

«Talento Tech»

# Desarrollo Web 1

Clase 04



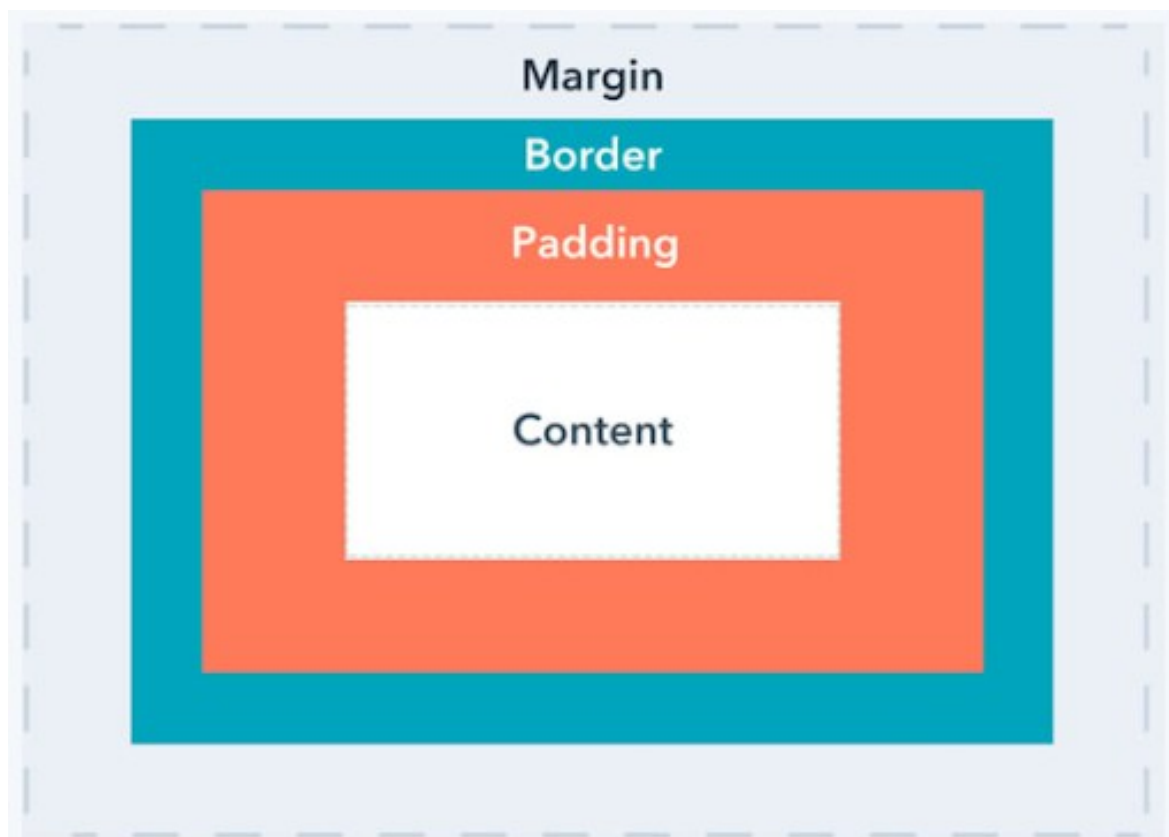


## Clase N° 4 | Conceptos básicos

### Temario:

- Margin y padding
- Concepto de herencia y especificidad en HTML
- Contenedores límites
- Posicionamiento de elementos utilizando "position"
- Z-index

## Margin y Padding:



El **margen (margin)** y el **relleno (padding)** son propiedades importantes en **CSS** que nos permiten controlar el espacio de alrededor como también el espacio dentro de los elementos de una página web.

Ambas propiedades (**margin y padding**) se pueden configurar en diferentes unidades de medida, como píxeles, porcentajes o em, según tus necesidades de diseño.

Además, puedes especificar valores diferentes para cada lado del elemento utilizando la notación abreviada, por ejemplo, **margin-top (superior)**, **margin-right (derecho)**,

**margin-bottom (inferior) y margin-left (izquierdo)**, o simplemente utilizando un solo valor para establecer el mismo margen o relleno en los cuatro lados.

El margen y el relleno son herramientas poderosas para controlar el diseño y la estructura de una página web. Con ellos, puedes crear espacios equilibrados, separar elementos visualmente y lograr diseños limpios y ordenados.

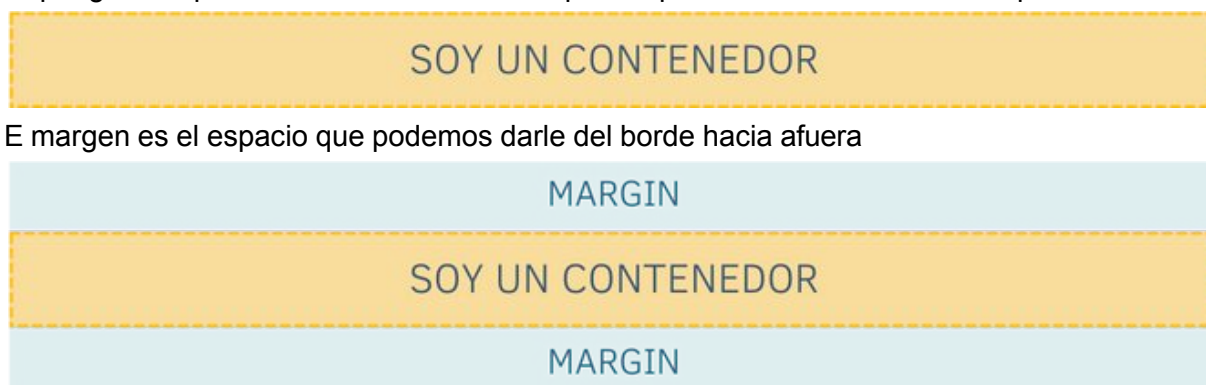
Es importante comprender cómo funcionan estas propiedades y cómo pueden afectar el diseño general de tu página.

