniendo en cuenta sus contenidos y el sitio disponible en la página.

## Altura

La propiedad CSS que controla la altura de los elementos se denomina ***height***.

Al igual que sucede con ***width***, la propiedad ***height*** no admite valores negativos.

Si se indica un porcentaje, se toma como referencia la altura del elemento padre. Si el elemento padre no tiene una altura definida explícitamente, se asigna el valor ***auto*** a la altura.

El valor ***inherit*** indica que la altura del elemento se hereda de su elemento padre.

El valor ***auto***, que es el que se utiliza si no se establece de forma explícita un valor a esta propiedad, indica que el navegador debe calcular automáticamente la altura del elemento, teniendo en cuenta sus contenidos y el sitio disponible en la página.

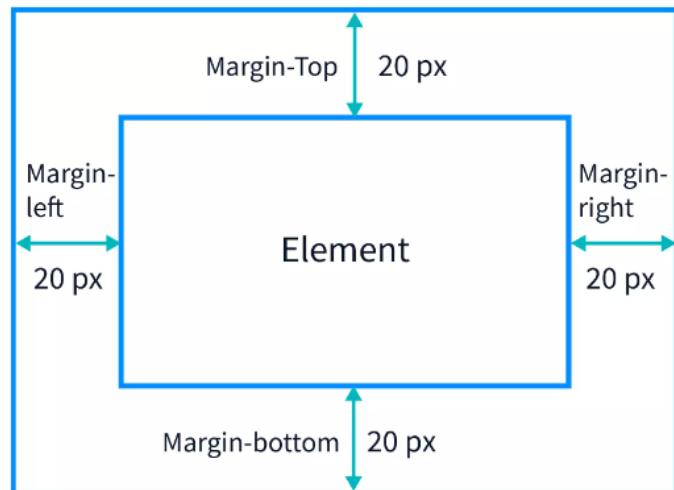
## Margen

CSS define cuatro propiedades para controlar cada uno de los márgenes horizontales y verticales de un elemento.

- ***margin-top***
- ***margin-right***
- ***margin-bottom***
- ***margin-left***

Cada una de las propiedades establece la separación entre el borde lateral de la caja y el

resto de cajas adyacentes:



Las unidades más utilizadas para indicar los márgenes de un elemento son los **píxeles** (cuando se requiere una precisión total), los **em** (para hacer diseños que mantengan las proporciones) y los **porcentajes** (para hacer diseños líquidos o fluidos).

Los márgenes verticales (**margin-top** y **margin-bottom**) sólo se pueden aplicar a los elementos de bloque y las imágenes, mientras que los márgenes laterales (**margin-left** y **margin-right**) se pueden aplicar a cualquier elemento.

Además de las cuatro propiedades que controlan cada uno de los márgenes del elemento, CSS define una propiedad que permite establecer los cuatro márgenes de forma directa empleando una única propiedad.

**La propiedad *margin* admite entre uno y cuatro valores, con el siguiente significado:**

- *Si solo se indica un valor, todos los márgenes tienen ese valor.*
- *Si se indican dos valores, el primero se asigna al margen superior e inferior y el segundo se asigna a los márgenes izquierdo y derecho.*

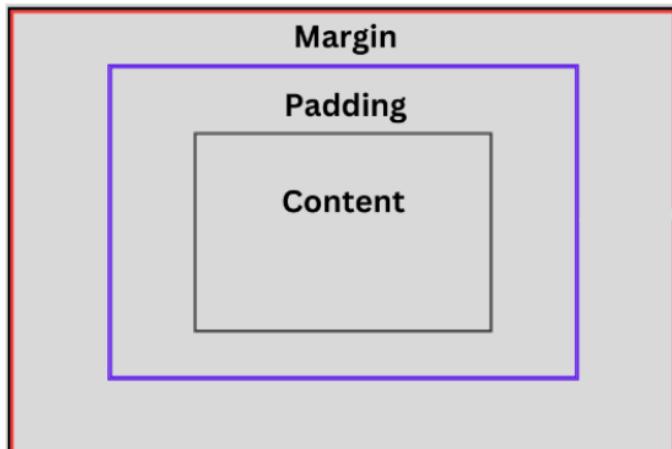
- Si se indican tres valores, el primero se asigna al margen superior, el tercero se asigna al margen inferior y el segundo valor se asigna los márgenes izquierdo y derecho.
- Si se indican los cuatro valores, el orden de asignación es: margen superior, margen derecho, margen inferior y margen izquierdo.

## Relleno

CSS definen cuatro propiedades para controlar cada uno de los espacios de relleno horizontal y vertical de un elemento.

- *padding-top*
- *padding-right*
- *padding-bottom*
- *padding-left*

Cada una de las propiedades establece la separación entre el lateral de los contenidos y el borde lateral de la caja:



La propiedad que permite definir de forma simultánea los cuatro rellenos se denomina **padding**.

La propiedad padding admite entre uno y cuatro valores, con el mismo significado que el de la propiedad **margin**.

## Bordes

CSS permite definir el aspecto de cada uno de los cuatro bordes horizontales y verticales de los elementos. Para cada borde se puede establecer su anchura, su color y su estilo.

### Anchura

La anchura de los bordes se controla con las cuatro propiedades siguientes:

- *border-top-width*
- *border-right-width*
- *border-bottom-width*
- *border-left-width*

La anchura de los bordes se puede indicar mediante una medida (absoluta o relativa y en cualquier unidad de medida de las definidas) o mediante las palabras clave **thin** (borde delgado), **medium** (borde normal) y **thick** (borde ancho).

### Color

El color de los bordes se controla con las cuatro propiedades siguientes:

- *border-top-color*
- *border-right-color*
- *border-bottom-color*
- *border-left-color*

### Estilo

Por último, CSS permite establecer el estilo de cada uno de los bordes mediante las siguientes propiedades:

- *border-top-style*
- *border-right-style*
- *border-bottom-style*
- *border-left-style*

El estilo de los bordes sólo se puede indicar mediante alguna de las palabras reservadas definidas por CSS.

Como el valor por defecto de esta propiedad es *none*, los elementos no muestran ningún borde visible a menos que se establezca explícitamente un estilo de borde.

Los bordes más utilizados en los diseños habituales son *solid* y *dashed*, seguidos de *double* y *dotted*.

La propiedad *borde* admite los 3 valores en la misma regla:

```
div { border: 5px dotted #FFF }
```

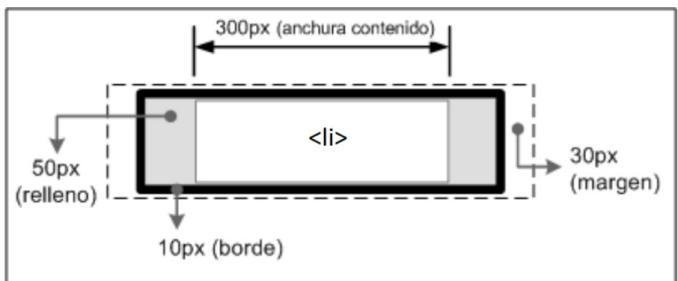
## Margen, relleno, bordes y modelo de cajas

La anchura y altura de un elemento no solamente se calculan teniendo en cuenta sus propiedades *width* y *height*. El margen, el relleno y los bordes establecidos a un elemento determinan la anchura y altura final del elemento.

