Block vs. Inline

Concepto general:

Que en CSS (y HTML), los elementos se dividen en dos partes principales para la visualización.

Block(bloque) y Inline (en línea)

El block es el que ocupa todo el ancho disponible del contendor y siempre comienzan en una nueva línea. Un ejemplo de ello sería las etiquetas como <p>, <h1> a <h6>, entre otras.

El Inline es el que solo ocupa espacio necesario según su contenido, sin romper la línea. Algunos ejemplos típicos serían las etiquetas , <a>, entre otras.

Block:

Conocido como elemento de bloque es el que ocupa todo el ancho posible y empuja el siguiente elemento hacia abajo. Se podría decir que “Se comporta como una caja completa”.

Ventajas:

- Permite estructurar el diseño (secciones, encabezados, contenedores)
- Acepta propiedades de tamaño fácilmente como (width, height, margin ,padding)
- Fácil de alinear y distribuir el layout.

Desventajas:

- No se puede colocar de manera natural en la misma línea sin usar técnicas adicionales como “display: inline-block o flex”.
- Puede generar más espacio vertical del necesario si no se llegan a ajustar bien los márgenes o alturas.

Caso de Uso:

- Contenedores principales (, <section>)
- Artículo o bloques texto(<p>, <article>)
- Encabezados (<h1>-<h6>)

Comparativa rápida

Característica	`display: block`	`display: inline`
Ocupa todo el ancho disponible	✅ Sí	❌ No

Comienza en nueva línea	✅ Sí	❌ No
Acepta `width` y `height`	✅ Sí	❌ No
Ideal para contenedores grandes	✅	❌
Ideal para resaltar texto o enlaces	❌	✅

Ejemplo (código):

HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Block vs Inline</title>

  <!-- Vincula el archivo externo de estilos CSS -->
  <link rel="stylesheet" href="estilos.css">
</head>

<body>

  <!-- Título principal de la página -->
  <h2> Elementos Block vs Inline</h2>

  <!-- ================ SECCIÓN: ELEMENTOS DE BLOQUE ================ -->
  <section>
    <h3>⬜ Elementos tipo <code>block(bloque)</code></h3>


    <!-- Cada uno de estos elementos usa la clase .bloque -->
    <div class="bloque">Bloque "1"</div>
    <div class="bloque">Bloque "2"</div>
    <p class="bloque">Es un párrafo(etiqueta "p") en bloque</p>

    <!-- Texto explicativo -->
    <p>Como se ve cuando uno comienza en una nueva línea ⬇</p>
  </section>

  <!-- ================ SECCIÓN: ELEMENTOS EN LÍNEA ================ -->
  <section>
    <h3>↦ Elementos tipo <code>inline(en linea)</code></h3>

    <!-- Ejemplo con elementos en línea dentro de un párrafo -->
    <p>
      Este texto contiene diferentes palbras
      <span class="en-linea">resaltadas</span>
      <span class="en-linea">en línea</span>
```

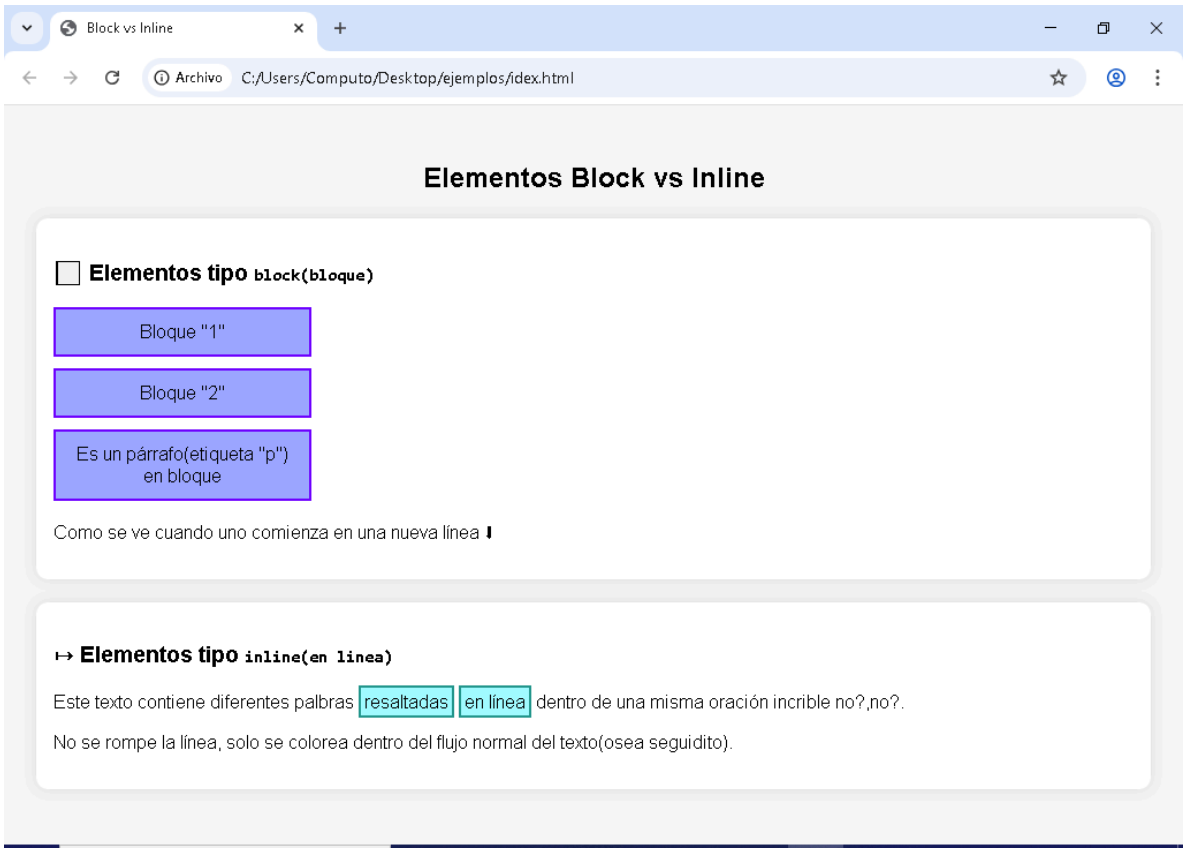
```
      dentro de una misma oración incrible no?,no?.
    </p>