[\\*Insertar presentacion Clase 03-Márgenes y Padding](#)

## Margin

Los márgenes con una propiedad CSS que sirve para separar elementos. Como vimos antes, hay muchos elementos que conforman un sitio web, y el espacio es uno de ellos. El margen determina la distancia entre el borde exterior del elemento y los elementos adyacentes.

Supongamos que tenemos un contenedor que ocupa el 100% del ancho de la pantalla



La propiedad margin se puede utilizar para establecer los márgenes superior, derecho, inferior e izquierdo de un elemento por separado, o se puede especificar un valor único para

aplicar el mismo margen a todos los lados. También es posible utilizar valores negativos para el margen, lo que hace que el elemento se superponga a otros elementos cercanos.

#### **Escritura individual:**

```
{  
  margin-top: 20px;  
  margin-bottom: 30px;  
  margin-left: 10px;  
  margin-right: 7px;  
}
```

#### **Escritura abreviada:**

```
{  
  margin: 20px, 30px, 10px, 7px;  
}  
    //arriba, derecha, abajo, izq.
```

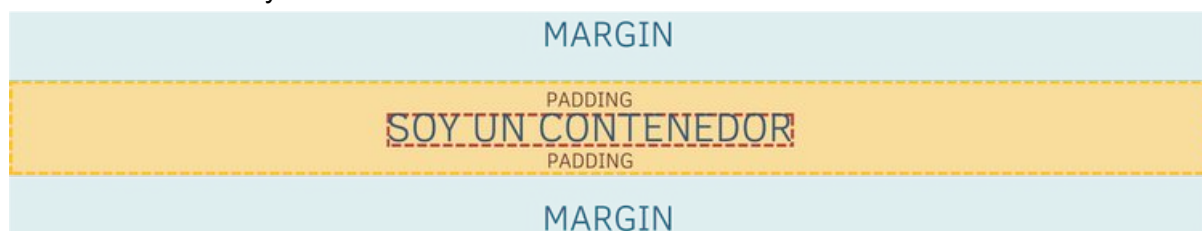
#### **Escritura abreviada en pares**

```
{  
  margin: 20px, 30px;  
}
```

//arriba y abajo, izq y der.

## **Padding**

El padding (relleno en español) es una propiedad que se utiliza para controlar el espacio de relleno alrededor del contenido de un elemento. El padding determina la distancia entre el borde del elemento y el contenido dentro de él.



La propiedad `padding` se puede utilizar para establecer el relleno, derecho, inferior e izquierdo de un elemento por separado, o se puede especificar un valor único para aplicar el mismo margen a todos los lados. También es posible utilizar valores negativos para el margen, lo que hace que el elemento se superponga a otros elementos cercanos.

#### **Escritura individual**

```
{  
  padding-top: 20px;  
  padding-bottom: 30px;  
  padding-left: 10px;  
  padding-right: 7px;  
}
```

#### **Escritura abreviada**

```
{  
  padding: 20px 30px 10px 7px;  
}
```

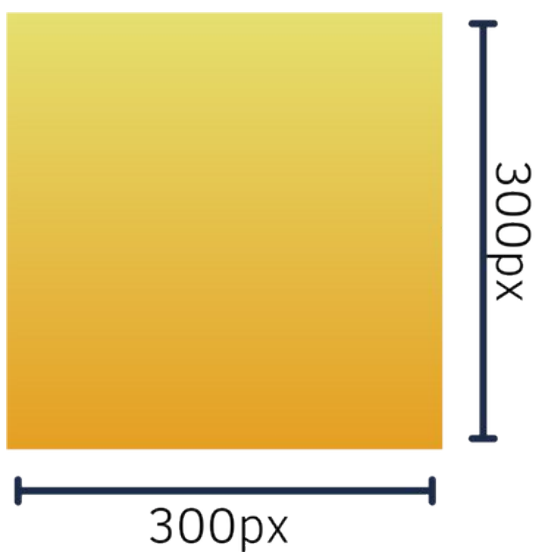
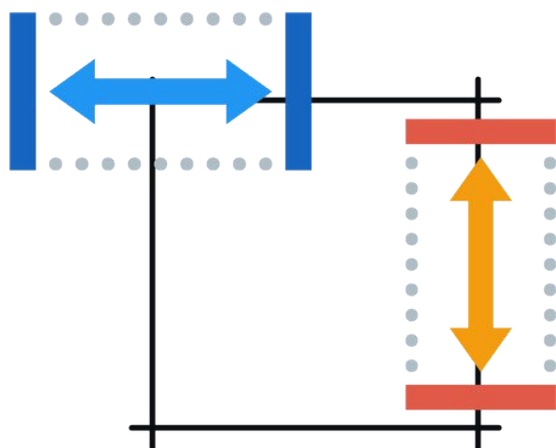
//arriba, derecha, abajo, izq.

