

# Diseño CCS

## Introducción

¡Te damos la bienvenida a esta sección sobre diseño CSS! 

Ya aprendimos sobre los conceptos básicos de CSS y aplicamos algunos estilos simples a nuestra página HTML.

En esta clase, **exploraremos técnicas de CSS más avanzadas para mejorar la disposición y el diseño de nuestra página web**. Cubriremos *Box Model*, *Display* y *Position* (“*visualización y posicionamiento*”), *Flexbox* y *Grid*, y el *diseño responsivo*.

Al final de esta sección, podrás mejorar la disposición y el diseño de la página HTML en la que hemos estado trabajando.

¡Vamos!

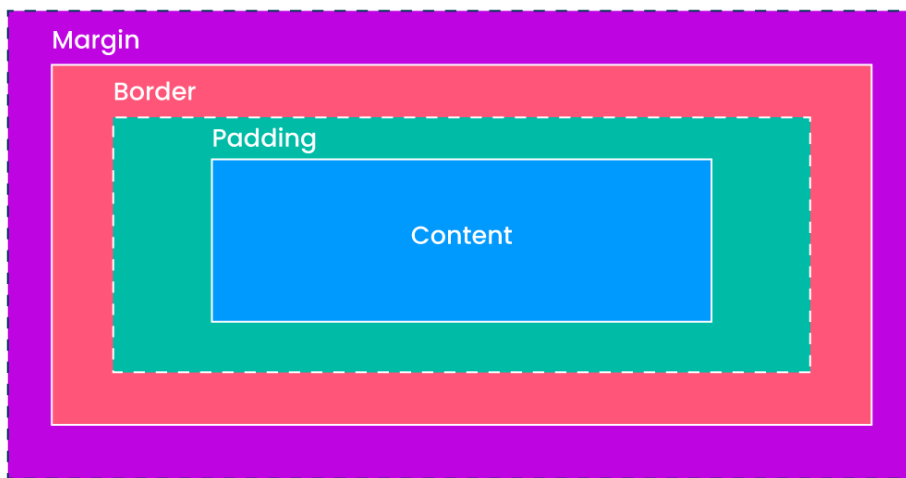
## Modelo de cajas

El **Box Model** (“modelo de caja”) es uno de los conceptos más importantes en CSS, y es fundamental para comprender cómo se estructuran y diseñan los elementos en una página web.

Describe cómo se componen los elementos HTML en una página. Cada elemento se considera una “caja” que consta de cuatro partes:

- Content
- Padding
- Border
- Margin

## Box Model CSS



Veamos en más detalle de qué trata cada parte:

**Content** → Es el área donde se muestra el *contenido real* del elemento (como el texto o las imágenes).

**Padding** → La propiedad de *padding* ("relleno") se usa para definir el *espacio entre el contenido de un elemento y su borde*. Los valores que se pueden utilizar incluyen:

- Un *valor numérico*, que define el espacio en píxeles.
- Un *valor porcentual*, que define el espacio en relación al ancho del elemento.
- La palabra clave *"auto"*, que centra el contenido horizontalmente.

**Border** → La propiedad de *border* se usa para definir el *borde de un elemento*, es decir, es la línea que rodea el contenido y el padding. Los valores que se pueden utilizar incluyen:

- El *ancho del borde* en píxeles, como `"1px"`.
- El *tipo de borde*, como `"solid"` para un borde sólido o `"dashed"` para un borde discontinuo.
- El *color del borde*, como `"black"` para un borde negro o `"#cccccc"` para un color hexadecimal.

**Margin** → La propiedad de *margin* ("margen") se usa para definir el *espacio entre el borde y el elemento adyacente* (como otro elemento o el borde del navegador). Los valores que se pueden utilizar incluyen:

- Un *valor numérico*, que define el espacio en píxeles.
- Un *valor porcentual*, que define el espacio en relación al ancho del elemento padre.

- La palabra clave *"auto"*, que centra el elemento horizontalmente.