    <!-- Texto explicativo -->
    <p>No se rompe la línea, solo se colorea dentro del flujo normal del
texto(osea seguidito).</p>
  </section>

</body>
</html>
```

CSS(codigo):

```
/* ======== ESTILOS GENERALES ======== */
body {
    font-family: Arial, sans-serif;   /* Fuente sencilla */
    background-color: #f5f5f5;        /* Fondo gris calro para mas comodidad
*/
    padding: 20px;                    /* Espaciado interior general */
  }

  /* Estilos de h2*/
  h2 {
    text-align: center;
    color: black;
  }

  /* Caja blanca para separar visualmente cada bloque de ejemplo y sea mas
etendible(las trajetas)*/
  section {
    background-color: white;
    border-radius: 10px;
    padding: 15px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    margin-bottom: 20px;
  }

  /* ======== ELEMENTOS DE BLOQUE ======== */
  /* Ocupan todo el ancho disponible y comienzan en nueva línea */
  .bloque {
    display: block;
    background-color: #9fa8ff;
    border: 2px solid #6f00ff;
    color: #000000;
    text-align: center;
    margin: 10px 0;
    padding: 10px;
```

```
  width: 200px;
}

/* ======== ELEMENTOS EN LÍNEA ======== */
/* Se mantienen dentro del flujo del texto sin romper la línea */
.en-linea {
  display: inline;
  background-color: #a1fcff;
  border: 2px solid #249690;
  color: #000000;
  padding: 3px;
}
```

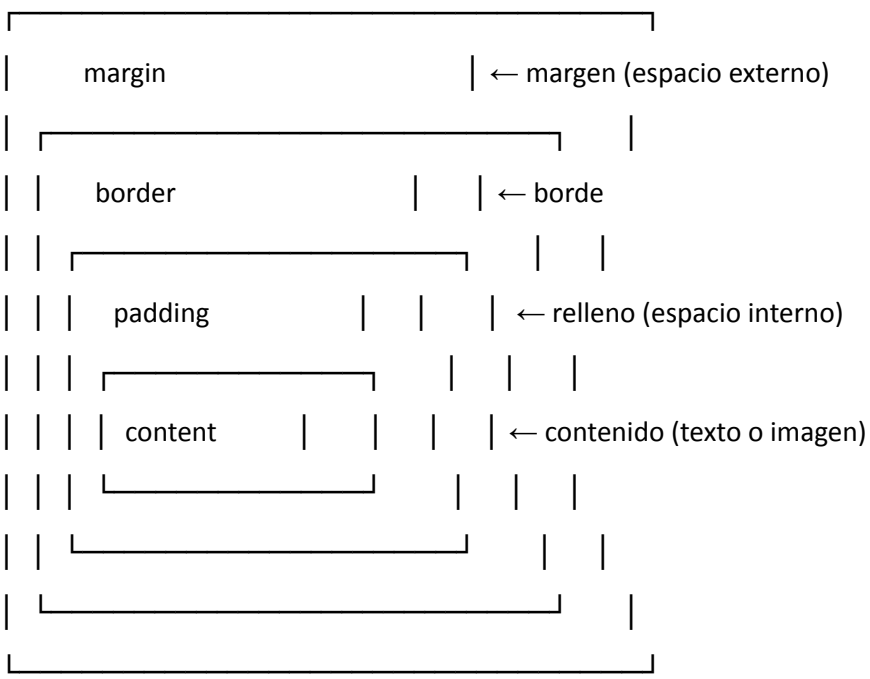
Ejemplo(visual):



Box Model (modelo de caja):

¿Qué es?