#### **Escritura abreviada en pares**

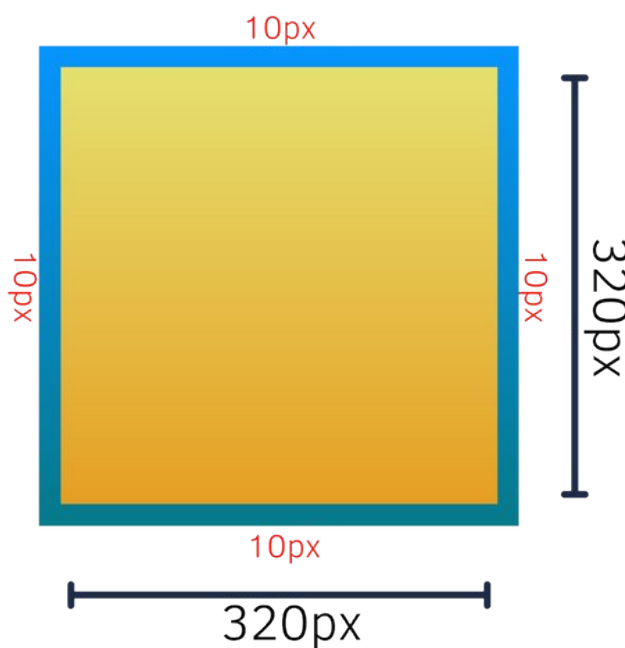
```
{  
  padding: 20px 30px;  
}
```

## **Box Sizing**

La propiedad **`box-sizing`** en **CSS** es una propiedad que se utiliza para controlar cómo se calcula el tamaño total de un elemento, teniendo en cuenta su contenido, relleno y borde. Por defecto, el modelo de caja CSS incluye el tamaño del contenido, el relleno y el borde en el cálculo del ancho y alto de un elemento. Esto significa que si especificas un ancho o alto determinado, el tamaño total del elemento será mayor, ya que se agregarán el relleno y el borde al tamaño del contenido.

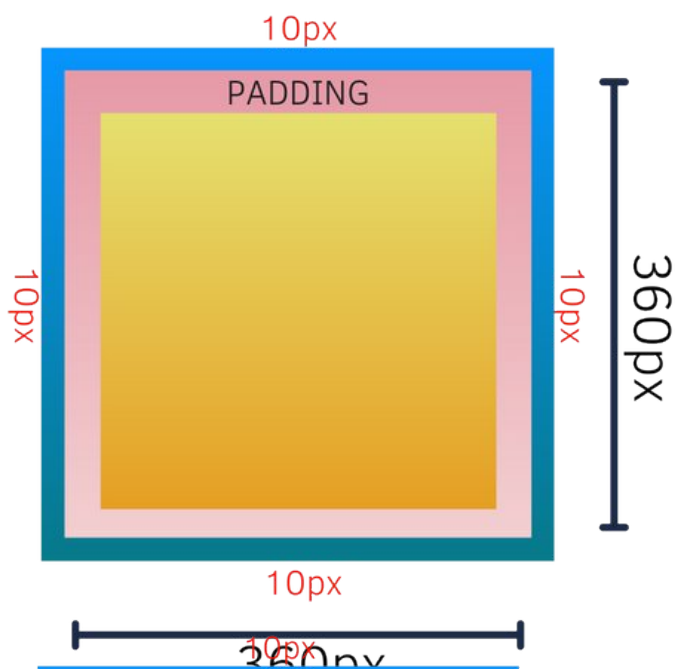


height: 300px;  
width: 300px;

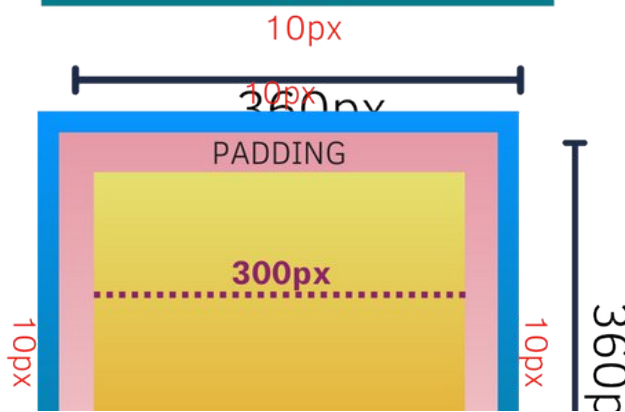


¿Qué pasa si agrego bordes?  
 ¿Medirá más la caja?  
 ¿Qué pasa si agrego padding?

Si agregamos bordes de 10px en todo su contorno, el contenedor pasará a medir 320px, sin importar si le hemos pedido que mida 300px.

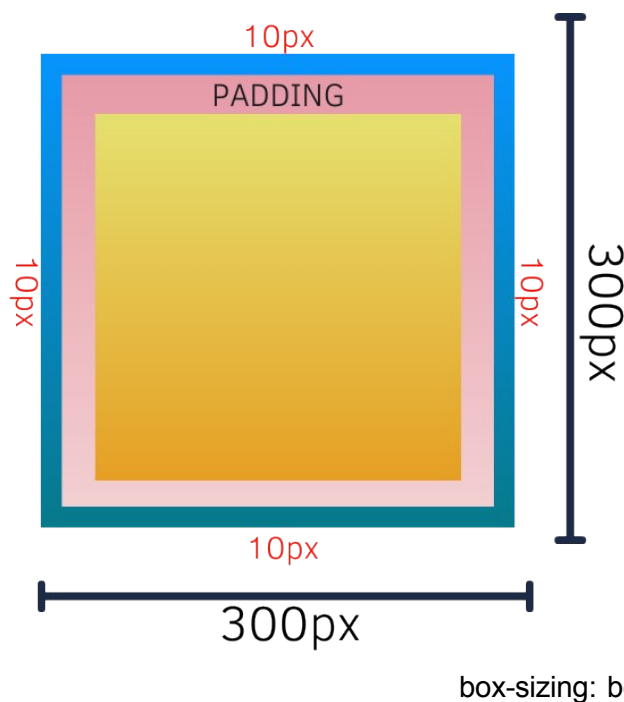


Si agregamos padding de 20px en todo su contorno, la caja pasa a medir 360px de ancho y alto, ya que se sigue sumando espacio, sin importar si se habían designado 300px en un principio.





