# Unidad 3: Box Model y Flexbox

## 3.1. Introducción al Box Model

El **Box Model** es un concepto fundamental y es la base de la maquetación en CSS. Imagina cada elemento HTML como una caja rectangular que está compuesta de **cuatro partes,** que van desde el centro hacia afuera. Comprender cómo funcionan estas partes, te permite controlar el tamaño, el espaciado y la posición de todos los elementos de tu página web.

### ¿Cuáles son estas cuatro partes que mencionamos?

#### 1. Content

##### ¿Qué es?

El **content** es el área central de la caja, donde se muestra el contenido del elemento: texto, imágenes, etc. Su tamaño se define a partir de las propiedades **width** (*ancho*) y **height** (*alto*)

##### Ejemplo

```
<div class="content-box">Contenido del elemento</div>

.content-box {
width: 12rem;
height: 6rem;
background-color: #D3D3D3;
}
```

En este ejemplo, el área de contenido tiene un tamaño de 12rem de ancho y 6rem de alto, y se muestra con un fondo gris claro. Como vimos en la unidad anterior, con CSS se pueden modificar cualquier tipo de estilos. Entonces, si alguien te preguntara ahora: ¿podrías cambiarle el ancho, el alto y el color de fondo? Claramente, ¡porque ya sabes cómo aplicar estilos con CSS!

# 2. Padding

## ¿Qué es?

El **padding** es la separación que se añade **entre el contenido y el borde**, es decir, el espacio que agregamos del *borde hacia adentro*. Sirve para que el contenido no se "pegue" a los bordes de la caja, dándole un respiro visual. Esta propiedad expande el tamaño del elemento seleccionado, sin afectar a otros elementos adyacentes.

## Ejemplo

```
<div class="padding-box">Contenido con Padding</div>

.padding-box {
    width: 12rem;
    height: 6rem;
    background-color:#D3D3D3;
    padding: 2rem;
}
```

Aquí, el **padding** de 2rem agrega espacio alrededor del contenido dentro del elemento, aumentando su tamaño visual sin cambiar las dimensiones definidas por width y height.

# 3. Border

## ¿Qué es?

El **border** es una línea que rodea el *padding* (si es que existe) y el contenido del elemento. El **border** también aumenta el tamaño total del elemento.

## Ejemplo

```
<div class="border-box">Contenido con Border</div>

.border-box {
    width: 12rem;
    height: 6rem;
    background-color:#D3D3D3;
    padding: 2rem;
    border: 5px solid #000;
}
```

En este caso, el **border** de 5px rodea el padding y el contenido, incrementando el tamaño visual del elemento.

# 4. Margin

## ¿Qué es?

El **margin** es el espacio fuera del borde del elemento. Su función principal es separar una caja de otras cajas adyacentes. El **margin** no forma parte del tamaño del elemento, solo crea distancia.

## Ejemplo

```
<div class="margin-box">Contenido con Margin</div>

.margin-box {
    width: 12rem;
    height: 6rem;
    background-color: #D3D3D3;
    padding: 2rem;
    border: 5px solid #000;
    margin: 1rem;
}
```

En este ejemplo, el **margin** de 1rem crea espacio adicional alrededor del borde del elemento, separándolo de otros elementos circundantes.

# Efecto en el Tamaño y Espaciado

## Tamaño del Elemento

El tamaño total del elemento se calcula sumando el **content**, **padding**, **border** y **margin**. La fórmula es:

*Tamaño total* = width + padding izquierdo + padding derecho + borde izquierdo + borde derecho + margin izquierdo + margin derecho

## Ejemplo Completo

```
<div class="box-model">Box Model Completo</div>

.box-model {
    width: 12rem;
    height: 6rem;
    background-color: #D3D3D3;
    padding:2rem;
    border: 5px solid #000;
    margin: 1rem;
}
```

En este ejemplo, el tamaño total visual del elemento sería:

- Ancho total = 12rem (content) + 4rem (padding) + 10px (border) + 2rem (margin)

*Recordatorio:*

a) El padding suma 4rem dado que son 2rem del lado izquierdo + 2rem del lado derecho.
b) El margin suma 2rem dado que es 1rem del lado izquierdo + 1rem del lado derecho.