En el siguiente ejemplo se muestran los estilos CSS de un elemento:

```
li {
  width: 200px;
  padding-left: 50px;
  padding-right: 50px;
  margin-left: 30px;
  margin-right: 30px;
  border: 10px solid black;
}
```

La anchura total con la que se muestra el elemento no son los 300 píxel indicados en la propiedad *width*, sino que se tienen en cuenta todos sus márgenes, rellenos y bordes:



De esta forma, la anchura del elemento en pantalla sería igual a la suma de la anchura original, los márgenes, los bordes y los rellenos:

$$30\text{px} + 10\text{px} + 50\text{px} + 300\text{px} + 50\text{px} + 10\text{px} + 30\text{px} = 480 \text{ píxel}$$

Así, la anchura/altura establecida con CSS siempre hace referencia a la anchura/altura del contenido.

La anchura/altura total del elemento debe tener en cuenta además los valores del resto de partes que componen la caja del box model.

*La anchura y altura de un elemento no solamente se calculan teniendo en cuenta sus propiedades width y height. El margen, el relleno y los bordes establecidos a un elemento determinan la anchura y altura final del elemento.*

## Fondos

El último elemento que forma el box model es el fondo de la caja del elemento. El fondo puede ser un color simple o una imagen. El fondo solamente se visualiza en el área ocupada por el contenido y su relleno, ya que el color de los bordes se controla directamente desde los bordes y las zonas de los márgenes siempre son transparentes.

Para establecer un color o imagen de fondo en la página entera, se debe establecer un fondo al elemento `<body>`. Si se establece un fondo a la página, como el valor inicial del fondo de los elementos es transparente, todos los elementos de la página se visualizan con el mismo fondo a menos que algún elemento especifique su propio fondo.

CSS define cinco propiedades para establecer el fondo de cada elemento (`background-color`, `background-image`, `background-repeat`, `background-attachment`, `background-position`).

En ocasiones, es necesario crear un fondo más complejo que un simple color.

CSS permite mostrar una imagen como fondo de cualquier elemento:

**background-image:** Establece una imagen como fondo para los elementos

\* Valores: `<url>` | `none` | `inherit`

CSS permite establecer de forma simultánea un color y una imagen de fondo. En este caso, la imagen se muestra delante del color, por lo que solamente si la imagen contiene zonas transparentes es posible ver el color de fondo.

Las imágenes de fondo se indican a través de su URL, que puede ser absoluta o relativa. Suele ser recomendable crear una carpeta de imágenes que se encuentre en el mismo directorio que los archivos CSS y que almacene todas las imágenes utilizadas en el diseño de las páginas.

Así, las imágenes correspondientes al diseño de la página se mantienen separadas del resto de imágenes del sitio y el código CSS es más sencillo (por utilizar URL relativas) y más fácil de mantener (por no tener que actualizar URL absolutas en caso de que se cambie la estructura del sitio web).

CSS introduce la propiedad `background-repeat` que permite controlar la forma de repetición de las imágenes de fondo.

## Flexbox

***El propósito de Flexbox es mejorar la eficiencia en la creación de diseños, permitiendo el alineamiento y distribución de espacios entre elementos en un contenedor de manera más eficiente, incluso cuando las dimensiones de estos elementos son desconocidas o dinámicas debido al concepto de "flex".***

Exploraremos los principios fundamentales de CSS Flexbox para el alineamiento y posicionamiento, así como el correcto uso de sus funcionalidades.

## Introducción a Flexbox

Durante mucho tiempo, las únicas herramientas disponibles para crear diseños CSS y posicionar elementos de manera compatible con varios navegadores eran `float` y `position`.

Sin embargo, estas herramientas presentaban limitaciones frustrantes, especialmente en términos de responsividad.

Tareas básicas en diseño, como centrar verticalmente un elemento secundario respecto a uno principal, o hacer que los elementos secundarios ocupen el mismo espacio o garantizar que las columnas tengan el mismo tamaño sin importar el contenido interno, eran difíciles o incluso imposibles de manejar de manera práctica y flexible con `float` o `position`.

Flexbox fue desarrollada para simplificar y me-

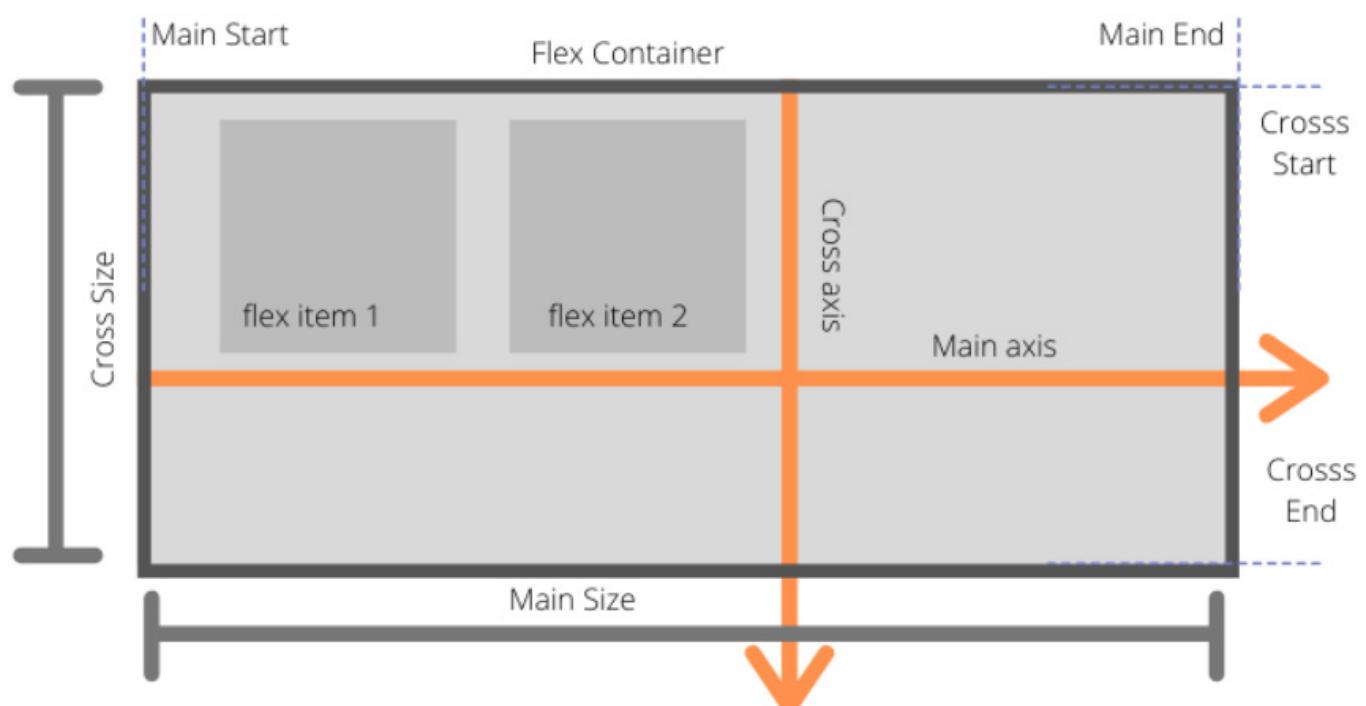
jorar estas tareas: permite posicionar los elementos secundarios en cualquier dirección y ajustar sus dimensiones de forma flexible para adaptarse a diferentes situaciones.

## Los componentes

Flexbox no se limita a una sola propiedad; es un módulo completo que requiere ciertas declaraciones tanto en el contenedor (llamado **flex container**) como en los elementos secundarios (**los flex items**).