Veamos un ejemplo de cómo se aplica el *box model* a un elemento HTML simple:

```
<div class="ejemplo">
  Este es un ejemplo de contenido
</div>
```

En este caso, el elemento es un *"div"* con la clase *"ejemplo"*. El contenido es simplemente el texto: *"Este es un ejemplo de contenido"*.

El *padding*, *border* y *margin* no se han definido explícitamente, por lo que serían 0.

💡 De todas maneras, el navegador por defecto agrega estilos. Por lo que, si quisiéramos resetearlos, vamos a poder hacerlo con el siguiente código CSS:

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

Ahora, si aplicamos algunos estilos CSS para agregar *padding*, *border* y *margin*, se podría declarar así:

```
.ejemplo {
  padding: 10px;
  border: 1px solid black;
  margin: 20px;
}
```

💡 Como podemos observar en la imagen, **la propiedad de border tiene asignados varios valores en una misma línea** (el ancho, tipo y color). Se escribe de esa manera para simplificar el código y que sea más legible y eficiente.

Y el resultado final se vería así:



Como se puede ver, el contenido está rodeado por un espacio de 10px de padding, luego por un borde negro sólido de 1px, y finalmente por un margen de 20px. *Todo esto compone la "caja" del elemento.*

Ahora bien, si quisiéramos darle diferentes valores a los espacios de cada uno de los cuatro lados del elemento, **las propiedades de padding y margin se pueden declarar junto con la palabra: top, right, bottom o left**, para especificar la cantidad de espacio que se desea agregar en una dirección específica. Por ejemplo:

Para agregar 200px de margen en la parte superior y a la parte derecha de un elemento, pero no en la parte izquierda y en la parte inferior, podríamos utilizar la siguiente regla CSS:

```
.ejemplo {
  padding: 10px;
  border: 1px solid black;
  margin-top: 200px;
  margin-right: 200px;
}
```

Y el resultado final se vería así:

HTML

```
1 <div class="ejemplo">
2   Este es un ejemplo de contenido
3 </div>
```


CSS

```
1 .ejemplo {
2   padding: 10px;
3   border: 1px solid black;
4   margin-top: 200px;
5   margin-right: 200px;
6 }
```

Este es un ejemplo de contenido

Otra forma de declarar los *margins* que vimos anteriormente sería la siguiente:

```
.ejemplo {
  padding: 10px;
  border: 1px solid black;
  margin: 200px 200px 0px 0px;
}
```

 Cuando el *margin* se declara de manera abreviada se divide en cuatro valores separados por espacios. El primer valor corresponde al margen superior (*top*), el segundo al margen derecho (*right*), el tercero al margen inferior (*bottom*) y el cuarto al margen izquierdo (*left*).

De esta manera, al asignar "200px 200px 0px 0px" se está indicando un margen superior e inferior de 200px y un margen derecho e izquierdo de 0px, simplificando

la escritura del código y haciendo que sea más fácil de leer. Y el resultado que obtenemos será el mismo:



Este es un ejemplo de contenido

## Actividad de Box Model

Para que practiques y comprendas mejor cómo funciona el Box Model de CSS y las propiedades que lo rodean, tales como *tamaño*, *posicionamiento*, *border*, *margin* y *padding* de un elemento, te proponemos hacer un ejercicio.

La práctica del **Box Model** es fundamental para que puedas construir páginas web bien estructuradas y adaptativas a diferentes dispositivos y tamaños de pantalla. Además, te permitirá desarrollar habilidades de diseño más avanzadas y crear sitios web de alta calidad y usabilidad.