- Alto total = 6rem (content) + 4rem (padding) + 10px (border) + 2rem (margin)

*Recordatorio:*

c) El padding suma 4rem dado que son 2rem arriba + 2rem abajo.
d) El margin suma 2rem dado que es 1rem arriba + 1rem abajo.

# Guía de Valores Abreviados para ***margin*** **y** ***padding***

Las propiedades ***margin*** y ***padding*** en CSS permiten controlar el espaciado de un elemento de manera concisa mediante el uso de valores, tal como viste anteriormente. Dominar la siguiente sintaxis es esencial para escribir código limpio y eficiente. En los próximos ejemplos usaremos la propiedad margin, pero recordá que aplica lo mismo para el padding:

## 1. Un Valor: Aplicación Uniforme

Al asignar un solo valor, la propiedad se aplica de manera uniforme a los cuatro lados del elemento: arriba, derecha, abajo e izquierda.

- **Sintaxis:**

```
.box-model{
     margin: 1rem;
}
```

- **Resultado: Se establece un margin de 1rem en los cuatro lados, es decir, arriba, derecha, abajo e izquierda.**

## 2. Dos Valores: Eje vertical y horizontal

La utilización de dos valores permite controlar el espaciado en los ejes principales. El primer valor corresponde al eje **vertical** (arriba y abajo), y el segundo al eje **horizontal** (derecha e izquierda).

- **Sintaxis:**

```
.box-model{

     margin: 1rem 2rem;

}
```

- **Resultado:**
    - El primer valor del margin aplica 1rem al eje vertical (arriba y abajo) y el segundo, 2rem al eje horizontal (izquierda y derecha).

## 3. Tres Valores: Distribución Asimétrica

Con tres valores, la propiedad asigna un espaciado distinto a los lados superior e inferior, mientras que los laterales mantienen el mismo valor. La secuencia es **arriba**, **laterales**, y **abajo**.

- **Sintaxis:**

```
.box-model{

     margin: 1rem 2rem 4rem;

}
```

- **Resultado:**
    - El primer valor aplica 1rem en la parte **superior**.
    - El segundo valor aplica 2rem en el **eje horizontal.**
    - El tercer valor aplica 4rem en la parte **inferior.**

## 4. Cuatro Valores: Control Individual por Lado

Este formato es el más detallado y proporciona un control completo. Los valores se aplican en el sentido de las agujas del reloj, comenzando por el lado superior: arriba, derecha, abajo, izquierda.

- **Sintaxis:**

```
.box-model{

     margin: 1rem 3rem 5rem 2rem;

}
```

- **Resultado:**
    - El primer valor aplica 1rem en la parte **superior**.
    - El segundo valor aplica 3rem del lado **derecho**
    - El tercer valor aplica 5rem en la parte **inferior.**

- El cuarto valor aplica 2rem del lado **izquierdo.**

## Conclusión

Como habrás notado, la forma más sencilla de comprender qué son el padding y el margin, es: el padding determina el espacio del border hacia adentro, es decir, cómo generar más espacio entre el content y el border; y el margin es la separación del border hacia afuera.

Comprender el Box Model y cómo cada una de sus partes (content, padding, border y margin) afecta el tamaño y el espaciado de los elementos es crucial para el diseño web efectivo. Al utilizar estas propiedades de manera consciente, puedes controlar con precisión la apariencia y el comportamiento de los elementos HTML en tu página web.

# 3.2. La Propiedad `display` en CSS y su Impacto en la Visualización de los Elementos

La propiedad `display` en CSS es una de las más importantes para controlar cómo se muestran los elementos HTML en una página web. A continuación, se detallan los valores más comunes: `display: block`, `display: inline` y `display: inline-block`, con ejemplos de cómo afectan la visualización de los elementos.

## Valores de la Propiedad `display`

### 1. `display: block`

#### ¿Qué es?

Un elemento con `display: block` se comporta como un bloque, ocupando todo el ancho disponible de su contenedor. Siempre inicia en una nueva línea, empujando a los elementos siguientes hacia abajo. Permite la manipulación de dimensiones con `width`, `height`, `padding` y `margin`.