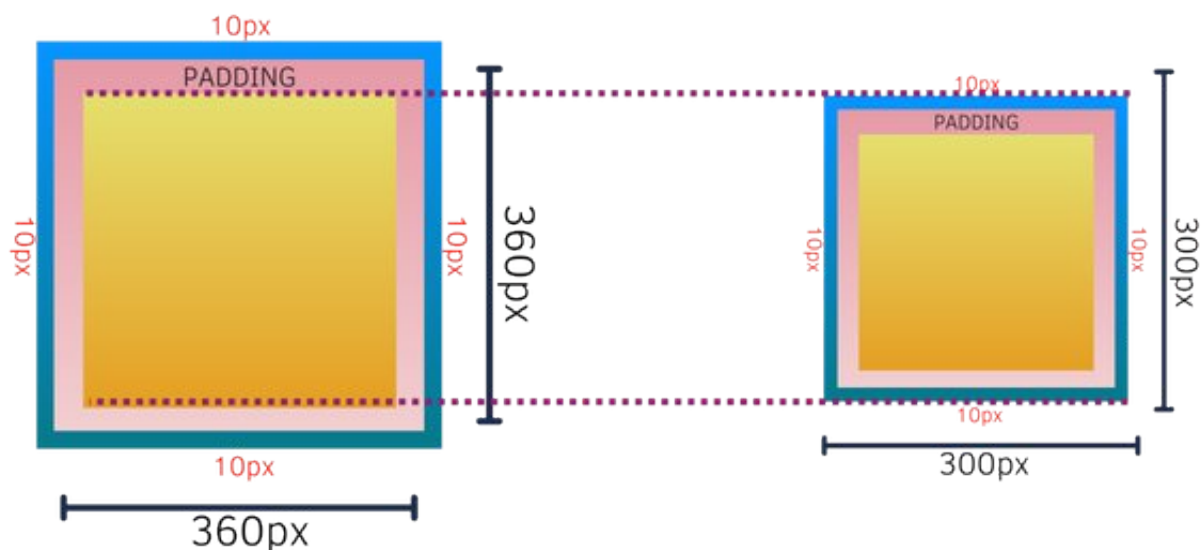
Esto pasa porque la prioridad box-sizing tiene por defecto el valor “content-box”, y cuando asignamos cierto ancho o alto, prioriza el contenido, no todo el contenedor en sí. Todo borde o padding que se agregue después, sumará a estas medidas.



Si cambiamos el valor de box-sizing por “border-box”, lo que va a suceder es que el ancho y alto que se asigne va a contemplar también el padding y el borde.

**Content vs Border**  
box-sizing: content-box;

box-sizing: border-box;



## Herencia y Especificidad en HTML

La herencia y la especificidad son dos conceptos fundamentales en **CSS** que afectan cómo se aplican los estilos a los elementos **HTML**. Vamos a profundizar en estos conceptos, considerando la relación entre componentes padres e hijos.

Si siguieron el ejemplo de la clase 3 paso a paso, podrán ver que aunque aplicamos la **'font-family'** solamente al elemento **'body'**, todos los elementos de la página cambiaron de fuente. ¿Por qué ocurre esto?

## Componentes Padres e Hijos

En el diseño web, los componentes **HTML** a menudo tienen una estructura jerárquica, donde un elemento padre contiene elementos hijos. Por ejemplo:

```

10 <body>
11   <header>
12     <h1>Aprendiendo herencia y especificidad</h1>
13   </header>
14
15   <div class="padre">
16     <p>Este es un párrafo dentro del div. hijo</p>
17     <p class="hijo">Este es otro párrafo hijo del div con la clase hijo</p>
18     <p id="hijoID" class="hijo">Este tercer párrafo también es hijo de div y tiene clase div. también tiene id hijoID</p>
19     <div>
20       <p>los elementos pueden tener <span>otros elementos hijos</span> dentro</p>
21     </div>
22   </div>
23
24 </body>

```

En este ejemplo, el div con la clase “padre” es el elemento padre, y los “p” son los elementos hijos dentro de él. También vemos que tiene otro div como elemento hijo que a su vez también tiene un “p” como hijo.

Esta jerarquía es esencial para entender cómo la herencia y la especificidad afectan la aplicación de estilos.

## Herencia en CSS

La herencia en **CSS** implica que ciertos estilos aplicados a un elemento padre se transmiten automáticamente a sus elementos hijos. Por eso cuando le aplicamos **font-family** al elemento “body”, la fuente se aplicó en todos los elementos de la página.

**Importante:** No todos los estilos se heredan. Por defecto, propiedades como color, font-family o font-size se heredan, mientras que propiedades como border o margin no.

**Es decir:** La herencia facilita la aplicación de estilos consistentes a través de una estructura **HTML** sin tener que declarar esos estilos para cada elemento individualmente.

## Especificidad en CSS

La especificidad determina qué regla **CSS** se aplicará cuando existan múltiples reglas para el mismo elemento. Se mide en términos de lo específica que es una regla. Cuanto más específica sea una regla, mayor prioridad tendrá.





Hay 3 tipos de selectores fundamentales: de **id**, **clase**, y **elemento**.

La especificidad se calcula asignando valores a los selectores. Por ejemplo, un selector de **id** tiene más peso que un selector de **clase**.