Mientras que el diseño convencional utiliza direcciones `block` e `inline`, el diseño Flex se apoya en direcciones de “`flex flow`”.

Para comprender mejor este concepto, se presenta un diagrama de la especificación que ilustra la idea central detrás del diseño Flex.



**Los ítems se distribuirán en el diseño siguiendo el eje principal(Main axis) o transversal (Cross axis):**

**EJE PRINCIPAL:**

el eje principal de un flex container es el eje primario y a lo largo de él son insertados los flex ítems. Precaución: El eje principal no es necesariamente horizontal; Dependerá de la propiedad flex-direction (se verá más abajo).

**EJE TRANSVERSAL:**

El eje perpendicular al eje principal se llama eje transversal. Su dirección depende de la dirección del eje principal.

**MAIN-START | MAIN-END:**

los flex ítems se insertan en el contenedor empezando por el lado start, dirigiéndose hacia el lado end.

**CROSS-START | CROSS-END:**

Líneas flex se llenan con ítems y se agregan al contenedor, comenzando desde el lado cross start del flex container hacia el lado cross end.

**TAMAÑO PRINCIPAL:**

El ancho o alto de un flex ítem, dependiendo de la dirección del contenedor, es el tamaño principal del ítem. La propiedad de tamaño principal de un flex ítem puede ser tanto width como height, dependiendo de cuál esté en la dirección principal.

**CROSS SIZE:**

El ancho o alto de un flex ítem, dependiendo de lo que haya en la dimensión transversal, es el cross size del ítem. La propiedad cross size puede ser el ancho o el alto del ítem, lo que se encuentre en la transversal.

## Contenedor flex

Un contenedor flex es un elemento HTML que actúa como un contenedor para los elementos flex-items. Es el elemento padre que controla cómo se disponen y se comportan sus hijos flex-items.

Para crear un contenedor flex, se utiliza la propiedad CSS display con el valor flex o inline-flex. Aquí hay un ejemplo:

```
<body>
  <div class="container">
    <div class="item">Item 1</div>
    <div class="item">Item 2</div>
    <div class="item">Item 3</div>
  </div>
</body>

.container {
  display: flex;
  /* Otras propiedades de estilo para el contenedor */
}

.item {
  /* Estilos para los elementos flex-items */
}
```

En este ejemplo, la clase .container se convierte en un contenedor flex al aplicar display: flex;. Todos los elementos hijos directos (.item) de este contenedor se convierten automáticamente en flex-items.

display: flex; crea un contenedor flex de bloque, que ocupa todo el ancho disponible y puede crecer en altura según sea necesario. Una vez que se establece un contenedor flex, se pueden aplicar otras propiedades CSS para controlar el comportamiento y la disposición de los flex-items dentro de él. Por ejemplo, flex-direction, justify-content, align-items, etc.

Es importante tener en cuenta que solo los elementos hijos directos del contenedor flex se convierten en flex-items. Los descendientes anidados de un flex-item no se ven afectados por las propiedades de Flexbox aplicadas al contenedor flex, a menos que esos descendientes también se conviertan en contenedores flex.

En conclusión, podemos decir que el contenedor flex es el elemento clave en Flexbox que permite distribuir y alinear sus elementos hijos (flex-items) de manera flexible y adaptable. Esto lo hace ideal para crear diseños responsivos y adaptables para diferentes tamaños de pantalla y dispositivos.

## Flex-items

Los flex-items son los elementos hijos directos de un contenedor flex. Son los elementos que se distribuyen y alinean dentro del contenedor flex según las propiedades establecidas en él. Los flex-items solo pueden ser hijos directos del contenedor flex. Si un elemento está anidado dentro de otro elemento que es un flex-item, ese elemento anidado ya no se considera

un flex-item a menos que se convierta en un contenedor flex por sí mismo.

Es importante destacar que los flex-items no solo se ven afectados por las propiedades de Flexbox aplicadas al contenedor flex en el que están contenidos directamente, sino también por las propiedades de Flexbox que se les apliquen individualmente.

### Ejemplo:

```
<body>
  <div class="container">
    <div class="item">Item 1</div>
    <div class="item">
      <div class="sub-item">Sub-item 1</div>
      <div class="sub-item">Sub-item 2</div>
    </div>
    <div class="item">Item 3</div>
  </div>
</body>
```

```
.container {
  display: flex;
}

.item {
  background-color: #f1f1f1;
  padding: 10px;
  margin: 5px;
  flex-grow: 1; /* Permite que los flex-items crezcan y ocupen el espacio disponible
}

.sub-item {
  background-color: #ddd;
  padding: 5px;
}
```

En este ejemplo, solo los elementos .item son flex-items, mientras que los .sub-item no se ven afectados por las propiedades de Flexbox del contenedor .container. Sin embargo, los .item sí se ven afectados por la propiedad **flex-grow: 1;** aplicada individualmente.

Item 1

Sub-item 1

Sub-item 2

Item 3

## Ejes principales y secundarios

Flexbox introduce dos ejes principales: el eje principal (main axis) y el eje secundario (cross axis). Estos ejes son fundamentales para comprender cómo se distribuyen y alinean los flex-items dentro del contenedor flex.

**Eje principal (main axis):** Es el eje a lo largo del cual se distribuyen los flex-items en el orden de flujo definido por la propiedad `flex-direction`.

Por defecto, `flex-direction` es `row`, lo que significa que el eje principal es horizontal y los flex-items se distribuyen de izquierda a derecha.

**Eje secundario (cross axis):** Es el eje perpendicular al eje principal, donde se alinean los flex-items de acuerdo con propiedades como `align-items` y `align-self`.

La dirección del eje principal se establece con la propiedad `flex-direction` aplicada al contenedor flex.

Esta propiedad puede tomar cuatro valores:

**row (valor predeterminado):** Los flex-items se distribuyen en una línea horizontal, de izquierda a derecha.

**column:** Los flex-items se distribuyen en una columna vertical, de arriba hacia abajo.

**row-reverse:** Los flex-items se distribuyen en una línea horizontal, pero en orden inverso, de derecha a izquierda.

**column-reverse:** Los flex-items se distribuyen en una columna vertical, pero en orden inverso, de abajo hacia arriba.

Ejemplo:

```
.container {
  display: flex;
  flex-direction: row; /* Eje principal horizontal */
}
```

En este ejemplo, el eje principal es horizontal, y los flex-items se distribuyen de izquierda a derecha. El eje secundario es vertical.

Comprender estos ejes es fundamental para entender cómo funcionan las propiedades de alineación y distribución de Flexbox.

## Justificar contenido en el eje principal

La propiedad CSS `justify-content` se aplica al contenedor flex y controla cómo se alinean los flex-items a lo largo del eje principal.

Esta propiedad determina cómo se distribuye el espacio disponible entre los flex-items y sus extremos.

Los valores posibles para `justify-content` son:

**Flex-start (valor predeterminado):** Los flex-items se alinean al inicio del eje principal.

**Flex-end:** Los flex-items se alinean al final del eje principal.

**Center:** Los flex-items se centran en el eje principal.

**Space-between:** Los flex-items se distribuyen uniformemente a lo largo del eje principal, con el primer flex-item al inicio y el último al final.

**Space-around:** Los flex-items se distribuyen uniformemente a lo largo del eje principal, con un espacio igual entre ellos y en los extremos.

