

# Unidad 3: Box Model y Flexbox

## Detalle de las Cuatro Partes Principales del Box Model

El Box Model es un concepto fundamental en CSS que describe cómo se colocan y se dimensionan los elementos HTML en una página web. Comprender las cuatro partes principales del Box Model - content, padding, border y margin - es esencial para controlar el tamaño y el espaciado de los elementos.

### 1. Content

#### ¿Qué es?

El contenido (content) es el área donde se muestra el contenido del elemento, como texto, imágenes o otros elementos embebidos.

#### Ejemplo

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

En este ejemplo, el área de contenido tiene un tamaño de 200px de ancho y 100px 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 relleno (padding) es el espacio entre el contenido del elemento y su borde. El padding expande el tamaño del elemento sin afectar a otros elementos adyacentes.

#### Ejemplo

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

```
.padding-box {  
  
  width: 200px;
```

```
height: 100px;

background-color: lightgray;

padding: 20px;

}
```

Aquí, el padding de 20px 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 borde (border) es una línea que rodea el padding (si existe) y el contenido del elemento. El borde también aumenta el tamaño total del elemento.

#### Ejemplo

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

```
.border-box {

width: 200px;

height: 100px;

background-color: lightgray;

padding: 20px;

border: 5px solid black;

}
```

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

### 4. Margin

#### ¿Qué es?

El margen (margin) es el espacio fuera del borde del elemento. El margen no afecta el tamaño del elemento, pero sí crea espacio entre el elemento y otros elementos adyacentes.

## Ejemplo

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

```
.margin-box {  
  
    width: 200px;  
  
    height: 100px;  
  
    background-color: lightgray;  
  
    padding: 20px;  
  
    border: 5px solid black;  
  
    margin: 30px;  
  
}
```

En este ejemplo, el margen de 30px 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 contenido, el padding, el borde y el margen. 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: 200px;  
  
    height: 100px;  
  
    background-color: lightgray;  
  
    padding: 20px;  
  
    border: 5px solid black;
```

```
margin: 30px;  
}
```

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

- Ancho total = 200px (content) + 40px (padding) + 10px (border) + 60px (margin) = 310px
- Alto total = 100px (content) + 40px (padding) + 10px (border) + 60px (margin) = 210px

## 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.

## 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 explican los valores `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?

El valor `block` hace que un elemento se comporte como un bloque. Los elementos de bloque ocupan todo el ancho disponible de su contenedor y comienzan en una nueva línea.

#### 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

```
<div class="block-element">Elemento de Bloque</div>
```

```
<p class="block-element">Este es un párrafo.</p>
```

```
.block-element {  
  
    display: block;  
  
    width: 100%;  
  
    margin-top: 0;  
  
    margin-bottom: 1rem;  
  
    padding: 0.5rem;  
  
    background-color: #f0f0f0;  
  
    border: 1px solid #ccc;  
  
    font-family: Arial, sans-serif;  
  
    font-size: 16px;  
  
}
```

En este ejemplo, los elementos `<div>` y `<p>` se muestran como bloques, ocupando el 50% del ancho de su contenedor, empezando en una nueva línea y con un margen de 10px arriba y abajo.

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 este referido a estilos.

## 2. `display: inline`

### ¿Qué es?

El valor `inline` hace que un elemento se comporte como un elemento en línea. Los elementos en línea no comienzan en una nueva línea y solo ocupan el espacio necesario para su contenido, todo lo contrario que en el `display: block`.

Importante: en un elemento con este tipo de **display**, no se puede aplicar margin, padding, height ni width.

## 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** combina características de los elementos en línea y de bloque. Un elemento con **display: inline-block** se muestra en línea con otros elementos, pero permite definir dimensiones como ancho y alto, similares a un elemento de bloque.

## Ejemplo en CSS

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

```
.inline-block-element {  
  
    display: inline-block;  
  
    width: 150px;  
  
    height: 100px;
```

```
background-color: lightgreen;

margin: 10px;

}
```

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.

# 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: 20px;  
  
    left: 10px;  
  
    background-color: lightblue;  
  
    padding: 10px;  
  
}
```

En este ejemplo, el elemento `<div>` se desplaza 20px hacia abajo y 10px 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: 300px;

  height: 200px;

  background-color: lightgray;

}

.absolute-box {

  position: absolute;

  top: 50px;

  left: 50px;

  background-color: lightcoral;

  padding: 10px;

}
```

En este ejemplo, el elemento `<div class="absolute-box">` se posiciona 50px desde la parte superior y 50px 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: 10px;  
  
}
```

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: 1000px;

  background-color: lightyellow;

}

.sticky-box {

  position: sticky;

  top: 10px;

  background-color: lightpink;

  padding: 10px;