El **Box Model (modelo de caja)** define **cómo se calculan el tamaño, los márgenes y el espacio** que ocupa cada elemento en una página web.
Cada caja tiene **cuatro partes**:



Ventajas:

- Permite controlar el espacio y tamaño exacto de los elementos.
- Facilita alinear y distribuir contenido visualmente.
- Es esencial para crear diseños coherentes y limpios.

Desventajas:

- Puede ser confuso al principio, sobre todo al calcular anchos totales.

- Si no se usa bien, puede romper el diseño (por ejemplo, si el padding o border suman más del ancho esperado).

Casos de uso:

- Crear tarjetas, botones o secciones con espacio internos y externos.
- Ajusta la separación entre elementos.
- Controla cómo se expanden o alinean los bloques.

Ejemplo(código):

HTML:

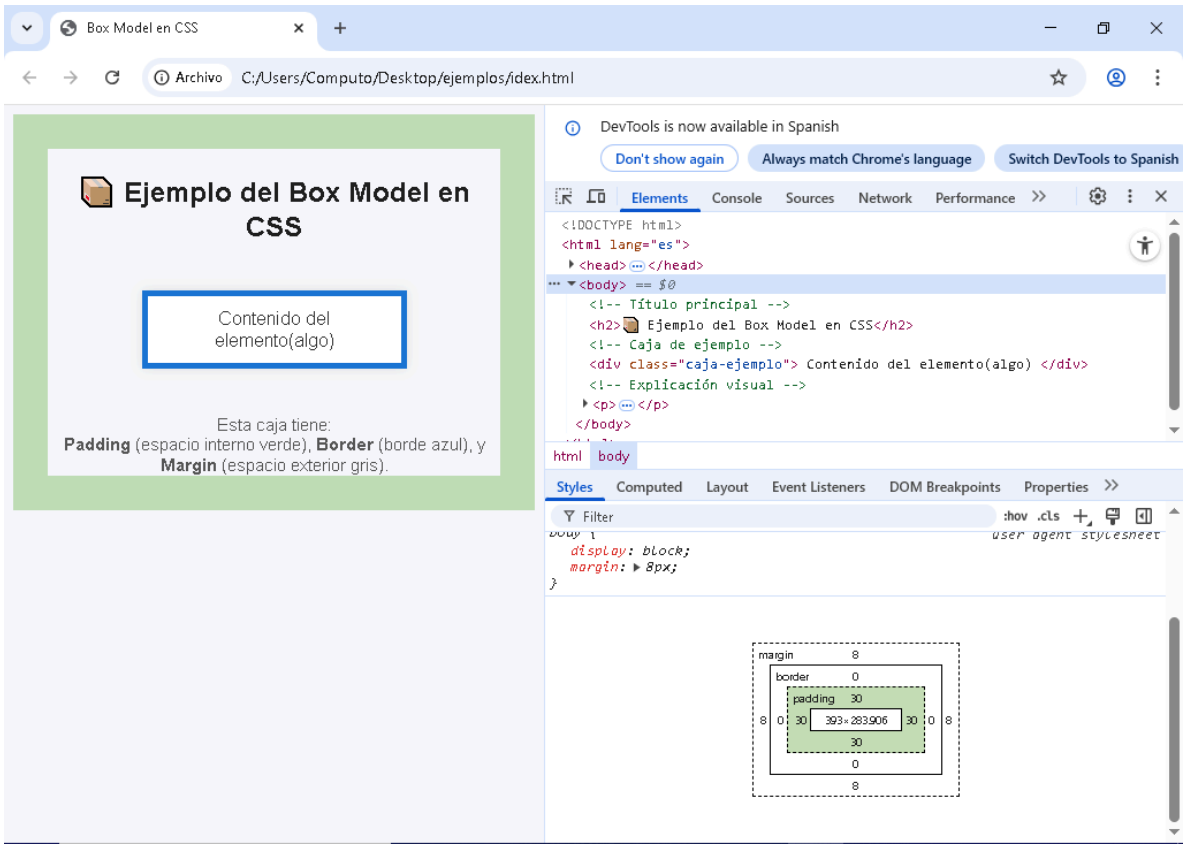
```html
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Box Model en CSS</title>
  <!-- Enlace al archivo CSS -->
  <link rel="stylesheet" href="estilos.css">
</head>
<body>

  <!-- Título principal -->
  <h2>📦 Ejemplo del Box Model en CSS</h2>

  <!-- Caja de ejemplo -->
  <div class="caja-ejemplo">
    Contenido del elemento(algo)
  </div>

  <!-- Explicación visual -->
  <p>
    Esta caja tiene: <br>
    <strong>Padding</strong> (espacio interno verde),
    <strong>Border</strong> (borde azul), y
    <strong>Margin</strong> (espacio exterior gris).
  </p>

</body>
</html>
```

CSS:

```css
/* ======== ESTILOS GENERALES ======== */
body {
    font-family: Arial, sans-serif;
    background-color: #f8f9fa;
    padding: 30px;
```