**Selector de id (#miElemento)** - Alta especificidad.

**Selector de clase (.miClase)** - Media especificidad.

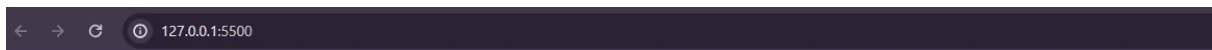
**Selector de elemento (p, div) - Baja especificidad.**

```
3  body {
4      font-family: 'Roboto', sans-serif;
5      /* Se aplica a todos los elementos hijos */
6  }
7
8
9  #padre{
10     color: blue
11 }
12 p{
13     color: red
14     /* el selector de elemento, tiene baja especificidad */
15 }
16 .hijo{
17     color: green
18     /* La class tiene una especificidad mayor, se aplicaran
19     los estilos a los elementos que tengan esta clase,
20     sobrescribiendo los estilos que sean del elemento. */
21 }
22 #hijoID{
23     /* si un elemento tiene class y tambien tiene ID
24     el ID tiene mayor especificidad, por lo tanto
25     serán los estilos del ID los que se apliquen al
26     elemento */
27     color: orange
28 }
29
30
```

En este caso, la regla para **#hijoID** prevalecerá, incluso si se aplica otra regla a un elemento “p” aunque el mismo tiene asignada una clase.

Lo mismo sucede para el selector de clase: Prevalecerán los estilos aplicados mediante este selector por sobre los que se asignen usando el selector de elementos.

Resultado de este código:



# Aprendiendo herencia y especificidad

Este es un párrafo dentro del div. hijo

Este es otro párrafo hijo del div con la clase hijo

Este tercer parrafo tambien es hijo de div y tiene clase div. también tiene id hijoID

los elementos pueden tener otros elementos hijos dentro

## Contenedores Límites

Cómo venimos viendo, todos los elementos de nuestra página están contenidos dentro de otros.

A estos elementos que contienen otros se los denomina comúnmente contenedores. Su tarea principal va a ser crear y delimitar áreas o secciones donde aparecerán sus elementos hijos y que luego formatearemos con **CSS**.

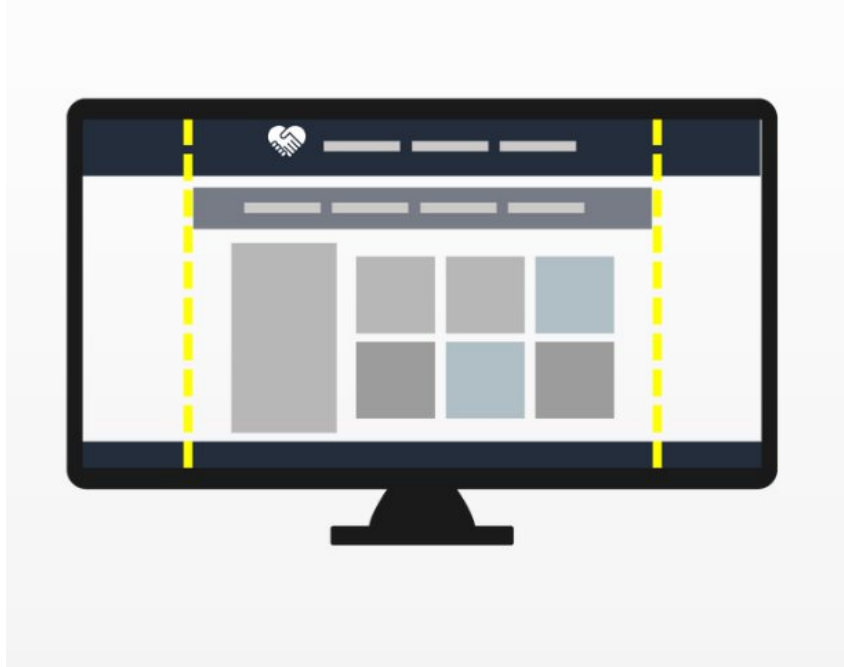
Para crear la estructura de nuestro sitio web, utilizamos contenedores semánticos. Estos contenedores tienen un propósito específico y reflejan la naturaleza del contenido que contienen.

**<header>**, **<nav>**, **<main>**, **<footer>**, **<section>**, **<aside>**, etc.

También podemos utilizar contenedores genéricos que nos sirven para agrupar elementos y organizar estructuras más complejas. el principal será **<div>**

Todos estos elementos son elementos de bloque, es decir que por defecto van a ocupar el 100% del ancho de la pantalla. Por lo general, no va a ser lo mejor que todos los elementos ocupen el 100% del ancho de la pantalla, sobre todo en el caso de pantallas muy grandes.

Por lo tanto, cuando visitemos un sitio en un dispositivo de pantallas muy grande, veremos que la mayoría de los sitios estarán “contenidos” dejando espacios libres en sus costados.



Cuando empezamos a armar nuestra web y comenzamos a estructurar nuestro sitio, suele pasar que todo ocupa el 100% del ancho de pantalla. Esto pasa porque no hay contenedores que contengan o limiten el contenido que estamos generando. Lo que tenemos que hacer es generar un contenedor que envuelva este contenido y lo limite con un ancho máximo.

Para este ejemplo usaremos un `<h1>` y un `<p>`.

## Proyecto

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Provident corrupti adipisci minima iste numquam voluptatibus incidunt molestiae dolor voluptatem fugiat, vitae cumque. Vitae facilis ipsa quod amet voluptatem, ex cupiditate.