}
```

En este ejemplo, el elemento `<div class="sticky-box">` se mantiene a 10px del borde superior de la ventana del navegador cuando se desplaza, hasta que se alcanza el final de su contenedor.

## 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: 20px;
```

```
    line-height: 1.6;
```

```
}
```

```
h2 {
```

```
    margin-top: 40px;
```

```
    color: #333;
```

```
}
```

```
.example-container {
```

```
    margin-bottom: 40px;
```

```
}
```

```
.example-title {
```

```
    font-weight: bold;
```

```
    margin-bottom: 10px;
```

```
}
```

```
/* Ejemplos de display */
```

```
.block-example {  
  
    display: block;  
  
    width: 100%;  
  
    margin-bottom: 10px;  
  
    padding: 10px;  
  
    background-color: #d9f0fc;  
  
    border: 1px solid #00aaff;  
  
}
```

```
.inline-example {  
  
    display: inline;  
  
    color: #ff5733;  
  
    font-weight: bold;  
  
    margin: 5px;  
  
}
```

```
.inline-block-example {  
  
    display: inline-block;  
  
    width: 150px;  
  
    height: 100px;  
  
    background-color: #c5e1a5;  
  
    margin: 5px;  
  
    text-align: center;  
  
    line-height: 100px;  
  
    border: 1px solid #7cb342;  
  
}
```

```
/* Ejemplos de position */
```

```
.relative-example {  
  
    position: relative;  
  
    top: 20px;  
  
    left: 10px;  
  
    background-color: #ffcc80;  
  
    padding: 10px;  
  
}
```

```
.absolute-container {  
  
    position: relative;  
  
    width: 300px;  
  
    height: 200px;  
  
    background-color: #e0e0e0;  
  
    margin-bottom: 20px;  
  
}
```

```
.absolute-example {  
  
    position: absolute;  
  
    top: 50px;  
  
    left: 50px;  
  
    background-color: #ff8a80;  
  
    padding: 10px;  
  
}
```

```
.fixed-example {  
  
    position: fixed;
```

```
    top: 10px;

    right: 10px;

    background-color: #b39ddb;

    padding: 10px;

    z-index: 1000;

}
```

```
.sticky-container {

    height: 1000px;

    background-color: #fff9c4;

    padding: 10px;

}
```

```
.sticky-example {

    position: sticky;

    top: 10px;

    background-color: #ffc107;

    padding: 10px;

}
```

```
</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>

<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.

## Propiedades Principales de Flexbox para el Contenedor

Flexbox es un modelo de diseño en CSS que facilita la creación de layouts flexibles y eficientes. Se centra en el uso de un contenedor flexible (flex container) y elementos hijos flexibles (flex items). Esto nos permitirá organizar los elementos de nuestra web de una forma más dinámica, y servirá a futuro para adaptar tu sitio a distintos dispositivos.

A continuación, se explican las propiedades principales que se aplican al contenedor flex, junto con ejemplos de código CSS.

## 1. display: flex

### ¿Qué es?

La propiedad `display: flex` define un contenedor como un flex container y habilita el uso de todas las propiedades Flexbox en sus elementos hijos.

### Ejemplo

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

```
  <div>Item 1</div>
```

```
  <div>Item 2</div>
```

```
  <div>Item 3</div>
```

```
</div>
```

CSS

```
.flex-container { display: flex; background-color: lightgray; }
```

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

### ¿Qué es?

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

CSS

```
.flex-container {
```

```
display: flex;

flex-direction: row;

}

css

.flex-container {

display: flex;

flex-direction: column;

}
```

### 3. flex-wrap

#### ¿Qué es?

La propiedad **flex-wrap** especifica si los elementos flexibles deben ajustarse en una sola línea o pueden envolverse en múltiples líneas.

#### Valores

- **nowrap** (valor por defecto): Todos los elementos se colocan en una sola línea.
- **wrap**: Los elementos se envuelven en múltiples líneas según sea necesario.
- **wrap-reverse**: Los elementos se envuelven en múltiples líneas en orden inverso.

#### Ejemplo

```
css

.flex-container {

display: flex;

flex-wrap: wrap;

}
```

## 4. flex-flow

### ¿Qué es?

La propiedad **flex-flow** es una forma abreviada de establecer tanto **flex-direction** como **flex-wrap**.

### Sintaxis

CSS

```
flex-flow: <flex-direction> <flex-wrap>;
```

### Ejemplo

CSS

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

## 5. justify-content

### ¿Qué es?

La propiedad **justify-content** alinea los elementos flexibles a lo largo del eje principal del contenedor (horizontalmente en una fila, verticalmente en una columna).

### 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

CSS

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

## 6. align-items

### ¿Qué es?

La propiedad **align-items** alinea los elementos flexibles a lo largo del eje transversal (perpendicular al eje principal) del contenedor.

### 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

CSS

```
.flex-container {  
  
    display: flex;  
  
    align-items: center;  
  