```
    text-align: center;
  }

  h2 {
    color: #222;
  }

  /* ======== CAJA DE EJEMPLO ======== */
  /* Demuestra visualmente las partes del Box Model */
  .caja-ejemplo {
    background-color: #fff;
    color: #333;
    padding: 10px;                    /* Espacio interno (verde en el
esquema mental) */
    border: 5px solid #1976d2;        /* Borde */
    margin: 40px auto;                /* Espacio externo y centrado */
    width: 200px;                     /* Ancho del contenido */
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
  }

  /* Estilo visual del texto <p>  */
  p {
    color: #555;
    font-size: 15px;
    max-width: 400px;
    margin: 0 auto;
  }
```

Ejemplo (visual):



Box Sizing:

Es el que por defecto, el navegador suma el padding y el border al ancho y alto del elemento, lo que puede causar que el diseño se rompa si no se toma en cuenta..

La propiedad box-sizing te permiten cambiar la forma en que el navegador calcula el tamaño total de una caja.

Valor	Descripción	Ejemplo visual
`content-box` *(por defecto)*	El ancho y alto **solo incluyen el contenido**. El `padding` y `border` se agregan **por fuera**, haciendo la caja más grande.	Más difícil de controlar el tamaño total.
`border-box`	El ancho y alto **incluyen contenido, padding y border** dentro del mismo tamaño total.	Mucho más predecible y usado en diseños modernos.
`content-box` *(por defecto)*	El ancho y alto **solo incluyen el contenido**. El `padding` y `border` se agregan **por fuera**, haciendo la caja más grande.	Más difícil de controlar el tamaño total.
`border-box`	El ancho y alto **incluyen contenido, padding y border** dentro del mismo tamaño total.	Mucho más predecible y usado en diseños modernos.

Ventajas de border-box

- Facilita el control del tamaño total del elemento.
- Hace más sencillo trabajar con diseños responsivos.
- Evita errores al sumar manualmente bordes y rellenos.

Desventajas

- Si trabaja con librerías antiguas que usan content-box, los tamaños pueden no coincidir.
- Puede ser confuso si no se sabe cómo se calcula el espacio interno.

Casos de uso:

- El diseño de tarjetas, columnas, formularios y layouts responsivos.
- Cuando necesitas mantener un ancho exacto sin importar el borde o el relleno.

Ejemplo(código):

HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Box Sizing en CSS</title>
  <!-- Enlace al archivo CSS -->
  <link rel="stylesheet" href="estilos.css">
</head>
<body>


  <h2> Ejemplo de <code>box-sizing</code></h2>


  <!-- Contenedor principal para comparar ambos tipos -->
  <section class="contenedor">
    <!-- Caja con box-sizing: content-box -->
    <div class="caja content-box">
      content-box <br>
      (ancho total = contenido + padding + borde)
    </div>
```

```
    <!-- Caja con box-sizing: border-box -->
    <div class="caja border-box">
      border-box <br>
      (ancho total = incluye padding y borde)
    </div>
  </section>

  <p>Ambas cajas tienen el mismo ancho definido (200px), pero la de abajo no crece porque usa <code>border-box</code>,inreible no?,no?.</p>

</body>
</html>
```