Envolveremos los elementos `<h1>` y `<p>` en un contenedor `<div>` al cual le asignaremos como atributo la clase 'contenedor'

```
.contenedor{
  max-width: 70%; /* definimos el ancho del contenedor */
  margin: 0 auto; /* centramos el contenedor */
  border: 1px solid #ccc; /* aplicamos un borde para verlo*/
}
```

**Resultado:** ahora el contenedor limita el ancho de los elementos que contiene.



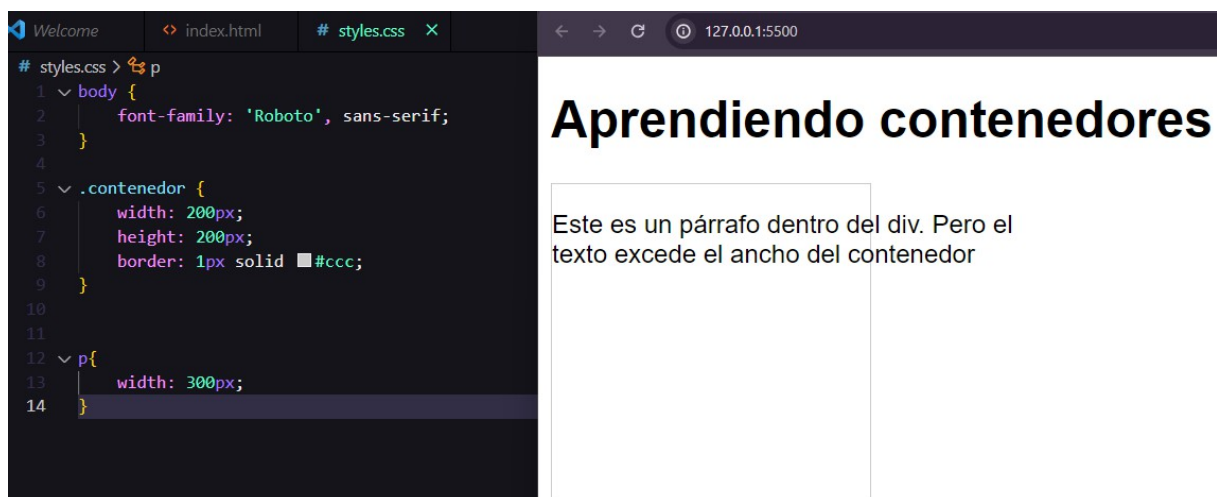
Podríamos encontrarnos con problemas con el del siguiente ejemplo:



```

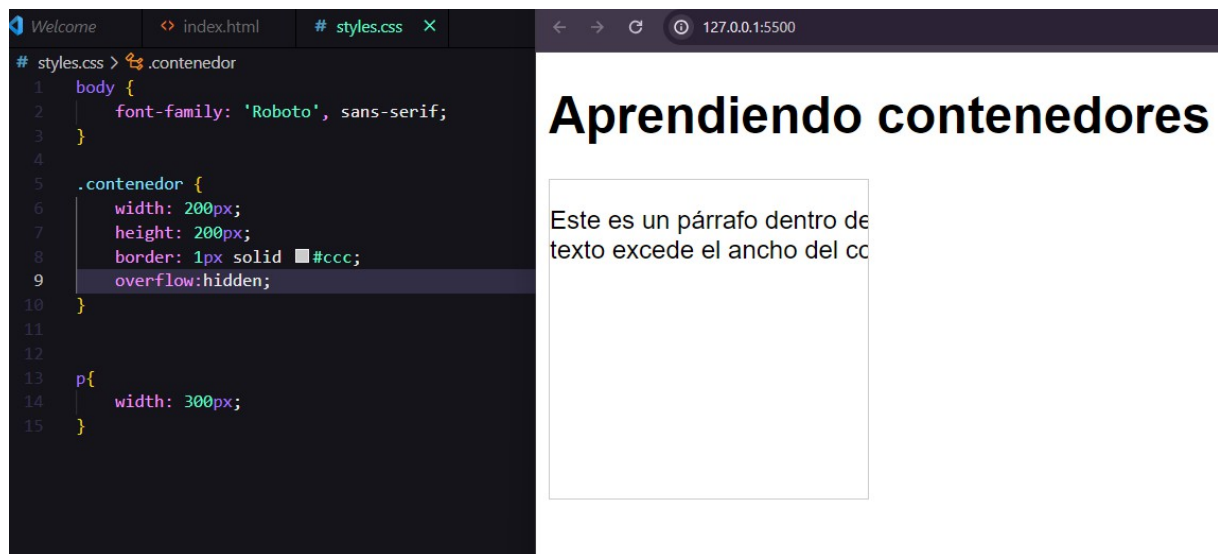
10
11 <body>
12   <header>
13     <h1>Aprendiendo contenedores</h1>
14   </header>
15
16   <div class="contenedor">
17     <p>Este es un párrafo dentro del div. Pero el texto excede el ancho del contenedor </p>
18   </div>
19
20 </body>

```

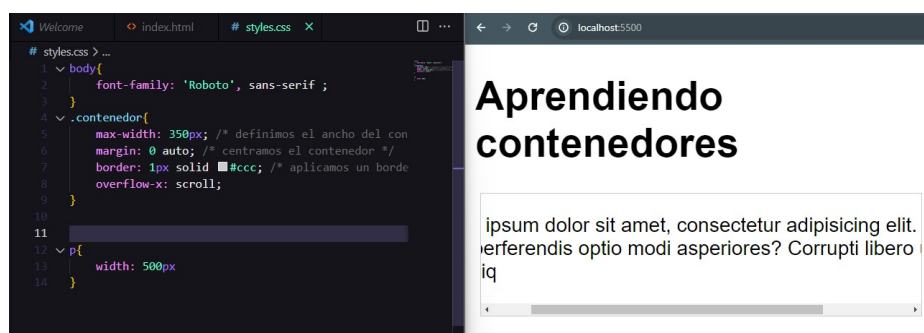


Una de las propiedades clave para controlar cómo se maneja el contenido que sobresale los límites de su contenedor es **'overflow'**. Veremos 2 ejemplos sencillos de cómo afecta el contenido de un contenedor ésta propiedad: **'hidden'** y **'scroll'**, pero hay muchos más. En este caso, aplicamos la propiedad **overflow hidden**.