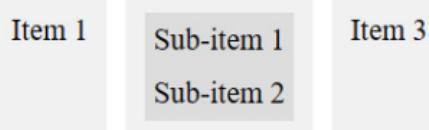
**Space-evenly:** Los flex-items se distribuyen uniformemente a lo largo del eje principal, con un espacio igual entre ellos y en los extremos.

Ejemplo:

```
.container {
  display: flex;
  justify-content: center; /* Centra los flex-items en el eje principal */
}
```

En este ejemplo, los flex-items se centrarán horizontalmente dentro del contenedor flex. Es importante tener en cuenta que `justify-content` solo afecta la alineación de los flex-items en el eje principal.

Para controlar la alineación en el eje secundario, se utiliza la propiedad `align-items` o `align-self`.



## Alinear contenido en el eje secundario

La propiedad CSS align-items se aplica al contenedor flex y controla cómo se alinean los flex-items a lo largo del eje secundario (perpendicular al eje principal). Esta propiedad determina cómo se distribuye el espacio disponible entre los flex-items y los extremos del eje secundario.

Los valores posibles para align-items son:

**Stretch (valor predeterminado):** Los flex-items se estiran para ocupar todo el alto disponible en el eje secundario.

**Flex-start:** Los flex-items se alinean al inicio del eje secundario.

**Flex-end:** Los flex-items se alinean al final del eje secundario.

**Center:** Los flex-items se centran en el eje secundario.

**Baseline:** Los flex-items se alinean en la línea base de su contenido de texto.

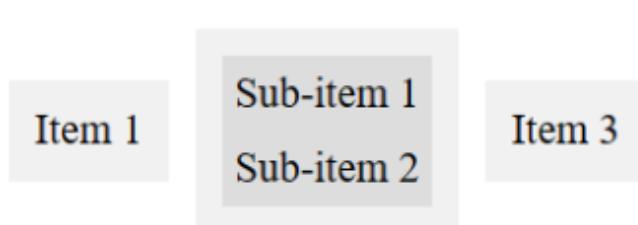
Ejemplo:

```
.container {
  display: flex;
  justify-content: center; /* Centra los flex-items en el eje principal */
  align-items: center; /* Centra los flex-items en el eje secundario */
}
```

En este ejemplo, los flex-items se centrarán verticalmente dentro del contenedor flex y se centrarán horizontalmente por el justify-content.

Es importante tener en cuenta que align-items solo afecta la alineación de los flex-items en el eje secundario (vertical en este caso).

*Además, la propiedad align-self se puede aplicar a flex-items individuales para anular la alineación predeterminada establecida por align-items en el contenedor flex.*



## Ordenar flex-items

La propiedad CSS `order` se aplica a los flex-items individuales y permite cambiar su orden visual sin modificar el orden en el HTML. Esta propiedad es útil cuando se desea reordenar los elementos basados en ciertos criterios, como el diseño responsive, sin tener que cambiar la estructura HTML subyacente.

La propiedad `order` acepta valores numéricos enteros. Los valores más bajos aparecen primero en el orden visual. El valor predeterminado es 0.

Ejemplo:

```
<body>
  <div class="container">
    <div class="item_uno">Item 1</div>
    <div class="item_dos">Item 2</div>
    <div class="item_tres">Item 3</div>
  </div>
</body>
```

```
.container {
  display: flex;
}

.item_uno {
  order: 2;
}

.item_dos {
  order: 3;
}

.item_tres {
  order: 1;
}
```

*En este ejemplo, el orden visual de los flex-items será: Item 3, Item 1, Item 2, aunque en el HTML aparecen en un orden diferente.*

Es importante tener en cuenta que `order` solo cambia el orden visual de los flex-items dentro del contenedor flex. No afecta el orden del contenido en el DOM ni el orden de tabulación para la accesibilidad.

Además, **order no debe utilizarse como un reemplazo de un correcto orden semántico en el HTML**.

## Flexibilidad de los flex-items

Flexbox proporciona tres propiedades CSS que controlan la flexibilidad y el tamaño de los flex-items: **flex-grow**, **flex-shrink** y **flex-basis**. Estas propiedades se aplican individualmente a cada flex-item y determinan cómo se comportan cuando hay espacio disponible o cuando hay restricciones de tamaño en el contenedor flex.

**flex-grow:** Define la capacidad de un flex-item para crecer y ocupar el espacio disponible en el contenedor flex.

El valor predeterminado es 0 (no crece). Un valor mayor que 0 indica que el flex-item puede crecer y ocupar el espacio restante en proporción a los demás flex-items que también tienen un valor `flex-grow` mayor que 0.

**flex-shrink:** Define la capacidad de un flex-item para encogerse cuando hay poco espacio disponible en el contenedor flex.

El valor predeterminado es 1 (encoge proporcionalmente). Un valor mayor que 0 indica que el flex-item puede encogerse en proporción a los demás flex-items que también tienen un valor `flex-shrink` mayor que 0. Un valor de 0 indica que el flex-item no se encoge.

**flex-basis:** Establece el tamaño base de un flex-item antes de que se distribuya el espacio restante según `flex-grow` y `flex-shrink`. Puede ser una longitud (200px) o un porcentaje (50%). El valor predeterminado es auto, lo que significa que el tamaño base se calcula en función del contenido del elemento.

Estas tres propiedades se pueden abreviar utilizando la propiedad **flex**.

Por ejemplo:

```
.item {
  flex: 1 1 200px; /* flex: grow shrink basis */
}
```

*En el ejemplo de la imagen 58, cada flex-item tiene una capacidad de crecimiento y encogimiento igual (1), y un tamaño base de 200px.*

*imagen 58*

Es importante tener en cuenta que **flex-grow** y **flex-shrink** solo tienen efecto cuando hay espacio disponible o restricciones de tamaño en el contenedor **flex**, respectivamente.

Si todos los **flex-items** tienen los mismos valores para estas propiedades, se distribuirán de manera proporcional. Si solo uno de ellos tiene un valor diferente, ese **flex-item** tendrá prioridad para crecer o encogerse según corresponda.

## Envolver flex-items

La propiedad CSS **flex-wrap** se aplica al contenedor **flex** y controla si los **flex-items** deben ajustarse en una sola línea (**nowrap**) o pueden

envolverse en múltiples líneas (**wrap** o **wrap-reverse**) dentro del contenedor.

**nowrap (valor predeterminado):** Los **flex-items** se ajustan en una sola línea, incluso si el contenedor **flex** no tiene suficiente espacio para acomodarlos.

**wrap:** Los **flex-items** se envuelven en múltiples líneas si el contenedor **flex** no tiene suficiente espacio para acomodarlos en una sola línea.

**wrap-reverse:** Los **flex-items** se envuelven en múltiples líneas, pero en orden inverso (de abajo hacia arriba).

Ejemplo:

```
.container {
  display: flex;
  flex-wrap: wrap; /* Permite que los flex-items se envuelvan en múltiples líneas */
}
```

*En este ejemplo, si los flex-items no caben en una sola línea dentro del contenedor flex, se envuelven en múltiples líneas para adaptarse al espacio disponible.*

*La propiedad **flex-wrap** es especialmente útil para crear diseños responsivos que se adapten a diferentes tamaños de pantalla. Al combinarla con propiedades como **flex-grow** y **flex-shrink**, se pueden crear diseños que se ajusten de manera inteligente a diferentes dispositivos y resoluciones.*

## Alineación de líneas flex

Cuando hay varias líneas de flex-items debido a la propiedad `flex-wrap`, la propiedad CSS `align-content` aplicada al contenedor flex controla cómo se alinean esas líneas dentro del contenedor.