#### Ejemplos de Elementos de Bloque

Por defecto, los siguientes elementos son de tipo bloque: `<div>`, `<p>`, `<h1> - <h6>`, `<section>`, `<article>`, `<header>`, `<footer>`.

#### Ejemplo en CSS

Elemento de Bloque

Este es un párrafo.

```
.block-element {
  display: block;
  width: 50%;
  margin: 1rem 0;
  padding: 0.5rem;
  background-color: #f0f0f0;
}
```

En este caso, aunque el `div` o el `p` solo ocupen el 50% del ancho, su naturaleza de bloque hará que el siguiente elemento comience en una nueva línea, independientemente del espacio restante.

Si utilizamos un elemento en bloque, como puede ser un `<h2>`, por más que tengas poco material en el content (término del cual hablamos antes), por defecto va a ocupar el 100% de la pantalla.

De igual forma, ten en cuenta que todo esto podemos cambiarlo, recuerda que con CSS podemos modificar todo lo que esté referido a estilos.

## 2. `display: inline`

### ¿Qué es?

Los elementos `inline` solo ocupan el espacio necesario para su contenido. No comienzan en una nueva línea, lo que permite que otros elementos se sitúen a su lado en la misma línea. A diferencia de los elementos de bloque, las propiedades `width`, `height` y los márgenes verticales (`margin-top` y `margin-bottom`) no tienen efecto en ellos.

### Ejemplos de Elementos en Línea

Por defecto, los siguientes elementos son de tipo en línea: **`<span>`**, **`<a>`**, **`<img>`**, **`<strong>`**, **`<em>`**.

### Ejemplo en CSS

```
<span class="inline-element">Elemento en Línea</span>

<a href="#" class="inline-element">Enlace</a>
```

```
.inline-element {
    display: inline;
    color: red;
    font-weight: bold;
}
```

En este ejemplo, los elementos `<span>` y `<a>` se muestran en línea, ocupando solo el espacio necesario para su contenido y permitiendo que otros elementos se alineen a su lado en la misma línea.

### 3. `display: inline-block`

#### ¿Qué es?

El valor `inline-block` ofrece un comportamiento híbrido. El elemento se muestra en línea con otros, lo que significa que no inicia en una nueva línea, pero a la vez, permite el control total de sus dimensiones y márgenes (`width`, `height`, `padding` y `margin`), como si fuera un bloque.

#### Ejemplo en CSS

```
<div class="inline-block-element">Elemento Inline-Block</div>

.inline-block-element {
    display: inline-block;
    width: 8rem;
    height: 6rem;
    background-color: lightgreen;
    margin: 1rem;
}
```

En este ejemplo, el elemento `<div>` se muestra en línea con otros elementos, pero permite definir su tamaño y aplicar márgenes, comportándose como un bloque dentro de una línea.

Este tipo de `display` es muy útil, ya que si queremos que un elemento solamente ocupe el ancho de su contenido (`display: inline`) pero necesitamos a su vez aplicarle margin, padding, height o width, podemos usarlo. En conclusión, nos permite usar margin y padding, darle un width y un height, y el elemento entonces ocupará el tamaño de su contenido.

## Comparación de los Valores `display`

### `block`

- **Ocupación de Ancho Completo**: Ocupa todo el ancho disponible.
- **Nueva Línea**: Siempre comienza en una nueva línea.
- **Dimensiones**: Permite definir ancho y alto.

### `inline`

- **Ocupación de Espacio Necesario**: Solo ocupa el espacio necesario para su contenido.
- **Misma Línea**: Permite que otros elementos se alineen en la misma línea.

- **Dimensiones**: No permite definir ancho y alto.

### `inline-block`

- **Comportamiento Híbrido**: Se alinea en línea con otros elementos pero permite definir dimensiones.
- **Dimensiones**: Permite definir ancho y alto.
- **Misma Línea**: Permite que otros elementos se alineen en la misma línea.

## Conclusión

La propiedad `display` en CSS es crucial para controlar la disposición y el flujo de los elementos HTML en una página web. Comprender cómo funcionan los valores `block`, `inline` e `inline-block` te permitirá diseñar layouts más flexibles y eficientes, adaptando la visualización de los elementos a las necesidades específicas de tu proyecto.