**¡Manos a la obra!**

1. Crear un archivo HTML con un elemento *"div"* y dale una *clase*.

```
<!DOCTYPE html>
<html>
<head>
  <title>Box Model Exercise</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="box">
    <p>Contenido de la caja</p>
  </div>
</body>
</html>
```

2. Crear un archivo CSS llamado *"style.css"* y agregar estilos al elemento *"div"* utilizando el **Box Model**. Agregar un *border*, *margin*, *padding* y ajustar su tamaño.

```
.box {
  background-color: lightgrey;
  width: 300px;
  height: 150px;
  padding: 20px;
  border: 5px solid black;
  margin: 50px;
}
```

3. Abrir el archivo HTML en tu navegador y observar cómo se aplica el **Box Model** al elemento *"div"*.


4. Modificar los valores de *padding*, *border* y *margin* para ver cómo afectan la apariencia del elemento.

En este video te mostramos cómo inspeccionar un elemento para que puedas interactuar directamente con el código HTML que lo conforma.

 [Aquí puedes ver como inspeccionar tu archivo HTML en el navegador](#)

5. Añadir 2 elementos “div” más con diferentes clases y aplicar diferentes estilos de Box Model para cada uno.

### [Instalación Extensión Live server](#)

 **TIP:** En este video te contamos como funciona la extensión de Visual Studio Code conocida como Live Server; un módulo de software que agrega funcionalidades al editor de código, como la opción de ver tu página en vivo en un navegador real.

3... 2... 1... ¡Stop!

¡Llegamos al final de este ejercicio! En este video te mostramos paso a paso la forma de solucionarlo junto con el resultado esperado.

### [Ver video](#)

## Display y Position

Recordemos que **cada elemento HTML en una página web es una “caja rectangular”**.

Las propiedades de **display** y **position** determinan cómo se comporta esa caja rectangular. Es decir, **se utilizan para controlar la visualización y el posicionamiento de los elementos HTML**.

Algunos valores comunes para la propiedad de **display** (“visualización”) incluyen:

- **none:** Oculta completamente el elemento, es decir, *no se muestra en la página web*. Es muy útil para hacer que un elemento desaparezca temporalmente o para ocultarlo en diferentes condiciones de visualización (esto se logra manipulando el CSS con javascript).
- **inline:** Convierte un elemento en un *elemento en línea*. Los elementos en línea son aquellos que no comienzan una nueva línea y sólo ocupan el ancho necesario para mostrar su contenido. Los elementos en línea pueden contener elementos en línea y texto, pero no otros bloques.
- **block:** Convierte un *elemento en un bloque*. Los elementos de bloque comienzan una nueva línea y ocupan todo el ancho disponible. Los elementos



de bloque pueden contener elementos en línea, otros bloques e incluso otros elementos de tipo inline-block.

- **inline-block:** Combina las características de los elementos inline y los elementos block. Los elementos inline-block *se muestran como elementos en línea, pero permiten establecer ancho y altura como si fueran elementos de bloque.*
- **inherit:** Hace que el elemento herede el *valor de la propiedad display de su elemento padre.*
- **flex:** Convierte un elemento en un *contenedor flexible*, lo que permite controlar el tamaño y la posición de sus elementos hijos.
- **grid:** Convierte un elemento en un *contenedor de cuadrícula*, lo que permite colocar elementos hijos en filas y columnas.

## [Ver video](#)

Y, algunos valores comunes para la propiedad de **position** (“posicionamiento”) incluyen:

- **static:** Este es el valor predeterminado de la propiedad de "position". Cuando se establece el valor de "static", *no se aplica ningún posicionamiento especial al elemento* y se colocará en el lugar que le corresponda en el flujo normal del documento.
- **relative:** El elemento se posiciona en relación con su posición normal, es decir, *puede moverse hacia arriba, abajo, izquierda o derecha desde su posición inicial.*
- **absolute:** El elemento *se posiciona en relación a su “ancestro posicionado” más cercano*, es decir, al “elemento padre” que tenga un valor de posición distinto a "static". Si no hay ningún “ancestro posicionado”, el elemento se posiciona en relación al bloque contenedor inicial del documento (generalmente al “body”). Cuando se establece el valor de “absolute” en un elemento, se elimina del flujo normal del documento, lo que significa que no afecta a la posición de otros elementos en la página.
- **fixed:** El elemento se posiciona en relación con la ventana del navegador, lo que significa que *permanece en la misma posición incluso cuando se desplaza la página.*

Cuando se establece el valor de “fixed” en un elemento, también se elimina del flujo normal del documento. El elemento se coloca en la posición especificada con las propiedades “top”, “right”, “bottom” y “left”. Si no se especifica ninguna de estas propiedades, el elemento se coloca en la esquina superior izquierda de su “ancestro posicionado” más cercano.