Por lo tanto el contenido que exceda el ancho del contenedor, quedará “oculto” o recortado por el borde del contenedor.



Veamos qué sucede cuando utilizamos la propiedad **overflow hidden**. En este caso, tendremos una barra de desplazamiento que nos permitirá ver todo el contenido.



# Propiedad Position

La propiedad de css position va a definir cómo se posiciona un elemento dentro del documento en relación con los otros elementos de la página. Hasta ahora hemos visto cómo los elementos se acomodan uno debajo del otro a medida que los vamos agregando en nuestro **HTML**. El posicionamiento nos va a permitir sacar elementos del flujo normal del documento y hacer que se comporten de manera diferente, por ejemplo, colocándolos uno encima de otro o manteniéndolos siempre en el mismo lugar dentro de la ventana del navegador.

Veremos los diferentes valores de posición y cómo usarlos para lograr diseños y layouts mucho más interesantes:

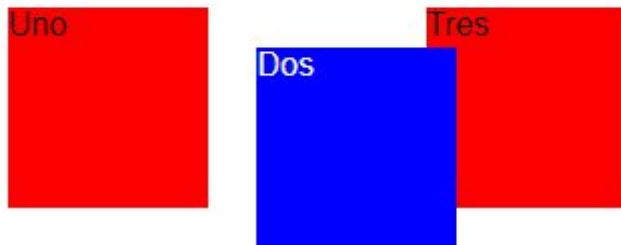
## Static, Relative y Absolute

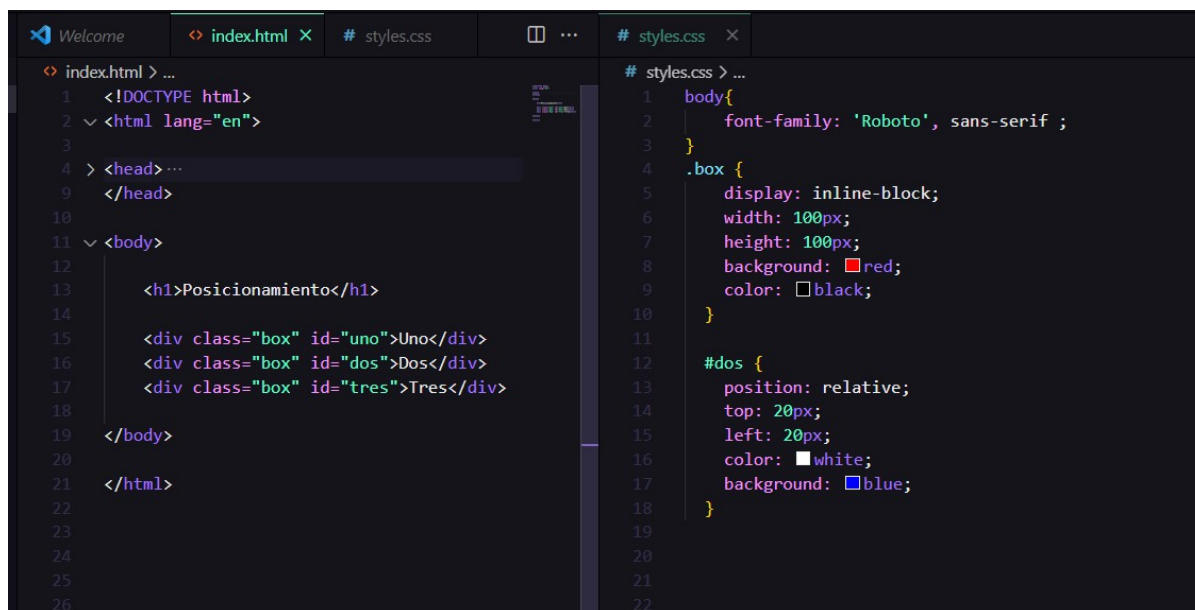
**Static:** es el valor por defecto. Los elementos se posicionarán según el flujo normal del documento. (no se verán afectados por las propiedades top, right, left y bottom).

**Relative:** Este valor permite mover el elemento desde su posición normal en el flujo del documento, pero aún ocupará espacio en el diseño como si estuviera en su ubicación original.

← → ↻ ⓘ 127.0.0.1:5500/index.html

# Posicionamiento



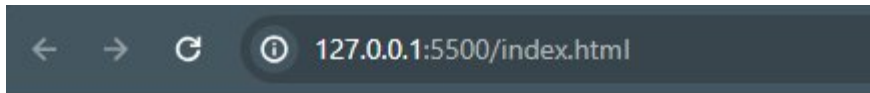
A screenshot of a code editor with two tabs: 'index.html' and '# styles.css'. The 'index.html' tab is active, showing a basic HTML structure with a head and body. The body contains an h1 element with the text 'Posicionamiento' and three div elements with class 'box' and IDs 'uno', 'dos', and 'tres'. The '# styles.css' tab is also visible, showing CSS rules for the body and the boxes. The body rule sets the font-family to 'Roboto'. The '.box' rule sets display to inline-block, width to 100px, height to 100px, background to red, and color to black. The '#dos' rule sets position to relative, top to 20px, left to 20px, color to white, and background to blue.

```
index.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3
4 > <head>...
9 </head>
10
11 <body>
12
13 <h1>Posicionamiento</h1>
14
15 <div class="box" id="uno">Uno</div>
16 <div class="box" id="dos">Dos</div>
17 <div class="box" id="tres">Tres</div>
18
19 </body>
20
21 </html>
22
23
24
25
26

# styles.css > ...
1 body{
2   font-family: 'Roboto', sans-serif ;
3 }
4 .box {
5   display: inline-block;
6   width: 100px;
7   height: 100px;
8   background: red;
9   color: black;
10 }
11
12 #dos {
13   position: relative;
14   top: 20px;
15   left: 20px;
16   color: white;
17   background: blue;
18 }
19
20
21
22
```

En este ejemplo, la caja “#dos” tiene posicionamiento relativo, lo que significa que pudimos ajustar su posición usando las propiedades **top y left**. haciendo que se desplace **20px** para cada lado. Este tipo de posicionamiento es útil cuando deseas realizar ajustes finos sin afectar el diseño general de la página.