## 3.3. FlexBox

Hasta ahora has comprendido los fundamentos del modelo de caja y los valores básicos de la propiedad `display`, que son esenciales para el comportamiento natural de los elementos en una página. Sin embargo, para ir más allá de la maquetación estática y crear interfaces verdaderamente dinámicas y responsivas, debes dominar **Flexbox**.

Flexbox proporciona propiedades poderosas para manipular la distribución y el alineamiento de los elementos. Este sistema se basa en la interacción entre un **contenedor padre** y sus **elementos hijos**. A continuación, exploraremos las propiedades clave que aplican a ambos.

### Propiedades Principales de Flexbox para el Contenedor

La activación de Flexbox en un elemento padre lo convierte en un `flex container` y le otorga un control total sobre la distribución de sus hijos, los `flex items`.

### 1. `display: flex`

Esta es la propiedad fundamental. Al aplicarla a un elemento, este se convierte en un **flex container**, y sus hijos directos pasan a ser **flex items**. Esto habilita todas las demás propiedades de Flexbox.

### Ejemplo

```
<div class="flex-container">
   <div>Item 1</div>
   <div>Item 2</div>
   <div>Item 3</div>
</div>
```

```
.flex-container {

  display: flex;  background-color:#d3d3d3;

}
```

En este ejemplo, el contenedor `<div class="flex-container">` se convierte en un flex container, permitiendo que sus hijos se comporten como flex items.

## 2. `flex-direction`

La propiedad flex-direction especifica la dirección en la que los elementos flexibles (flex items) se colocan dentro del contenedor.

### *Valores*

- **row** (valor por defecto): Los elementos se colocan en una fila horizontal.
- **row-reverse**: Los elementos se colocan en una fila horizontal en orden inverso.
- **column**: Los elementos se colocan en una columna vertical.
- **column-reverse**: Los elementos se colocan en una columna vertical en orden inverso.

### Ejemplo

```
.flex-container {
   display: flex;
   flex-direction: row; (valor por defecto una vez activamos flexbox)
}

.flex-container {
   display: flex;
   flex-direction: column;
}
```

## 3. `flex-wrap`

Controla si los elementos deben permanecer en una sola línea o si pueden **envolverse** en múltiples líneas cuando no hay suficiente espacio.

### *Valores*

- **nowrap** (valor por defecto)
- **wrap** (los elementos se envuelven cuando no hay espacio)
- **wrap-reverse** (se envuelven de manera invertida)

**Nota:** `flex-flow` es una propiedad abreviada para combinar `flex-direction` y `flex-wrap` (`flex-flow: row wrap;`).

### Ejemplo

```
.flex-container {
    display: flex;
    flex-wrap: wrap;
}
```

## 4. `justify-content`

Alinea los elementos a lo largo del **eje principal** (el eje definido por `flex-direction`, `horizontalmente en fila y verticalmente en una columna`). Es crucial para distribuir el espacio horizontal o vertical.

### *Valores*

- **flex-start (valor por defecto):** Los elementos se alinean al inicio del contenedor.
- **flex-end:** Los elementos se alinean al final del contenedor.
- **center:** Los elementos se centran en el contenedor.
- **space-between:** Los elementos se distribuyen uniformemente con el primer elemento al inicio y el último al final.
- **space-around:** Los elementos se distribuyen uniformemente con espacio igual alrededor de cada uno.
- **space-evenly:** Los elementos se distribuyen con espacio igual entre ellos.

### Ejemplo

```
.flex-container {
    display: flex;
    justify-content: center;
}
```

## 5. `align-items`

Alinea los elementos a lo largo del **eje transversal** (el eje perpendicular al eje principal).

### *Valores*

- **stretch (valor por defecto):** Los elementos se estiran para llenar el contenedor.
- **flex-start:** Los elementos se alinean al inicio del contenedor.
- **flex-end:** Los elementos se alinean al final del contenedor.

- **center:** Los elementos se centran en el contenedor.
- **baseline:** Los elementos se alinean a lo largo de su línea base.