- **sticky:** El elemento se comporta como un elemento posicionado relativamente hasta que alcanza una posición específica en la página, donde se vuelve fijo.


- **inherit:** Hereda el comportamiento del elemento padre para dicha propiedad

 [Ver video](#)

## Flexbox y Grid

**Flexbox** y **Grid** son *módulos modernos de diseño CSS* que ofrecen opciones poderosas y flexibles para crear diseños responsivos.

En esta clase descubriremos cómo estas dos tecnologías han revolucionado la forma en que los diseñadores y desarrolladores abordan la creación de sitios web flexibles y adaptativos. Si bien hablaremos de las dos, haremos mayor énfasis en Flexbox.

 *Pssst, ¿Te gustaría entender mejor qué son los módulos de diseño? ¡Muy bien! Puedes usar el siguiente “prompt” en el **chat GPT**: “Explicar de manera simple y concisa, y que sea fácil de entender. ¿Qué son los módulos modernos de diseño CSS?”.*

**Flexbox está diseñado para diseños unidimensionales**, lo que significa que **se utiliza para crear diseños en una sola dirección, ya sea en filas o columnas**. Con Flexbox, podemos controlar la alineación, el tamaño y el orden de los elementos en su diseño. Es especialmente útil para diseñar elementos como menús de navegación, listas de elementos, elementos de una página de perfil de usuario y formularios.

Por otro lado, **Grid está diseñado para diseños bidimensionales**, lo que significa que **se utiliza para crear diseños en dos direcciones, tanto filas como columnas**. Con Grid, podemos crear diseños complejos y precisos que permiten la colocación de elementos en cualquier lugar dentro del contenedor de diseño. Es especialmente útil para diseñar elementos como diseños de página, diseños de tarjetas y diseños de revistas.

Ambos módulos *tienen algunas características en común* como:

- Capacidad de controlar el espaciado entre los elementos.
- Capacidad de hacer que los elementos cambien de tamaño y posición automáticamente en función del tamaño de la pantalla del usuario.
- Capacidad de controlar el flujo del contenido.

También son **compatibles con la técnica de diseño responsivo**, lo que significa que se pueden ajustar automáticamente para adaptarse a diferentes tamaños de pantalla y dispositivos.

💡 *Son dos herramientas útiles para el diseño web y ofrecen una amplia variedad de opciones para crear diseños flexibles y responsivos. La elección de cuál usar dependerá de la naturaleza del proyecto y del tipo de diseño que se esté buscando.*

Ahora, te invitamos a abrir el enlace del material complementario y a leer acerca de las **propiedades de Flexbox** y sus posibles valores:

🔗 [Leer sobre \[Propiedades de Flexbox\]](#)

¿Demasiada información? ¡No te preocupes! Vas a poder ver el siguiente video el cual te ayudará a visualizar y comprender mejor cómo funciona Flexbox:

🔗 [Ver video](#)

¡Muy bien! Ahora continuemos con las **propiedades de Grid** y sus posibles valores. No tienes que leerlo ahora, puedes hacerlo cuando lo consideres necesario:

🔗 [Leer sobre \[Propiedades de Grid\]](#)

💡 *¿Te gustaría poder practicar de una forma didáctica y divertida tanto Flexbox como Grid? Te compartimos los enlaces de los siguientes juegos para que puedas hacerlo en tu tiempo libre:*

- [Flexbox Froggy](#)
- [Grid Garden](#)

## Diseño responsivo

El **diseño responsivo** (“responsive”) es la práctica de crear páginas web que se adaptan automáticamente a diferentes tamaños y orientaciones de pantalla, brindando una experiencia de visualización óptima en varios dispositivos (de escritorio, tablets o smartphones).