**Absolute:** los elementos que tengan **position absolute** se posicionan en relación con el primer elemento padre que tenga una posición distinta a static. Si no hay ningún contenedor con posición distinta a **static**, se toma como referencia el elemento ‘**html**’.



# Posicionamiento



```
index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3
4  > <head> ...
9  </head>
10
11 <body>
12   <h1>Posicionamiento</h1>
13
14   <div class="box" id="uno">Uno</div>
15   <div class="box" id="dos">Dos</div>
16   <div class="box" id="tres">Tres</div>
17
18 </body>
19 </html>
20
21
22
23
24
25
26
27

# styles.css > ...
1  body{
2    font-family: 'Roboto', sans-serif ;
3  }
4  .box {
5    display: inline-block;
6    width: 100px;
7    height: 100px;
8    background: red;
9    color: black;
10 }
11
12 #dos {
13   position: absolute;
14   top: 50px;
15   left: 30px;
16   background: transparent;
17   border: 2px solid blue;
18 }
19
20
21
22
23
```

En este ejemplo, la caja “#dos” tiene posicionamiento absoluto y no tiene ningún contenedor padre por lo que toma como referencia el elemento html, Si no aplicamos top ni left, nuestra caja ‘#dos’ aparecerá en el extremo superior izquierdo de la página. Usamos top y left para moverla desde ese punto. A tener en cuenta: al usar este posicionamiento la caja ya no ocupa lugar en el flujo normal del documento.

## Fixed y Sticky

**Fixed:** al aplicar esta propiedad, el elemento se situará en una posición fija en relación a la ventana del navegador. Este tipo de posicionamiento hace que el elemento permanezca en la misma posición incluso cuando scrolleamos hacia abajo en la página.

**Sticky:** El posicionamiento **sticky** combina características de **relative y fixed**. El elemento se posiciona según el flujo normal del documento hasta que alcanza una determinada posición durante el desplazamiento de la página. En ese punto, se comportará como un elemento **fixed**.

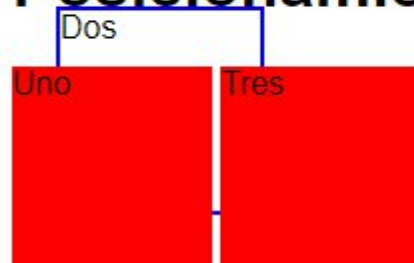
## Propiedad Z-index:

Cuando empezamos a jugar con el posicionamiento de los elementos de nuestra página es normal que varios de ellos queden superpuestos. para controlar cuál va a quedar “por delante” o “por detrás” disponemos de la propiedad **z-index** en **CSS**. Un valor mayor de **z-index** coloca un elemento encima de otro con un valor menor. (el valor por defecto

es 0). Veámoslo en el código:



# Posicionamiento





```
# styles.css ×
# styles.css > ...
1  body{
2      font-family: 'Roboto', sans-serif ;
3  }
4  .box {
5      display: inline-block;
6      width: 100px;
7      height: 100px;
8      background: red;
9      color: black;
10 }
11
12 #dos {
13     position: absolute;
14     top: 50px;
15     left: 30px;
16     background: transparent;
17     border: 2px solid blue;
18     z-index: -1;
19 }
20
21
22
23
```

Al aplicarle **z-index: -1** a la caja dos, ahora aparece por detrás de los otros elementos de la página.

- Ingresa en el siguiente enlace y analizá el código del ejemplo propuesto.

<https://stackblitz.com/edit/web-platform-xnsbjv?file=index.html>

Aprovecha para experimentar dentro de ese ejemplo, agregale colores, jugá con el posicionamiento de los elementos, modificá los márgenes y padding para comprender el uso de cada propiedad y atributo antes de aplicarlo en tu proyecto.



## Desafío N° 4:

Aplica estilos **CSS** a tu Página **HTML**:

- **Aplicar estilos CSS:** agrega estilos a tu página **HTML** para mejorar su apariencia y presentación visual.
- Experimenta con propiedades de estilo como **font-family**, **color**, **background-color**, **border**, **border-radius**, etc., para darle vida a tu página.
- **Utilizar colores:** selecciona una paleta de colores que complementen la temática de tu página y proporcionen una experiencia agradable para los usuarios.
- **Ajustar márgenes y padding:** mejora el espaciado y la distribución de elementos en tu diseño.
- Experimentá con diferentes valores para lograr un diseño equilibrado.

**Recuerden:** no usar margin y padding para distribuir elementos.

Sigue las instrucciones dadas en las clases anteriores y asegúrate de tener una estructura **HTML** sólida y significativa.

### Subí tu trabajo:

Comprimí la carpeta de tu proyecto con todos los documentos (HTML, CSS e imágenes) en un archivo .zip y subilo a la plataforma.





**Buenos Aires**  
*aprende*  
Agencia de Actividades para el Futuro

**BA** Buenos  
Aires  
Ciudad