### Ejemplo

```
.flex-container {
    display: flex;
    height: 10rem; /* Necesario para que `align-items` tenga efecto si es que el container no tiene un alto definido */
    align-items: center;
}
```

## 6. `align-content`

Alinea las líneas del contenedor flexible cuando hay espacio extra en el eje transversal.

### *Valores*

- **stretch (valor por defecto):** Las líneas se estiran para ocupar el espacio disponible.
- **flex-start:** Las líneas se alinean al inicio del contenedor.
- **flex-end:** Las líneas se alinean al final del contenedor.
- **center:** Las líneas se centran en el contenedor.
- **space-between:** Las líneas se distribuyen uniformemente con el primer línea al inicio y la última al final.
- **space-around:** Las líneas se distribuyen uniformemente con espacio igual alrededor de cada una.
- **space-evenly:** Las líneas se distribuyen con espacio igual entre ellas.

```
.flex-container {
    display: flex;
    flex-wrap: wrap;
    align-content: space-between; /* Distribuye las líneas con espacio entre ellas */
}
```

Hasta hace poco, cuando queríamos separar los elementos de un contenedor flex, teníamos que usar márgenes individuales en cada ítem. Esto complicaba el mantenimiento y podía dar problemas de alineación.

Hoy en día, Flexbox soporta la propiedad `gap`, que permite definir de manera sencilla la distancia entre los elementos hijos del contenedor.

```
.contenedor {
    display: flex;
    height: 8rem;
    align-items: center;
    gap: 1rem; /* Espacio uniforme entre los ítems */
```

}

## Conclusión

Las propiedades de Flexbox (display: flex, flex-direction, flex-wrap, flex-flow, justify-content, align-items y align-content) proporcionan una gran flexibilidad y control sobre la disposición de los elementos en un contenedor. Utilizando estas propiedades, puedes crear layouts responsivos y dinámicos que se adapten a diferentes tamaños de pantalla y necesidades de diseño.

# 3.5.Propiedades de Flexbox Aplicadas a los Ítems

Una vez que un elemento se ha convertido en un `flex item`, se pueden aplicar propiedades individuales para controlar su tamaño y posición dentro del contenedor.

### 1. `order`

Determina el **orden visual** de un `flex item` dentro del contenedor. El valor por defecto para todos los ítems es `0`. Los ítems con valores de `order` más bajos se muestran primero.

Ejemplo

.flex-container {

display: flex;

}

.item-1 { order: 2; }

.item-2 { order: 1; } /* Este ítem se mostrará primero */

.item-3 { order: 3; }

### 2. `flex-grow`

Define la **capacidad de crecimiento** de un `flex item`. Si hay espacio extra en el contenedor, los ítems con un valor de `flex-grow` mayor crecerán en mayor proporción. El valor por defecto es `0`.

.item-1 { flex-grow: 1; }

.item-2 { flex-grow: 2; } /* Crecerá el doble que Item 1 y Item 3 */

.item-3 { flex-grow: 1; }

**3. `flex-shrink`**

Determina la **capacidad de encogimiento** de un `flex item` si no hay suficiente espacio para todos. El valor por defecto es `1`. Un valor de `0` impide que el ítem se encoja.

.item-2 { flex-shrink: 3; } /* Se encogerá tres veces más que otros ítems */

**4. `flex-basis`**

Define el **tamaño inicial** de un `flex item` antes de que se distribuya el espacio restante. Es similar a `width`, pero específico de Flexbox.

.item-2 {

flex-basis: 8rem; /* Tendrá un tamaño base de 8rem */

}

**Nota:** La propiedad abreviada `flex` es una combinación de `flex-grow`, `flex-shrink` y `flex-basis` (`flex: 1 1 8rem;`).

# Propiedades de Posicionamiento en CSS

El posicionamiento en CSS es fundamental para controlar cómo se colocan los elementos en una página web. A continuación, se describen las diferentes propiedades de posicionamiento: `position: static`, `position: relative`, `position: absolute`, `position: fixed` y `position: sticky`. Se explican sus usos y se proporcionan ejemplos de código para ilustrar cómo se utilizan en proyectos web.