Para realizar un diseño responsivo se utilizan las “*media queries*”, que son una técnica de CSS que permite aplicar estilos específicos a un documento HTML en función de las características del dispositivo que lo está visualizando, como el tamaño de la pantalla, la resolución, la orientación, entre otros. De esta manera, es posible crear diseños web adaptables y optimizados para diferentes dispositivos.

Las *media queries* se definen usando la sintaxis “**@media**”, seguido de un conjunto de reglas CSS.

```
@media (regla-de-media-queries: valor) {  
  
    reglas css  
  
}
```

A continuación, te compartimos todas las opciones de uso posible de las *media queries*:

- **min-width:** Se aplica cuando la anchura de la pantalla es mayor o igual a un valor específico.
- **max-width:** Se aplica cuando la anchura de la pantalla es menor o igual a un valor específico.
- **min-height:** Se aplica cuando la altura de la pantalla es mayor o igual a un valor específico.
- **max-height:** Se aplica cuando la altura de la pantalla es menor o igual a un valor específico.
- **orientation:** Se aplica según la orientación del dispositivo, ya sea portrait (vertical) o landscape (horizontal).
- **aspect-ratio:** Se aplica según la relación entre el ancho y la altura de la pantalla.
- **device-aspect-ratio:** Se aplica según la relación entre el ancho y la altura del dispositivo.
- **resolution:** Se aplica según la resolución de la pantalla en píxeles por pulgada (PPI) o en puntos por pulgada (DPI).
- **min-resolution:** Se aplica cuando la resolución de la pantalla es mayor o igual a un valor específico.
- **max-resolution:** Se aplica cuando la resolución de la pantalla es menor o igual a un valor específico.
- **pointer:** Se aplica según el tipo de dispositivo de entrada que se esté utilizando, como none (para pantallas táctiles) o coarse (para dispositivos con puntero menos preciso).
- **hover:** Se aplica según la presencia o no de un dispositivo de puntero que permita la interacción a través de hover.

- **any-pointer:** Se aplica si el dispositivo tiene algún tipo de dispositivo de entrada de puntero.
- **any-hover:** Se aplica si el dispositivo tiene algún tipo de dispositivo de puntero que permita la interacción a través de hover.

En el siguiente video te ayudamos a comprender mejor **cómo funcionan las medias queries más utilizadas:**

 [\*\*Ver video\*\*](#)

# Desafío del día

¡Llegó el momento de poner en práctica algunos de los conceptos vistos! El desafío de hoy consiste en **mejorar el diseño y la estructura de la página HTML** sobre la que viniste trabajando en los encuentros anteriores.

1. Abrir el archivo "styles.css" de la sección anterior y comenzar añadiendo un contenedor para centrar nuestro contenido. Agregar el siguiente código CSS:

```
.container {  
  
    max-width: 800px;  
  
    margin: auto;  
  
    padding: 20px;  
  
}
```

2. Ahora, abrir el archivo "mi-primer-website.html" y envolver el contenido dentro de la etiqueta <body> con un nuevo elemento <div> con una clase de **"container"**. Tu HTML actualizado debería similar a este:

```
<body>  
  <div class="container">  
    <h1>Bienvenidos a mi primer Website</h1>  
    <h2>Un subtítulo en h2</h2>  
    <p>Soy [Tu Nombre], ¡y esta es mi primera página web!</p>  
    <p>Estoy aprendiendo desarrollo web para crear sitios web  
    | visualmente atrapantes y funcionales.  
    </p>  
    <h2>Otro subtítulo en h2</h2>  
    <p>A lo largo de este curso, estaré aprendiendo HTML, CSS  
    | y JavaScript. Estos lenguajes me ayudarán a crear una base  
    | sólida en el desarrollo web front-end  
    </p>  
  </div>  
</body>
```

3. Guardar tu archivo HTML y actualizar el navegador. Ahora deberías ver el contenido de tu página web centrado dentro de un contenedor.