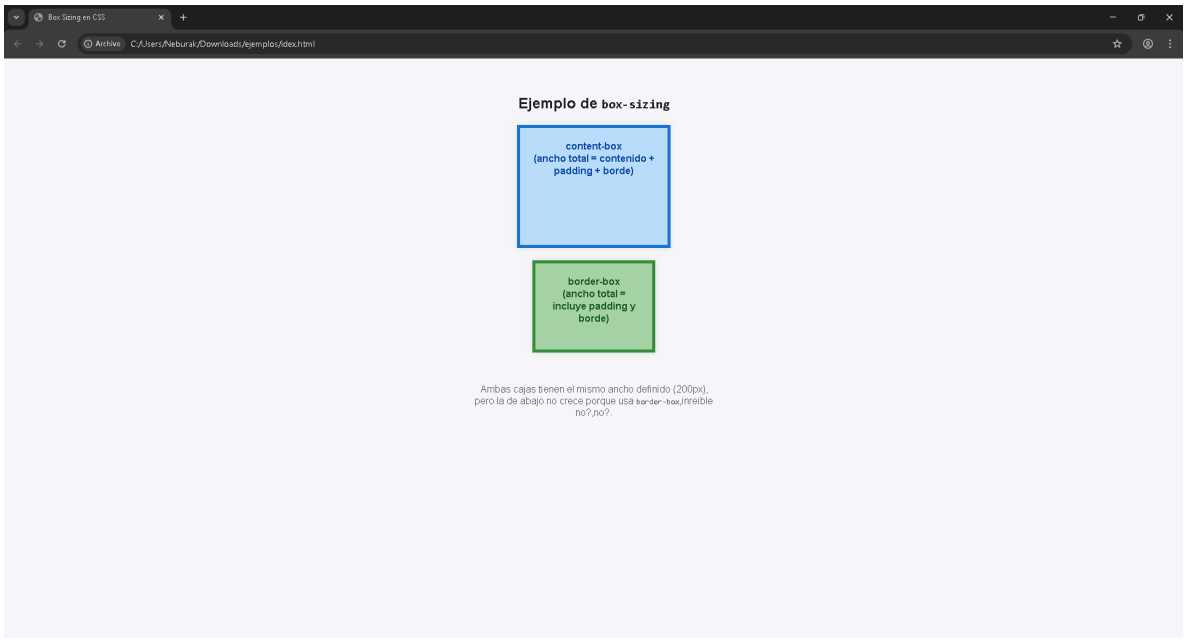
CSS:

```
/* ======== ESTILOS GENERALES ======== */
body {
    font-family: Arial, sans-serif;
    background-color: #f8f9fa;
    padding: 30px;
    text-align: center;
  }

  h2 {
    color: #222;
  }
```

```css
/* Contenedor para centrar ambas cajas */
.contenedor {
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 20px; /* Espacio entre cajas */
    margin-top: 20px;
}

/* ======== CAJAS DE EJEMPLO ======== */
.caja {
    width: 200px;             /* Ancho definido */
    height: 150px;            /* Alto fijo */
    background-color: #bbdefb;
    border: 5px solid #1976d2;
    padding: 20px;
    color: #0d47a1;
    text-align: center;
    font-weight: bold;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
}

/* Por defecto, el tamaño real será mayor porque suma padding y borde */
.content-box {
    box-sizing: content-box;
```

```
}


/* border-box incluye padding y el borde dentro del tamaño total */
.border-box {
    box-sizing: border-box;
    background-color: #a5d6a7;
    border-color: #388e3c;
    color: #1b5e20;
}


/* Texto  */
p {
    max-width: 400px;
    margin: 50px auto;
    color: #555;
}
```

Ejemplo(visual):



Position

¿Qué es position en CSS?

La propiedad position define cómo un elemento se coloca dentro del flujo de la página. Permite controlar su ubicación exacta en relación con otros elementos o con el contenedor que lo contiene.

Valor	Descripción	Ejemplo típico
static	Posición por defecto; el elemento sigue el flujo normal del documento.	Texto y párrafos normales.
relative	Se mueve respecto a su posición original sin sacarlo del flujo.	Ajustes pequeños o efectos visuales.
absolute	Se coloca en relación al contenedor posicionado más cercano (no al viewport).	Etiquetas, íconos o superposiciones.

fixed	Se mantiene fijo en la pantalla, incluso al hacer scroll.	Barras de navegación o botones flotantes.
sticky	Actúa como relative hasta cierto punto, y luego se fija al hacer scroll.	Encabezados o menús que se pegan arriba.

Ventajas

- Permite control preciso del diseño.
- Facilita efectos visuales y overlays.
- Ideal para elementos que deben permanecer visibles (navbars, popups, etc.).

Desventajas

- Un mal uso puede romper el flujo del diseño.
- absolute y fixed pueden superponerse a otros elementos.
- Puede requerir más cálculos de márgenes y z-index.

Casos de uso comunes:

- Crear un menú fijo.
- Posicionar íconos sobre imágenes.
- Mostrar mensajes flotantes o botones.
- Fijar encabezados con sticky.

Ejemplo(código):

- static: permanece en orden normal.
- relative: se mueve 20px hacia abajo y a la derecha, pero deja su espacio original.
- absolute: se coloca dentro del contenedor en la esquina superior derecha.
- fixed: flota en la esquina inferior derecha incluso al hacer scroll.
- sticky: se mantiene normal hasta que llega al borde superior, donde se “pega”.

HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Propiedad Position en CSS</title>
  <link rel="stylesheet" href="estilos.css">
</head>
<body>


  <h2>Ejemplo de <code>position</code> en CSS</h2>


  <section class="contenedor">
    <div class="caja static">static</div>
    <div class="caja relative">relative</div>
    <div class="caja absolute">absolute</div>
    <div class="caja fixed">fixed</div>
    <div class="caja sticky">sticky</div>
  </section>
```

```
  <p class="texto">Desplarse hacia abajo para ver cómo el elemento
<b>fixed</b> permanece visible y cómo el <b>sticky</b> se fija al llegar
arriba.(tara-no es burjeria en programacion)</p>

  <div class="espaciador"></div> <!-- Solo para poder hacer scroll -->

</body>
</html>
```

CSS:

```
/* ======== ESTILOS GENERALES ======== */
body {
    font-family: Arial, sans-serif;
    background-color: #f0f4f8;
    margin: 0;
    padding: 20px;
    height: 200vh; /* Altura grande para scroll */
  }

  h2 {
    text-align: center;
    color: #222;
  }
```

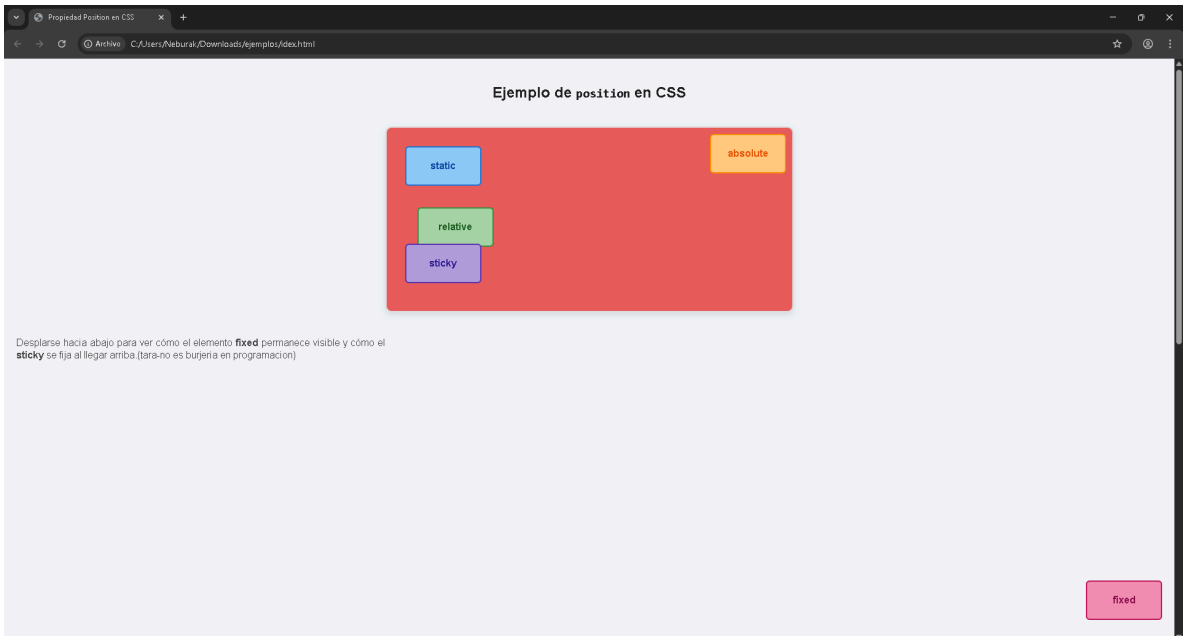
```css
/* ======== CONTENEDOR ======== */

.contenedor {

    position: relative; /* absolute se base en este */

    width: 90%;

    max-width: 600px;

    margin: 40px auto;

    background-color: #e75c5c;

    border: 2px solid #ddd;

    padding: 30px;

    border-radius: 10px;

    box-shadow: 0 4px 10px rgba(0,0,0,0.1);

}



/* ======== CAJAS ======== */

.caja {

    width: 120px;

    height: 60px;

    background-color: #90caf9;

    color: #0d47a1;

    text-align: center;

    line-height: 60px;

    border-radius: 6px;

    margin-bottom: 15px;

    font-weight: bold;

    border: 2px solid #1976d2;

}
```

```css
/* Static: sigue el flujo normal */
.static {
  position: static;
}

/* Relative: se mueve respecto a su posición original */
.relative {
  position: relative;
  top: 20px;
  left: 20px;
  background-color: #a5d6a7;
  border-color: #388e3c;
  color: #1b5e20;
}

/* Absolute: se posiciona respecto al contenedor */
.absolute {
  position: absolute;
  top: 10px;
  right: 10px;
  background-color: #ffcc80;
  border-color: #fb8c00;
  color: #e65100;
}
```

```css
/* Fixed: siempre visible al hacer scroll */
.fixed {
    position: fixed;
    bottom: 20px;
    right: 20px;
    background-color: #f48fb1;
    border-color: #c2185b;
    color: #880e4f;
}

/* Sticky: se queda fijo al hacer scroll */
.sticky {
    position: sticky;
    top: 0;
    background-color: #b39ddb;
    border-color: #5e35b1;
    color: #311b92;
}

/* Texto adicional */
.texto {
    margin-top: 40px;
    max-width: 600px;
    text-align: justify;
    color: #444;
}
```

```css
/* Espaciador para hacer scroll */
.espaciador {
    height: 1000px;
}
```

Ejemplo (visual):



Stacking Contexts:

En CSS, un **Stacking Context** (o **contexto de apilamiento**) determina **el orden en que los elementos se superponen** visualmente en una página.
Es como una “caja de capas” donde los elementos **se apilan unos sobre otros** según sus propiedades de posición, opacidad o `z-index`.

Cómo se crea un Stacking Context

Un nuevo contexto de apilamiento se crea cuando un elemento tiene alguna de las siguientes condiciones:

Propiedad / Condición	Ejemplo
`position` ≠ static y con `z-index` definido	`position: relative; z-index: 1;`
`opacity` < 1	`opacity: 0.9;`
`transform` aplicado	`transform: scale(1);`
`filter`, `perspective`, `clip-path`, etc.	`filter: blur(0);`
Elemento raíz del documento (`<html>`)	Siempre crea uno.

Ventajas

- Controla de forma precisa qué elementos se muestran “encima” de otros.
- Evita que `z-index` de un elemento afecte a otros fuera de su contexto.

- Facilita la organización visual en diseños complejos (menús, modales, overlays).

Desventajas

- Si no se entiende bien, puede parecer que `z-index` “no funciona”.
- Puede generar confusión cuando hay muchos contextos anidados.

Casos de uso

- Mostrar correctamente ventanas modales, tooltips o dropdowns.
- Asegurar que un overlay no quede por debajo de su fondo.
- Crear jerarquías visuales bien definidas.

Ejemplo(código):

HTML:

```html
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Stacking Contexts en CSS</title>
  <link rel="stylesheet" href="estilos.css">
</head>
<body>


  <h2>Ejemplo: Stacking Contexts (Contextos de apilamiento)</h2>
```

```
  <section class="contenedor">
    <!-- Caja base sin contexto especial -->
    <div class="caja caja1">Caja 1 (sin contexto)</div>

    <!-- Caja que crea un nuevo contexto de apilamiento -->
    <div class="caja caja2">
      Caja 2 (nuevo contexto)
      <div class="hija">Hija con z-index alto</div>
    </div>

    <!-- Caja que intenta colocarse encima -->
    <div class="caja caja3">Caja 3 (z-index mayor fuera)</div>
  </section>

</body>
</html>
```

CSS:

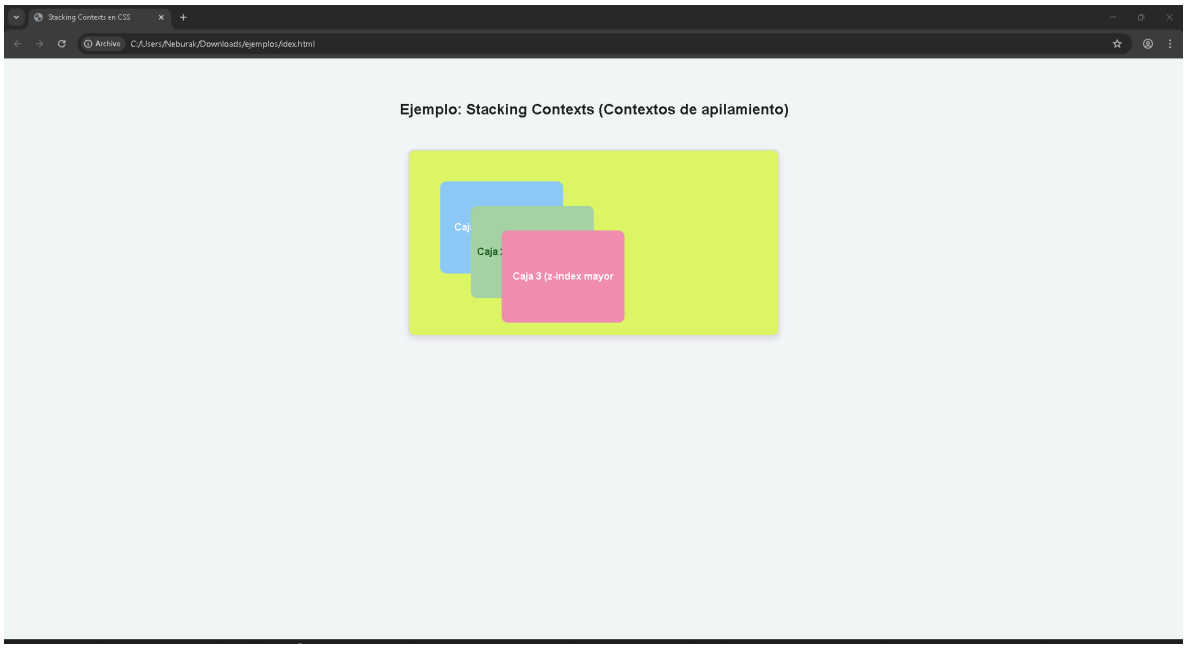
```
/* ===== ESTILOS GENERALES ===== */
body {
    font-family: Arial, sans-serif;
    background-color: #f4f6f8;
    padding: 40px;
  }
```

```
h2 {
    text-align: center;
    color: #222;
}

/* ===== CONTENEDOR ===== */
.contenedor {
    position: relative;
    width: 90%;
    max-width: 600px;
    height: 300px;
    margin: 50px auto;
    background-color: #dcf766;
    border: 2px solid #ddd;
    border-radius: 10px;
    overflow: hidden;
    box-shadow: 0 4px 10px rgba(0,0,0,0.1);
}

/* ===== CAJAS ===== */
.caja {
    position: absolute;
    width: 200px;
    height: 150px;
    color: white;
```

```
    font-weight: bold;
    text-align: center;
    line-height: 150px;
    border-radius: 10px;
  }