}
```

## 7. align-content

### ¿Qué es?

La propiedad **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.

## Ejemplo

CSS

```
.flex-container {  
  
    display: flex;  
  
    flex-wrap: wrap;  
  
    align-content: space-between;  
  
}
```

## 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.

---

## Propiedades de Flexbox Aplicadas a los Ítems (Flex Items)

Además de las propiedades que se aplican al contenedor flex (flex container), Flexbox ofrece varias propiedades que afectan directamente a los elementos hijos (flex items). A continuación, se describen las propiedades **order**, **flex-grow**, **flex-shrink** y

**flex-basis**, junto con ejemplos de código para ilustrar su uso y cómo afectan el comportamiento de los elementos flexibles.

## 1. order

### ¿Qué es?

La propiedad **order** determina el orden en que se muestran los ítems flexibles dentro del contenedor flex. Por defecto, todos los ítems tienen un valor **order** de 0.

### Uso

Se utiliza para cambiar el orden de los ítems sin alterar el orden en el HTML.

### Ejemplo

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

  <div class="item" style="order: 2;">Item 1</div>

  <div class="item" style="order: 1;">Item 2</div>

  <div class="item" style="order: 3;">Item 3</div>

</div>
```

### CSS

```
.flex-container { display: flex; } .item { padding: 10px;
background-color: lightcoral; margin: 5px; }
```

En este ejemplo, aunque **Item 1** aparece primero en el HTML, **Item 2** se mostrará primero en el contenedor flex debido a su valor **order** de 1, seguido por **Item 1** con **order** 2, y finalmente **Item 3** con **order** 3.

## 2. flex-grow

### ¿Qué es?

La propiedad **flex-grow** define la capacidad de un ítem flexible para crecer en proporción a los demás ítems flexibles dentro del mismo contenedor. El valor por defecto es 0 (no crece).

### Uso

Se utiliza para permitir que un ítem ocupe más espacio disponible dentro del contenedor.



## Ejemplo

html

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

  <div class="item" style="flex-grow: 1;">Item 1</div>

  <div class="item" style="flex-grow: 2;">Item 2</div>

  <div class="item" style="flex-grow: 1;">Item 3</div>

</div>
```

css

```
.flex-container {

  display: flex;

}

.item {

  padding: 10px;

  background-color: lightblue;

  margin: 5px;

}
```

En este ejemplo, **Item 2** crecerá dos veces más que **Item 1** y **Item 3** debido a su valor **flex-grow** de 2.

## 3. flex-shrink

### ¿Qué es?

La propiedad **flex-shrink** determina la capacidad de un ítem flexible para reducir su tamaño si es necesario. El valor por defecto es 1 (puede encogerse).

### Uso

Se utiliza para controlar cuánto puede encogerse un ítem en relación con los demás ítems dentro del contenedor.

## Ejemplo

html

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

  <div class="item" style="flex-shrink: 1;">Item 1</div>

  <div class="item" style="flex-shrink: 3;">Item 2</div>

  <div class="item" style="flex-shrink: 1;">Item 3</div>

</div>
```

css

```
.flex-container {

  display: flex;

  width: 300px;

}

.item {

  width: 200px;

  padding: 10px;

  background-color: lightgreen;

  margin: 5px;

}
```

En este ejemplo, cuando el contenedor tiene un ancho de 300px, **Item 2** se encogerá tres veces más que **Item 1** y **Item 3** debido a su valor **flex-shrink** de 3.

## 4. flex-basis

### ¿Qué es?

La propiedad **flex-basis** define el tamaño inicial de un ítem flexible antes de que se distribuya el espacio restante. Puede ser un valor fijo (px, em, etc.) o **auto**.

## Uso

Se utiliza para establecer el tamaño base de un ítem antes de aplicar **flex-grow** o **flex-shrink**.

## Ejemplo

htm

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

  <div class="item" style="flex-basis: 100px;">Item 1</div>

  <div class="item" style="flex-basis: 150px;">Item 2</div>

  <div class="item" style="flex-basis: 100px;">Item 3</div>

</div>
```

css

```
.flex-container {

  display: flex;

}

.item {

  padding: 10px;

  background-color: lightpink;

  margin: 5px;

}
```

En este ejemplo, **Item 1** y **Item 3** tendrán un tamaño inicial de 100px, mientras que **Item 2** tendrá un tamaño inicial de 150px.

## Conclusión

Las propiedades de Flexbox aplicadas a los ítems (**order**, **flex-grow**, **flex-shrink** y **flex-basis**) proporcionan un control preciso sobre cómo los elementos flexibles se distribuyen y dimensionan dentro de un contenedor flex. Comprender y utilizar estas propiedades te permitirá diseñar layouts más flexibles y adaptables a diferentes necesidades de diseño y tamaños de pantalla.