4. A continuación, crearemos un simple diseño responsivo de dos columnas utilizando Flexbox. Agregar el siguiente código CSS a tu archivo "styles.css":

```
.row {  
  
    display: flex;  
  
    flex-wrap: wrap;  
  
    margin: 10px;  
  
}
```

```
.column {  
  
    flex: 1;  
  
    padding: 10px;  
  
}
```

**5.** Actualizar tu archivo *"mi-primer-website.html"* para incluir un diseño de dos columnas. Envolver los párrafos existentes en un nuevo `<div>` con una clase de *"row"* y encerrar cada párrafo dentro de un `<div>` con una clase de *"column"*. Tu código HTML actualizado debería quedar similar al siguiente:

```
<body>  
  <div class="container">  
    <h1>Bienvenidos a mi primer Website</h1>  
    <div class="row">  
      <div class="column">  
        <h2>Un subtítulo en h2</h2>  
        <p>Soy [Tu Nombre], ¡y esta es mi primera página web!</p>  
        <p>Estoy aprendiendo desarrollo web para crear sitios web  
        | visualmente atrapantes y funcionales.  
        </p>  
      </div>  
      <div class="column">  
        <h2>Otro subtítulo en h2</h2>  
        <p>A lo largo de este curso, estaré aprendiendo HTML, CSS  
        | y JavaScript. Estos lenguajes me ayudarán a crear una  
        | base sólida en el desarrollo web front-end.  
        </p>  
      </div>  
    </div>  
  </div>  
</body>
```

**6.** Guardar tu archivo HTML y actualizar el navegador. Ahora deberías ver un diseño responsivo de dos columnas para tus subtítulos y párrafos.

7. Para hacer que el diseño sea verdaderamente responsivo, agregaremos una media query para apilar las columnas en pantallas más pequeñas. Agregar el siguiente código a tu archivo "styles.css":

```
@media (max-width: 600px) {  
  .column {  
    flex: 100%;  
  }  
}
```

**Este código aplicará el nuevo estilo a la clase ".column" cuando el ancho de la pantalla sea de 600 píxeles o menos**, haciendo que las columnas se apilen verticalmente.

8. Guardar tu archivo "styles.css" y probar la capacidad de respuesta de tu página web redimensionando la ventana del navegador. Recuerda que las columnas deberían apilarse cuando el ancho de la ventana sea de 600 píxeles o menos.

**¡Desafío terminado!** 🎉

## Resolución del desafío

En el siguiente video te compartimos un paso a paso de cómo resolver el desafío anterior:

📺 [Resolución de Ejercicio | Diseño CSS | Egg](#)

**¡Felicidades!** 🎊

Has mejorado con éxito el diseño y la estructura de tu página HTML. Hoy aprendiste sobre: *Box Model*, *Display* y *Position*, *Flexbox* y *Grid*, y el *diseño responsivo*. Estos conceptos te ayudarán a crear páginas web más complejas y visualmente agradables.

En el encuentro que le sigue, introduciremos JavaScript, un lenguaje de



programación potente que te permitirá agregar interactividad y contenido dinámico a tus sitios web.

## Valida tus conocimientos

Te facilitamos nuevamente los enlaces del [material complementario](#) para que puedas continuar profundizando sobre estos temas cuando más gustes:

- [Propiedades de Flexbox](#)
- [Propiedades de Grid](#)

Por último, te proponemos realizar la autoevaluación para poner a prueba los principales aprendizajes del día de hoy. ¿Vamos?

 [Realizar test](#)

Ten en cuenta que si has llegado hasta aquí, ya has cumplido con el objetivo del encuentro.

¡Hasta la próxima! 

## Mapa de conceptos vistos

