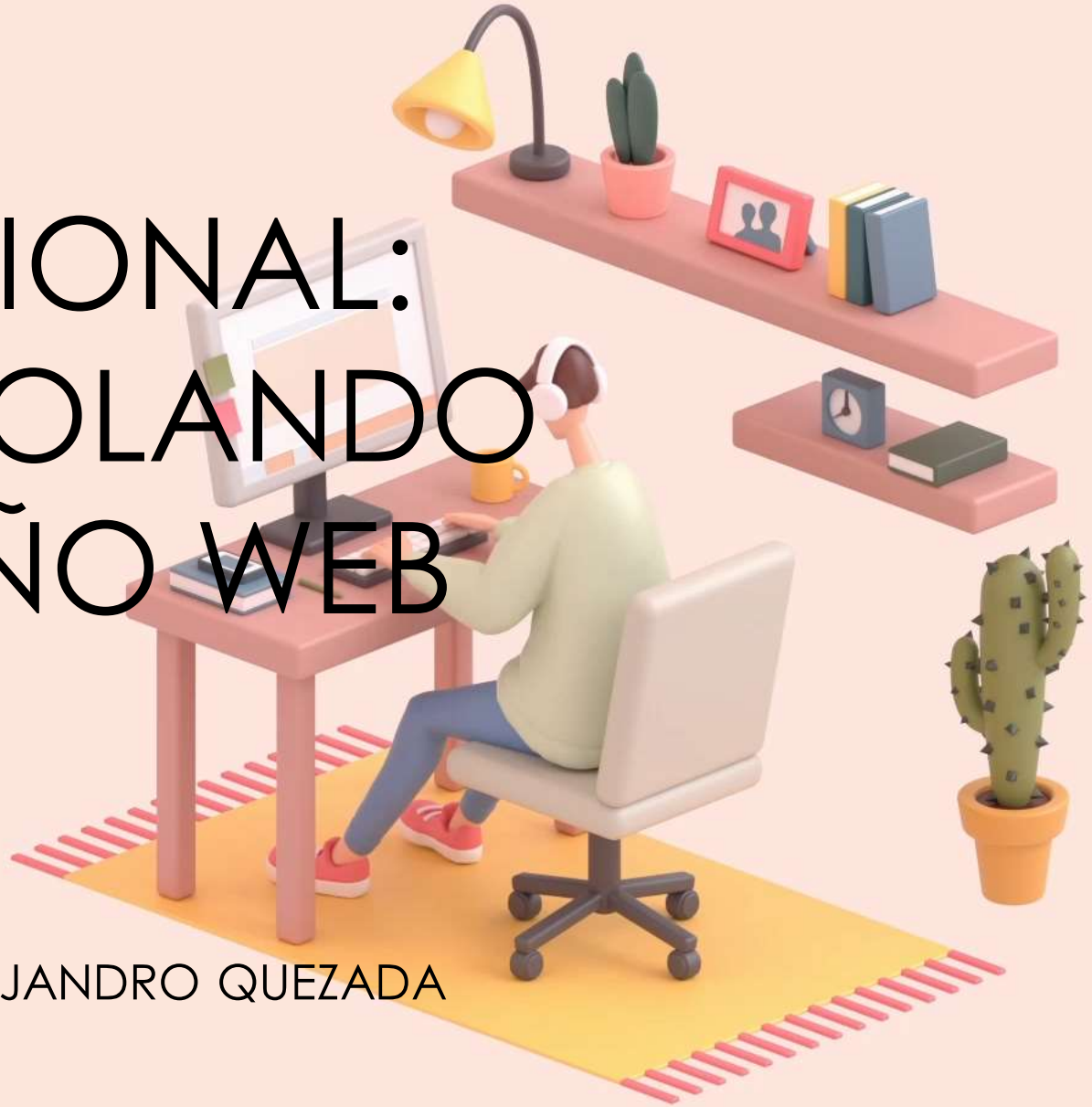
CSS PROFESIONAL: CONTROLANDO EL DISEÑO WEB



ING. MAURICIO ALEJANDRO QUEZADA BUSTILLO

CONEXIÓN CON LA CLASE ANTERIOR

En la clase anterior vimos:

- ✓ Qué es CSS
- ✓ Cómo aplicar estilos (inline, interno, externo)
- ✓ Sintaxis básica
- ✓ Selectores básicos
- ✓ Colres, tipografía y fondos
- ✓ Introducción al Box Model
- ✓ Cascada básica

¿CUÁNDO EMPEZAMOS A FALLAR CON CSS?

Situaciones comunes

❌ Un estilo no se aplica y no sabes por qué
❌ El elemento ocupa más espacio del esperado
❌ El margen no se comporta como esperabas
❌ Dos estilos se contradicen
❌ Un elemento se mueve "solo"

👉 CSS parece impredecible.

⚠ ¿Por qué ocurre esto?

- Porque CSS NO es solo estilos.

CSS controla:

✓ el flujo del documento
✓ el modelo de caja
✓ el posicionamiento
✓ la prioridad de reglas
✓ el comportamiento visual

CÓMO PIENSA EL NAVEGADOR

El navegador sigue estos pasos:

1 Lee el HTML → construye la estructura (DOM)
2 Lee el CSS → construye las reglas de estilo
3 Combina ambos → crea el **árbol de renderizado**
4 Calcula tamaños y posiciones
5 Dibuja los elementos en pantalla

👉 Este proceso se llama **renderizado**.