Los valores posibles para `align-content` son:

**Stretch (valor predeterminado):** Las líneas se estiran para ocupar todo el alto disponible en el contenedor flex.

**Flex-start:** Las líneas se alinean al inicio del contenedor flex en el eje secundario.

**Flex-end:** Las líneas se alinean al final del contenedor flex en el eje secundario.

**Center:** Las líneas se centran en el eje secundario dentro del contenedor flex.

**Space-between:** Las líneas se distribuyen uniformemente a lo largo del eje secundario, con la primera línea al inicio y la última al final.

**Space-around:** Las líneas se distribuyen uniformemente a lo largo del eje secundario, con un espacio igual entre ellas y en los extremos.

**Space-evenly:** Las líneas se distribuyen uniformemente a lo largo del eje secundario, con un espacio igual entre ellas y en los extremos.

Ejemplo:

```
.container {
  display: flex;
  flex-wrap: wrap;
  align-content: space-between; /* Distribuye las líneas de flex-items uniformemente */
  height: 500px; /* Altura fija para mostrar el efecto */
}
```

*En este ejemplo, si hay varias líneas de flex-items debido a `flex-wrap: wrap`, esas líneas se distribuirán uniformemente en el eje secundario (vertical), con la primera línea al inicio y la última al final, gracias a `align-content: space-between`. Además, se establece una altura fija de 500px para mostrar claramente el efecto.*

## Alineación individual de flex-items

Las propiedades `align-self` y `justify-self` permiten anular la alineación predeterminada de un flex-item individual en el eje secundario y principal, respectivamente.

ver página si-  
guiente

**Align-self** controla la alineación de un flex-item individual en el eje secundario, anulando el valor de align-items establecido en el contenedor flex.

Sus valores son auto (sigue la regla de align-items), flex-start, flex-end, center, baseline y stretch.

**Justify-self** controla la alineación de un flex-item individual en el eje principal, anulando el valor de justify-content establecido en el contenedor flex.

Sus valores son auto (sigue la regla de justify-content), flex-start, flex-end, center y stretch.

Ejemplo:

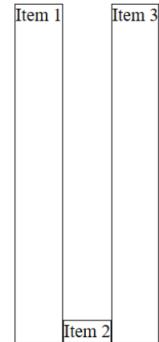
```
<body>
  <div class="container">
    <div class="item">Item 1</div>
    <div class="item__dos">Item 2</div>
    <div class="item">Item 3</div>
  </div>
</body>
```

```
.container {
  display: flex;
  height: 300px; /* Altura para mostrar el efecto de align-self */
}

.item {
  border: 1px solid black;
}

.item__dos {
  border: 1px solid black;
  align-self: flex-end;
}
```

En este ejemplo, el "Item\_\_dos" se alinea al final del eje secundario (abajo) debido a **align-self: flex-end;**, mientras que los otros flex-items siguen la alineación predefinida establecida por align-items en el contenedor. Los items tienen un border de color negro de 1píxel para poder mejorar la exemplificación visual:



Flexbox ha transformado la forma en que los desarrolladores web abordan el diseño de interfaces. Su versatilidad, facilidad de uso y capacidades de adaptación lo convierten en una herramienta indispensable para crear experiencias web modernas, atractivas y accesibles en una amplia gama de dispositivos y plataformas.

A medida que el diseño web continúa evolucionando, Flexbox seguirá siendo una pieza fundamental en el conjunto de herramientas de los desarrolladores web de vanguardia.

¿Vamos a practicar?

<https://flexboxfroggy.com/#es>

## Enlaces útiles

Al igual que con cualquier otra funcionalidad nueva que hemos aprendido, es fundamental practicar mucho e investigar siempre que tenemos dudas. A continuación se muestran algunos enlaces útiles.

[https://developer.mozilla.org/es/docs/Learn/CSS/CSS\\_layout/Flexbox](https://developer.mozilla.org/es/docs/Learn/CSS/CSS_layout/Flexbox)

[https://www.w3schools.com/css/css3\\_flexbox.asp](https://www.w3schools.com/css/css3_flexbox.asp)

## Grid

**Grid Layout es un sistema de diseño de dos dimensiones (filas y columnas) introducido en CSS3 que permite a los desarrolladores web crear diseños complejos de una manera más sencilla y eficiente que los métodos tradicionales.**

Antes de Grid, los diseñadores web tenían que depender de técnicas complejas como tablas o posicionamiento flotante para lograr diseños de múltiples columnas, lo cual a menudo resultaba engorroso y difícil de mantener.

Grid simplifica el proceso de crear diseños basados en cuadrículas, lo que permite a los desarrolladores controlar el tamaño, posición y capas de los elementos de manera precisa.

Al utilizar Grid, los desarrolladores pueden dividir una página en regiones o áreas principales, definir el comportamiento de los elementos dentro de esas áreas y controlar cómo se distribuye el espacio entre ellos.

## Diferencias entre Grid y Flexbox

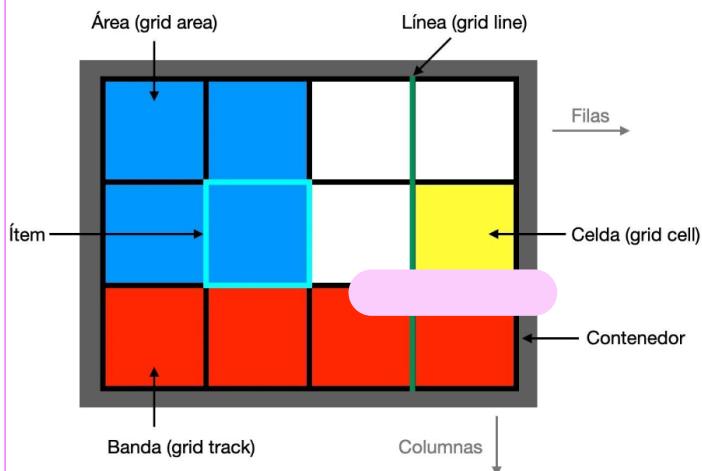
Tanto Grid como Flexbox son herramientas poderosas para crear diseños en CSS, pero se diferencian en su enfoque y capacidades.

GRID	FLEXBOX
<p>Es un sistema de diseño bidimensional que permite organizar elementos en filas y columnas simultáneamente. Es ideal para crear diseños complejos con múltiples elementos dispuestos en una cuadrícula.</p>	<p>Es un módulo de diseño unidimensional que permite organizar elementos en una sola dirección, ya sea en filas o columnas. Es excelente para crear diseños flexibles y adaptables, como menús de navegación, barras de herramientas y contenedores con un diseño lineal.</p>

*Antes de comenzar con Grid, es conveniente conocer el sistema de maquetación Flexbox, ya que Grid toma la filosofía (y muchas de las bases y conceptos) que se utilizan en él.*

## Conceptos fundamentales.

Para crear diseños basados en Grid CSS necesitaremos tener en cuenta una serie de conceptos que utilizaremos a partir de ahora y que definiremos a continuación:



**Contenedor:** El elemento padre contenedor que definirá la cuadrícula o rejilla.

**Ítem:** Cada uno de los hijos que contiene la cuadrícula (elemento contenedor).

**Celda (grid cell):** Cada uno de los cuadritos (unidad mínima) de la cuadrícula.

**Área (grid area):** Región o conjunto de celdas de la cuadrícula.

**Banda (grid track):** Banda horizontal o vertical de celdas de la cuadrícula.

**Línea (grid line):** Separador horizontal o vertical de las celdas de la cuadrícula.

## Definiendo un Grid Container

Para empezar a utilizar CSS Grid, primero debemos definir un elemento como Grid Container. Esto se logra aplicando la propiedad `display: grid` o `display: inline-grid` al elemento en cuestión.

```
<div class="container">
  <!-- Aquí van los Grid Items -->
</div>
```

```
.container {
  display: grid;
}
```

Una vez que tenemos un Grid Container, todos sus hijos directos se convierten automáticamente en Grid Items.