  /* Caja 1 - No crea nuevo stacking context */
  .caja1 {
    background-color: #90caf9;
    top: 50px;
    left: 50px;
    z-index: 1;
  }

  /* Caja 2 - Crea un nuevo contexto (por transform y z-index) */
  .caja2 {
    background-color: #a5d6a7;
    top: 90px;
    left: 100px;
    transform: scale(1); /* Esto crea un nuevo contexto */
    z-index: 2;
    color: #1b5e20;
  }

  /* Elemento hijo dentro de la caja 2 */
  .caja2 .hija {
```

```
    position: absolute;
    top: 40px;
    left: 60px;
    width: 100px;
    height: 60px;
    background-color: #2e7d32;
    z-index: 999; /* No sale de su contexto */
    line-height: 60px;
    border-radius: 6px;
  }


  /* Caja 3 - Está fuera, pero debajo visualmente */
  .caja3 {
    background-color: #f48fb1;
    top: 130px;
    left: 150px;
    z-index: 3;
  }


  /* Efecto visual */
  .contenedor .caja:hover {
    opacity: 0.9;
  }
```

Ejemplo(visual):



Flexbox

Flexbox (abreviatura de *Flexible Box*) es un modelo de diseño en CSS que facilita **la alineación, distribución y orden de los elementos** dentro de un contenedor, incluso cuando su tamaño es dinámico o desconocido.

Su objetivo principal es permitir **layouts más adaptables y centrados**, sin necesidad de usar márgenes complicados o cálculos de posición.

Ventajas

- Alinea y distribuye elementos fácilmente tanto **en filas como columnas**.
- Se adapta automáticamente al tamaño del contenedor (ideal para responsive design).
- Permite **centrar vertical y horizontalmente** con pocas líneas.
- Facilita el **espaciado uniforme** entre elementos.

Desventajas

Puede ser confuso al principio por la cantidad de propiedades disponibles.
No es ideal para estructuras muy grandes (ahí conviene usar **Grid**).

Casos de uso

- Diseñar barras de navegación.
- Alinear tarjetas o botones.
- Centrar elementos en la pantalla.
- Crear estructuras simples responsivas.

Propiedades principales

🔸 **En el contenedor (`display: flex`)**

Propiedad	Descripción
`flex-direction`	Define la dirección principal (row, column).
`justify-content`	Alinea horizontalmente los ítems.
`align-items`	Alinea verticalmente los ítems.
`flex-wrap`	Permite que los elementos salten de línea.

`gap`	Espacio entre los elementos.

- **En los hijos (ítems flexibles)**

Propiedad	Descripción
`flex`	Controla el tamaño flexible (crecimiento/encogimiento).
`align-self`	Alinea individualmente un ítem.
`order`	Cambia el orden visual de un ítem.

Ejemplo(codigo):

HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Ejemplo Flexbox</title>
  <link rel="stylesheet" href="estilos.css">
</head>
<body>


  <h2>Ejemplo de Flexbox</h2>
```

```
  <!-- CONTENEDOR PRINCIPAL -->
  <section class="contenedor">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item especial">3</div>
    <div class="item">4</div>
  </section>

  <p class="nota">Se observa cómo los elementos se alinean, reparten espacio y el ítem "3" crece más.</p>

</body>
</html>
```

CSS:

```
/* ===== ESTILOS GENERALES ===== */
body {
    font-family: Arial, sans-serif;
    background-color: #f3f4f6;
    padding: 30px;
  }

  h2 {
    text-align: center;
    color: #222;
  }
```

```css
/* ===== CONTENEDOR FLEX ===== */
.contenedor {
    display: flex; /* Activamos Flexbox */
    flex-direction: row; /* Dirección horizontal (por defecto) */
    justify-content: space-around; /* Espaciado horizontal */
    align-items: center; /* Alineación vertical */
    flex-wrap: wrap; /* Permite saltar de línea si no cabe */
    gap: 10px; /* Espacio entre cajas */
    background-color: #e0e0e0;
    padding: 20px;
    border-radius: 10px;
    min-height: 200px;
}

/* ===== ÍTEMS FLEX ===== */
.item {
    background-color: #90caf9;
    color: #0d47a1;
    border: 2px solid #1976d2;
    border-radius: 8px;
    width: 80px;
    height: 80px;
    text-align: center;
    line-height: 80px;
    font-weight: bold;
```

```
    font-size: 1.2em;
  }

  /* Ítem con tamaño flexible */
  .especial {
    background-color: #a5d6a7;
    border-color: #388e3c;
    color: #1b5e20;
    flex: 2; /* Este ítem ocupa más espacio */
  }

  /* Nota explicativa */
  .nota {
    margin-top: 20px;
    text-align: center;
    color: #444;
  }
```

Ejemplo(visual):


Ejemplo Flexbox
Archivo C:/Users/Neburak/Downloads/ejemplos/idex.html
Ejemplo de Flexbox
1
2
3
4
Se observa cómo los elementos se alinean, reparten espacio y el ítem "3" crece más.

FIN.