## 1. `position: static`

### ¿Qué es?

`position: static` es el valor por defecto. Los elementos con este valor se posicionan según el flujo normal del documento, sin afectar otros elementos.

### Uso

Se utiliza cuando no se necesita un posicionamiento especial.

### Ejemplo

```
<div class="static-box">Elemento Estático</div>

.static-box {
    position: static;
    display: block;
    width: auto;
    height: auto;
    margin: 0;
    padding: 0;
    background-color: transparent;
    border: none;
    font-family: inherit;
    font-size: inherit;
    color: inherit;
}
```

En este ejemplo, el elemento `<div>` se posiciona de acuerdo al flujo normal del documento.

## 2. `position: relative`

### ¿Qué es?

`position: relative` posiciona el elemento en relación con su posición original en el flujo del documento. Se pueden usar las propiedades `top`, `right`, `bottom` y `left` para desplazar el elemento.

Esta propiedad permite mover el elemento en cualquier dirección y que este no pierda su espacio en el documento.

### Uso

Útil para desplazar un elemento ligeramente desde su posición original sin sacarlo del flujo del documento.

### Ejemplo

```
<div class="relative-box">Elemento Relativo</div>

.relative-box {

    position: relative;

    top: 1.25rem;

    left: 0.6rem;

    background-color: lightblue;

    padding: 0.6rem;

}
```

En este ejemplo, el elemento `<div>` se desplaza 1.25rem hacia abajo y 0.6rem hacia la derecha desde su posición original.

## 3. `position: absolute`

### ¿Qué es?

`position: absolute` posiciona el elemento en relación con el primer ancestro posicionado (no estático). El elemento se elimina del flujo del documento, permitiendo que otros elementos ocupen su lugar.

### Uso

Se usa para colocar un elemento en una ubicación exacta dentro de su contenedor posicionado.

### Ejemplo

```
<div class="container">

    <div class="absolute-box">Elemento Absoluto</div>

</div>

.container {

    position: relative;

    width: 18rem;
```

```
    height: 12rem;

    background-color: lightgray;

}

.absolute-box {

    position: absolute;

    top: 3rem;

    left: 3rem;

    background-color: lightcoral;

    padding: 0.6rem;

}
```

En este ejemplo, el elemento `<div class="absolute-box">` se posiciona 3rem desde la parte superior y 3rem desde la izquierda del contenedor relativo.

## 4. `position: fixed`

### ¿Qué es?

`position: fixed` posiciona el elemento en relación con la ventana del navegador. El elemento se mantiene en la misma posición incluso cuando se desplaza la página.

### Uso

Ideal para elementos que deben estar siempre visibles, como menús de navegación o botones de vuelta al inicio.

### Ejemplo

```
<div class="fixed-box">Elemento Fijo</div>

.fixed-box {

    position: fixed;

    top: 0;

    right: 0;
```

```
    background-color: lightgreen;

    padding: 0.6rem;

}
```

En este ejemplo, el elemento `<div class="fixed-box">` se mantiene en la esquina superior derecha de la ventana del navegador, incluso al desplazarse.

## 5. `position: sticky`

### ¿Qué es?

`position: sticky` alterna entre `relative` y `fixed` dependiendo del desplazamiento de la página. El elemento se comporta como `relative` hasta que su contenedor alcanza una posición de desplazamiento específica, momento en el que se comporta como `fixed`.

### Uso

Útil para encabezados o menús que deben quedarse fijos después de desplazarse una cierta distancia.

### Ejemplo

```
<div class="sticky-container">

    <div class="sticky-box">Elemento Sticky</div>

    <p>Contenido de ejemplo...</p>

</div>

.sticky-container {

    height: 62rem;

    background-color: lightyellow;

}

.sticky-box {

    position: sticky;

    top: 0.6rem;

    background-color: lightpink;
```

```
    padding: 0.6rem;
}
```

En este ejemplo, el elemento `<div class="sticky-box">` se mantiene a 0.6rem del borde superior de la ventana del navegador cuando se desplaza, hasta que se alcanza el final de su contenedor.

## Control de superposición con `z-index`

Cuando varios elementos se solapan en una página, necesitamos decidir cuál aparece por encima de cuál. Para ello usamos la propiedad `z-index`, que controla el orden en el “eje Z” (profundidad).