## Grid Tracks: Filas y Columnas

Las filas y columnas del Grid se denominan "Grid Tracks". Para definir el tamaño de estas filas y columnas, utilizamos las propiedades `grid-template-columns` y `grid-template-rows`.

### Grid-template-columns

Esta propiedad define el ancho de las columnas en el Grid Container.

Podemos especificar un tamaño fijo (px, %, etc.) o utilizar la unidad fraccionaria `fr` para definir tamaños proporcionales (ver imágenes siguientes, 67 y 68).

```
<div class="container">
  <div class="item1">Item 1</div>
  <div class="item2">Item 2</div>
  <div class="item3">Item 3</div>
</div>
```

imagen 67

```
.container {
  display: grid;
  grid-template-columns: 200px 1fr 2fr;
  /* Una columna de 200px, otra de 1fr y otra de 2fr */
}
```

Imagen 68

En este ejemplo, la primera columna tendrá un ancho de 200px, la segunda ocupará 1 porción de espacio disponible, y la tercera ocupará 2 porciones de espacio disponible.

### Grid-template-rows

De manera similar, grid-template-rows define la altura de las filas del Grid Container.

```
.container {
  display: grid;
  grid-template-rows: 100px 200px auto;
  /* Una fila de 100px, otra de 200px y otra que ocupa el resto */
}
```

Imagen 69

En este caso, la primera fila tendrá una altura de 100px, la segunda de 200px, y la tercera ocupará el espacio restante (auto).

### Ejemplo combinado (imágenes 70 y 71)

Aclaraciones a tener en cuenta:

1. Se le dió un borde de color a negro a los items para poder visualizar mejor cada uno de los items.
2. Se agregaron 3 items más para el ejemplo.

```
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
  <div class="item">Item 4</div>
  <div class="item">Item 5</div>
  <div class="item">Item 6</div>
</div>
```

```
.container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  grid-template-rows: 100px 200px;
}

.item {
  border: 1px solid black;
```

Imagen 71

Dando esta grilla como resultado:

Item 1	Item 2	Item 3
Item 4	Item 5	Item 6

Imagen 72

Aquí, estamos definiendo un Grid Container con tres columnas (la del medio el doble de ancha) y dos filas (la segunda el doble de alta).

## Grid Gap

La propiedad grid-gap nos permite añadir espacio entre las celdas del Grid. Podemos especificar un valor único para aplicar el mismo espacio en filas y columnas, o dos valores separados por un espacio para defi-

nir el espacio de filas y columnas por separado.

*Utilizando la misma estructura HTML del ejemplo anterior:*

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 100px 100px;
  grid-gap: 10px; /* Espacio de 10px entre todas las celdas */
}

.item {
  border: 1px solid black;
}
```

## Posicionando Grid Items

Estas propiedades nos permiten especificar en qué columnas y filas debe ubicarse un Grid Item.

## Grid-column y grid-row

Estas propiedades nos permiten especificar en qué columnas y filas debe ubicarse un Grid Item.

Para mejor visualización del ejemplo el grid container tendrá un borde de color rojo y contendrá 6 elementos y cada elemento tendrá un borde negro.

Además para poder distinguir mejor cual es el elemento con el cual estaremos trabajando, se le dará un color de fondo azul.

Así se vería el grid container sin modificación:

Item 1	Item 2	Item 3
Item 4	Item 5	Item 6

Ahora aplicaremos al primer elemento hijo (first-child) que es el Item 1 las siguientes propiedades:

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 100px 100px;
  border: 2px solid red;
}

.item {
  border: 1px solid black;
}

.item:first-child {
  background-color: blue;

  /* El elemento ocupa de la columna 2 a la 4 */
  grid-column: 2 / 4;

  /* El elemento ocupa de la fila 1 a la 3 */
  grid-row: 1 / 3;
}
```

Dando como resultado la siguiente grilla:

Item 2	Item 1	
Item 3		
Item 4	Item 5	Item 6

En este ejemplo, el elemento item 1 se extenderá desde la columna 2 hasta la 4, y desde la fila 1 hasta la 3, ocupando un área de 2x2 celdas.

## Grid-area

En lugar de definir columnas y filas por separado, podemos utilizar la propiedad grid-area para especificar un área rectangular directamente.

Utilizaremos el mismo elemento hijo y la misma estructura del ejemplo anterior para la demostración:

```
.item:first-child {
    background-color: blue;

    /* fila-inicio / columna-inicio / fila-final / columna-final */
    grid-area: 2 / 1 / 4 / 3;
}
```

Dando como resultado:

Item 2	Item 3	Item 4
Item 1		Item 5
		Item 6

Aquí, estamos indicando que el elemento .item1 debe ocupar el área que comienza en la fila 2, columna 1, y termina en la fila 4, columna 3.

## Alineación dentro de la celda

Además de definir dónde se ubica un elemento dentro del Grid, también podemos controlar su alineación dentro de cada celda individual utilizando las propiedades justify-self y align-self.

```
.item:first-child {
    background-color: blue;

    justify-self: center; /* Centrado horizontalmente */
    align-self: end; /* Alineado al final verticalmente */
}
```

Resultado:

	Item 2	Item 3
Item 1		
Item 4	Item 5	Item 6

Como podemos observar, el elemento se centró horizontalmente con `justify-self` y verticalmente se situó al final del eje. Así también la celda redujo y adaptó su tamaño al contenido del mismo.

## Áreas nombradas

Una de las características más poderosas de CSS Grid es la capacidad de definir áreas nombradas en la cuadrícula. Esto facilita la colocación de elementos en áreas específicas y la creación de diseños complejos.

Para definir áreas nombradas, utilizamos la propiedad `grid-template-areas` en nuestro Grid Container.

```
<body>
  <div class="container">
    <header class="header">Header</header>
    <aside class="sidebar">Sidebar</aside>
    <main class="content">Content</main>
    <aside class="sidebar2">Sidebar 2</aside>
  </div>
</body>
```

```
.container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  grid-template-rows: 100px 200px;
  border: 2px solid red;

  grid-template-areas:
    "header header header"
    "sidebar content sidebar2";
}
```

Con `grid-template-areas` le damos un orden de columnas y filas a cada uno de los elementos de nuestro HTML en nuestra grilla.

Por último falta vincular la etiqueta HTML con el nombre que le dimos en nuestro `grid-template-areas` y lo haremos con la propiedad `grid-area` de la siguiente manera:

```
.header {
  grid-area: header;

  border: 1px solid black;
  background-color: orange;
}

.sidebar {
  grid-area: sidebar;

  border: 1px solid black;
  background-color: blue;
}

.content {
  grid-area: content;

  border: 1px solid black;
  background-color: red;
}

.sidebar2 {
  grid-area: sidebar2;

  border: 1px solid black;
  background-color: green;
}
```

## Resultado:



Nótese que se agregó a cada uno de los elementos HTML las propiedades de `background-color` y `border` para una mejor visualización del ejemplo.