### Ejemplo

A continuación encontrarán un ejemplo de código que incluye recursos gráficos y visualizaciones dinámicas para complementar la explicación de las propiedades `display` y `position` en CSS. Este código utiliza HTML y CSS para mostrar los efectos visuales de estas propiedades.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Demostración de CSS: display y position</title>
    <style>
        /* Estilo general */
        body {
            font-family: Arial, sans-serif;
            margin: 1.25rem;
            line-height: 1.6;
```

```
}
h2 {
    margin-top: 2.5rem;
    color: #333;
}
.example-container {
    margin-bottom: 2.5rem;
}
.example-title {
    font-weight: bold;
    margin-bottom: 0.6rem;
}
/* Ejemplos de display */
.block-example {
    display: block;
    width: 100%;
    margin-bottom: 0.6rem;
    padding: 0.6rem;
    background-color: #d9f0fc;
    border: 1px solid #00aaff;
}
.inline-example {
    display: inline;
    color: #ff5733;
```

```
    font-weight: bold;
    margin: 0.3rem;
}
.inline-block-example {
    display: inline-block;
    width: 9rem;
    height: 6rem;
    background-color: #c5e1a5;
    margin: 0.3rem;
    text-align: center;
    line-height: 6rem;
    border: 1px solid #7cb342;
}
/* Ejemplos de position */
.relative-example {
    position: relative;
    top: 1.25rem;
    left: 0.6rem;
    background-color: #ffcc80;
    padding: 0.6rem;
}
.absolute-container {
    position: relative;
    width: 18rem;
```

```
    height: 12.5rem;
    background-color: #e0e0e0;
    margin-bottom: 1.25rem;
}
.absolute-example {
    position: absolute;
    top: 3rem;
    left: 3rem;
    background-color: #ff8a80;
    padding: 0.6rem;
}
.fixed-example {
    position: fixed;
    top: 0.6rem;
    right: 0.6rem;
    background-color: #b39ddb;
    padding: 0.6rem;
    z-index: 1000;
}
.sticky-container {
    height: 62.5rem;
    background-color: #fff9c4;
    padding: 0.6rem;
}
```

```
        .sticky-example {
            position: sticky;
            top: 0.6rem;
            background-color: #ffc107;
            padding: 0.6rem;
        }
    </style>
</head>
<body>
    <h1>Demostración de CSS: display y position</h1>
    <h2>Propiedad display</h2>
    <div class="example-container">
        <div class="example-title">display: block</div>
        <div class="block-example">Este es un elemento de bloque</div>
    </div>
    <div class="example-container">
        <div class="example-title">display: inline</div>
        <span class="inline-example">Elemento en línea 1</span>
        <span class="inline-example">Elemento en línea 2</span>
    </div>
    <div class="example-container">
        <div class="example-title">display: inline-block</div>
        <div class="inline-block-example">Elemento 1</div>
        <div class="inline-block-example">Elemento 2</div>
```

```
    </div>
    <h2>Propiedad position</h2>
    <div class="example-container">
        <div class="example-title">position: relative</div>
        <div class="relative-example">Elemento relativo</div>
    </div>
    <div class="example-container">
        <div class="example-title">position: absolute</div>
        <div class="absolute-container">
            <div class="absolute-example">Elemento absoluto</div>
        </div>
    </div>
    <div class="example-container">
        <div class="example-title">position: fixed</div>
        <div class="fixed-example">Elemento fijo</div>
    </div>
    <div class="example-container sticky-container">
        <div class="example-title">position: sticky</div>
        <div class="sticky-example">Elemento sticky</div>
        <p>Desplázate hacia abajo para ver cómo el elemento sticky se comporta.</p>
    </div>
</body>
</html>
```

## Conclusión

Comprender y utilizar correctamente las propiedades de posicionamiento en CSS (`static`, `relative`, `absolute`, `fixed` y `sticky`) es crucial para diseñar layouts precisos y funcionales. Cada valor de `position` ofrece un conjunto único de comportamientos y posibilidades que pueden adaptarse a diversas necesidades de diseño web.

---