## Grid Implícito y Explícito

En CSS Grid, podemos definir explícitamente el número de filas y columnas utilizando las propiedades `grid-template-rows` y `grid-template-columns`. Estas filas y columnas definidas explícitamente se conocen como la "Grid explícita".

Sin embargo, si tenemos más elementos (Grid Items) que celdas definidas en la Grid explícita, CSS Grid creará automáticamente filas y columnas adicionales para acomodar estos elementos restantes. Estas filas y columnas adicionales se conocen como la "Grid implícita".

Por ejemplo:

Si tengo una grilla de 2 filas y 2 columnas (4 grid cells) pero tengo 5 grid items lo que me resultaría es que tengo un grid-item sobrante:

Item 1	Item 2
Item 3	Item 4
Item 5 (Grid Implícito)	

Para controlar el tamaño de las filas y columnas implícitas, podemos utilizar las propiedades `grid-auto-rows` y `grid-auto-columns`.

### Grid-auto-rows

Esta propiedad define el tamaño de las filas implícitas creadas por CSS Grid.

```
.container {
  display: grid;
  /* Dos columnas explícitas */
  grid-template-columns: 1fr 1fr;

  /* Filas implícitas de 100px de alto
  grid-auto-rows: 100px;

}

.item {
  border: 1px solid black;
}
```

Item 1	Item 2
Item 3	Item 4
Item 5 (Grid Implícito)	

En este ejemplo, hemos definido dos columnas explícitas, pero no hemos definido filas explícitas. Como tenemos cinco elementos, CSS Grid creará tres filas implícitas, cada una de 100px de alto, para acomodar todos los elementos.

### Grid-auto-columns

De manera similar, `grid-auto-columns` define el ancho de las columnas implícitas pero con una diferencia:

La propiedad se utiliza para definir el ancho de las columnas implícitas que se crean cuando un elemento se **posiciona** fuera del rango de las columnas definidas explícitamente con `grid-template-columns`.

Veamos con el ejemplo:

```
.container {
    display: grid;

    /* Definimos 2 columnas explícitas */
    grid-template-columns: 150px 150px;

    /* Ancho de las columnas implícitas fuera del rango */
    grid-auto-columns: 200px;

    /* Definimos 2 filas explícitas */
    grid-template-rows: 150px 150px;
}
```

En nuestro container estamos definiendo 2 columnas y 2 filas de 150 píxeles cada una y con grid-auto-columns prevenimos en el caso que posicionemos un grid-item fuera de las columnas ya establecidas creando una nueva columna de 200px.

Pongamos el 3er elemento de nuestro HTML en una 3er columna que no existe y le daremos unos estilos para identificarlo:

Item 1	Item 2	Item 3 (Grid Implicito)
Item 4	Item 5	Imagen 89

```
.item {
    border: 1px solid black;
}

.item:nth-child(3) {
    border: 1px solid black;
    background-color: blue;

    grid-column: 3;
}
```

Resultado (ver *Imagen 89*):

## Funciones repeat y minmax

Estas funciones nos permiten definir patrones repetidos y tamaños flexibles para nuestras filas y columnas en CSS Grid.

### Función repeat()

La función repeat() nos permite repetir un patrón de tamaños de columnas o filas.

Para utilizar la función nos pide 2 argumentos: la cantidad de veces que vamos a repetir, y el valor que vamos a repetir.

Utilizaremos 9 grid-items para el ejemplo:

```
<body>
  <div class="container">
    <div class="item">Item 1</div>
    <div class="item">Item 2</div>
    <div class="item">Item 3</div>
    <div class="item">Item 4</div>
    <div class="item">Item 5</div>
    <div class="item">Item 6</div>
    <div class="item">Item 7</div>
    <div class="item">Item 8</div>
    <div class="item">Item 9</div>
  </div>
</body>
```

```
.container {
  display: grid;
  border: 2px solid red;

  /* Tres columnas de 1fr cada una */
  grid-template-columns: repeat(3, 1fr);
  /* Tres filas de 100px cada una */
  grid-template-rows: repeat(3, 100px);
}

.item {
  border: 1px solid black;
}
```

## Resultado:

Item 1	Item 2	Item 3
Item 4	Item 5	Item 6
Item 7	Item 8	Item 9

Como podemos observar, en las propiedades `grid-template-columns` y `grid-template-rows` utilizamos la función `repeat` dandole como primer valor 3 y como segundo valor el tamaño que queremos repetir.

Por lo que podemos concluir que:

`grid-template-rows: 100px 100px 100px;`  
*es lo mismo que*  
`grid-template-rows: repeat(3, 100px)`

## Función minmax()

La función `minmax()` nos permite establecer un tamaño mínimo y un tamaño máximo para una columna o fila.

```
.container {
  display: grid;

  /* Tres columnas con un mínimo de 200px y un máximo de 350 */
  grid-template-columns: repeat(3, minmax(200px, 350px));

  /* Dos filas con un mínimo de 100px y un máximo automático */
  grid-template-rows: repeat(3, minmax(100px, auto));
}

.item {
  border: 1px solid black;
}
```

Esto quiere decir que si reduzco la ventana de mi navegador, las columnas se irán haciendo más chicas hasta llegar a un ancho de 200px como mínimo y en ese punto deja de achicarse.

Item 1	Item 2	Item 3
Item 4	Item 5	Item 6
Item 7	Item 8	Item 9

Como podemos ver, la 3er columna ya no se redujo en tamaño, si no que ahora el navegador me habilitó una barra gris en la parte inferior para poder hacer un scroll lateral, mante-

niendo así el ancho de 200px que declaramos como mínimo.

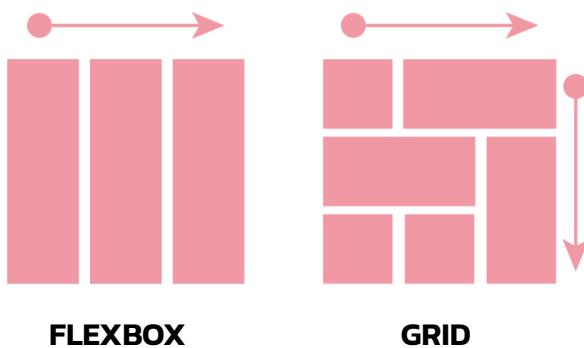
Y como se podrán imaginar, pasará lo mismo con el tamaño máximo de 350px que declaramos. Podremos agrandar el navegador y las columnas se agrandarán hasta llegar a un ancho de 350px. Una vez que cumplen esa condición, dejan de crecer y empieza a aparecer espacio sobrante en el lateral.

Item 1	Item 2	Item 3
Item 4	Item 5	Item 6
Item 7	Item 8	Item 9

CSS Grid es una herramienta poderosa que te brinda un control sin precedentes sobre la disposición de los elementos en tu página web. Aunque puede requerir un poco de práctica al principio, te recomiendo explorar más ejemplos y seguir practicando para dominar esta técnica de layout.

Combínalo con Flexbox y otras técnicas de CSS para crear diseños verdaderamente impresionantes.

*CSS Grid es una herramienta poderosa que te brinda un control sin precedentes sobre la disposición de los elementos en tu página web.*



# Metodología BEM

**BEM es una metodología de nomenclatura para escribir código CSS más modular, reutilizable y fácil de mantener. Se basa en tres conceptos principales: Bloque, Elemento y Modificador.**

**Bloque:** Representa un componente independiente y autónomo de la interfaz de usuario. Es la parte principal y contenedora de la estructura.

**Elemento:** Es una parte de un bloque que tiene sentido solo dentro de ese bloque. Los elementos no se utilizan fuera del contexto de su bloque padre.

**Modificador:** Es una variación de un bloque o elemento que altera su apariencia o comportamiento.

## Nomenclatura

La nomenclatura de BEM sigue un patrón estricto que nos ayuda a mantener una estructura clara y consistente en nuestro código CSS. Los patrones son los siguientes:

```
.bloque{}  
.bloque__elemento{}  
.bloque--modificador{}  
.bloque__elemento--modificador{}
```

*.bloque para el bloque base.  
.bloque\_\_elemento para los elementos que conforman el bloque.  
.bloque--modificador para los modificadores del bloque.  
.bloque\_\_elemento--modificador para los modificadores de un elemento específico.*

Siguiendo esta nomenclatura, podemos identificar fácilmente qué es cada pieza de código y a qué parte de la interfaz pertenece. Además, al utilizar una sintaxis tan específica, evitamos conflictos de nombres entre diferentes bloques, elementos y modificadores.

## Ejemplo temático

Para desarrollar la metodología BEM estaremos trabajando con la siguiente estructura HTML. Para mejor entendimiento se recomienda avanzar con la lectura y comparar lo leído con dicha estructura para comprenderlo.

```
<section class="gallery">  
  <h2 class="gallery__title">Galería de fotos</h2>  
  <ul class="gallery_list">  
    <li class="gallery__item">  
        
      <p class="gallery__caption">Descripción de la foto 1</p>  
    </li>  
    <li class="gallery__item">  
        
      <p class="gallery__caption">Descripción de la foto 2</p>  
    </li>  
    <li class="gallery__item">  
        
      <p class="gallery__caption">Descripción de la foto 3</p>  
    </li>  
  </ul>  
</section>
```

## Bloques

Un bloque es la unidad modular e independiente más grande de la metodología BEM. Representa un componente reutilizable de la interfaz de usuario que tiene un propósito y

funcionalidad específica.

Por ejemplo, en nuestra galería de fotos, el bloque principal sería `.gallery`, que **engloba toda la funcionalidad y elementos relacionados con mostrar una colección de imágenes**.

Los bloques deben ser únicos en todo el proyecto y no deben anidarse unos dentro de otros, ya que esto violaría el principio de independencia. Cada bloque se encarga de una pieza de funcionalidad específica y se puede reutilizar en diferentes partes de la aplicación sin afectar a otros componentes.

Cuando definimos los estilos de un bloque, lo hacemos mediante una clase con el mismo nombre del bloque, por ejemplo, `.gallery {}`. Aquí colocaremos todos los estilos base que se aplicarán a todo el componente, como el ancho, el margen, el fondo, etc.

## Elementos

Los elementos son partes individuales que conforman un bloque y no tienen sentido por sí mismos fuera del contexto del bloque.

Por ejemplo, en nuestro bloque `.gallery`, los elementos son `.gallery__title` (el título de la galería), `.gallery__list` (la lista de imágenes), `.gallery__item` (cada elemento individual de la lista), `.gallery__img` (la imagen en sí) y `.gallery__caption` (la descripción de la imagen).

Los elementos siempre deben ir precedidos por el nombre del bloque al que pertenecen, **separados por dos guiones bajos \_\_**. Esto nos ayuda a mantener una estructura clara y evitar conflictos de nombres con otros elementos o bloques.

Al definir los estilos de un elemento, lo hacemos mediante una clase que combine el nombre del bloque y el nombre del elemento, por

ejemplo, `.gallery__title {}`. Aquí colocaremos todos los estilos específicos que se aplicarán únicamente a ese elemento dentro del bloque.

## Modificadores

Los modificadores son una especie de “flags” que se utilizan para cambiar la apariencia o el comportamiento de un bloque o un elemento específico.

Por ejemplo, podríamos tener un modificador `.gallery__item--highlighted` que se aplicaría a un elemento de la galería para resaltarlo visualmente.

Los modificadores se representan con **dos guiones --** después del nombre del bloque o elemento al que se aplican. Pueden ir solos o combinados con elementos.

La principal ventaja de los modificadores es que nos permiten crear variaciones de un bloque o elemento sin tener que duplicar código o crear nuevas clases.

Simplemente aplicamos el modificador correspondiente y definimos los estilos específicos para esa variación.

Cuando definimos los estilos de un modificador, lo hacemos mediante una clase que combine el nombre del bloque o elemento y el nombre del modificador, por ejemplo, `.gallery__item--highlighted {}`. Aquí colocaremos todos los estilos específicos que se aplicarán a ese elemento cuando tenga el modificador aplicado.

**Resultado del ejemplo:**

```
/* Bloque */
.gallery {
  max-width: 800px;
  margin: 0 auto;
}

/* Elementos */
.gallery__title {
  text-align: center;
  font-size: 2rem;
  margin-bottom: 1rem;
}

.gallery__list {
  list-style: none;
  padding: 0;
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
  grid-gap: 1rem;
}
```

```
.gallery__item {
  background-color: #f0f0f0;
  padding: 1rem;
  border-radius: 4px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.gallery__item--highlighted {
  background-color: #ffffd6;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
}

.gallery__img {
  max-width: 100%;
  height: auto;
  display: block;
  margin-bottom: 0.5rem;
}

.gallery__caption {
  font-size: 0.9rem;
  color: #666;
  margin: 0;
}
```

## Galería de imágenes



Image1



Image2



Image3

Descripción de la foto 1

Descripción de la foto 2

Descripción de la foto 3

## Ventajas de BEM

La metodología BEM nos ofrece varias ventajas significativas en el desarrollo de interfaces de usuario:

**1**

### Modularidad

Al dividir nuestra interfaz en bloques independientes y reutilizables, promovemos la modularidad y la capacidad de reutilizar componentes en diferentes partes de la aplicación.

**2**
**Estructura clara:**

La nomenclatura estricta de BEM nos ayuda a mantener una estructura clara y organizada en nuestro código CSS, lo que facilita la comprensión y el mantenimiento a largo plazo.

**3**
**Especificidad plana:**

Todas las clases en BEM tienen la misma especificidad, lo que evita conflictos de especificidad y facilita la sobrescritura de estilos.

**4**
**Reutilización:**

Al tener bloques independientes y reutilizables, podemos reutilizar componentes completos en diferentes partes de la aplicación sin tener que duplicar código.

**5**
**Escalabilidad:**

A medida que nuestro proyecto crece, BEM nos permite mantener un código organizado y fácil de mantener, lo que facilita la escalabilidad y el trabajo en equipo.

**6**
**Consistencia:**

Al seguir una metodología estricta, promovemos la consistencia en el código CSS de todo el proyecto, lo que facilita que diferentes desarrolladores puedan trabajar en el mismo código sin introducir inconsistencias.

Estas ventajas hacen de BEM una metodología muy popular y ampliamente utilizada en el desarrollo de interfaces de usuario modernas y escalables.

## Créditos y atribuciones

---

\* **IMAGEN 1** <a href="https://www.freepik.es/vector-gratis/profesionales-ti-estan-creando-sitio-web-ilustracion-pantalla-portatil\_10780362.htm#page=7&position=11&from\_view=author&uuid=04b33207-47e4-4433-8187-3d2143f4e917">Imagen de vectorjuice</a> en Freepik