⚙ **¿Qué decide CSS?**

CSS determina:

✓ tamaño de los elementos
✓ posición en la página
✓ espacios entre elementos
✓ orden visual
✓ visibilidad

⚠ **Importante**

👉 Si cambias CSS… el navegador recalcula todo.

Esto se llama:

reflow → recalcular tamaños
repaint → redibujar estilos

TODO EN CSS ES UNA CAJA

🎁 **Idea fundamental**

👉 En CSS **cada elemento HTML se comporta como una caja**.

- No importa si es:
- un párrafo <p>
- un título <h1>
- una imagen <img>
- un contenedor

Cada caja contiene:

✓ contenido
✓ espacio interno
✓ borde
✓ espacio externo

👉 Esto se conoce como:

BOX MODEL

PARTES DEL BOX MODEL

1 Content (Contenido)

Es lo que contiene el elemento:

✓ texto
✓ imágenes
✓ botones

- width: 200px;
- height: 100px;

👉 define el área del contenido.

2 Padding (Espacio interno)

Espacio entre el contenido y el borde.

- padding: 20px;

✓ aumenta el espacio interno
✓ aumenta el tamaño visual

3 Border (Borde)

Línea que rodea el elemento.

- border: 3px solid black;

4 Margin (Espacio externo)

Espacio que separa el elemento de otros.

- margin: 20px;

✓ separa elementos
✓ no cambia el fondo
✓ no forma parte visual del elemento

EL PROBLEMA REAL DEL BOX MODEL

Observa este código:

```
div {
  width: 200px;
  padding: 20px;
  border: 5px solid black;
}
```

? Pregunta:

👉 ¿El ancho final del elemento es 200px?

❌ NO.

El ancho real será:

- contenido → 200px
- padding izquierda + derecha → 40px
- borde izquierda + derecha → 10px
- 👉 **TOTAL = 250px**

LA SOLUCIÓN PROFESIONAL: box-sizing

Problema

Por defecto:

☞ width **NO incluye** padding ni border
☞ esto complica el diseño

✅ Solución usada por profesionales

```
* {
  box-sizing: border-box;
}
```

🔍 ¿Qué hace?

Ahora el ancho declarado **incluye**:

✓ contenido
✓ padding
✓ borde

☞ el tamaño final se mantiene exacto.

📐 Ejemplo

```
div {
  width: 200px;
  padding: 20px;
  border: 5px solid black;
}
```

✓ ancho final = **200px reales**

MARGIN VS PADDING

- **Padding → espacio interno**

✓ separa el contenido del borde
✓ el fondo del elemento se expande
✓ forma parte visual del elemento

```
div {
  padding: 20px;
  background: lightblue;
}
```

👉 el color incluye el padding.

- **Margin → espacio externo**

✓ separa el elemento de otros
✓ NO afecta el fondo
✓ crea distancia entre cajas

```
div {
  margin: 20px;
}
```

👉 el espacio queda fuera del color.

MARGIN COLLAPSE

¿Te pasó esto?

Asignas margen superior e inferior:

```
p {
  margin: 20px 0;
}
```

Esperas 40px entre párrafos…

👉 pero solo hay **20px** 😲

⚠ ¿Qué está ocurriendo?

Esto se llama:

- **Margin Collapse**

👉 los márgenes verticales se combinan.

Cómo funciona

Si dos elementos tienen:

margin-bottom: 20px

margin-top: 20px

👉 el espacio final será **20px**, no 40px.

📌 Cuándo ocurre

✓ entre elementos verticales consecutivos
✓ entre padre e hijo
✓ solo en dirección vertical

! Cuándo NO ocurre

❌ en márgenes horizontales
❌ si hay padding o border entre ellos
❌ si el contenedor tiene overflow
❌ si usamos flexbox (veremos después)

💡 Cómo evitarlo

✓ agregar padding al contenedor
✓ usar border transparente
✓ cambiar el flujo

UNIDADES DE MEDIDA EN CSS

- **¿Por qué existen distintas unidades?**

Porque los elementos pueden adaptarse a:

✓ pantallas
✓ contenedores
✓ tamaños de texto
✓ resoluciones distintas

👉 Elegir la unidad correcta es clave.

px (pixeles)

Unidad fija.

p { font-size: 16px; }

✓ precisa
✓ predecible
✕ no adaptable

% (porcentaje)

Relativo al tamaño del contenedor padre.

div { width: 50%; }

✓ adaptable
✓ útil para layouts

em

Relativo al tamaño del elemento padre.

div { font-size: 2em; }

👉 depende del padre.

rem

rem

Relativo al tamaño raíz (html).

p { font-size: 1.5rem; }

👉 más predecible que em.

vh / vw

Relativo a la pantalla.

div { height: 100vh; }

✓ útil para pantallas completas

PX vs EM vs REM (ENTENDIENDO LAS DIFERENCIAS)

Elegir la unidad correcta evita errores de escala y mejora la accesibilidad.

- **PX (pixeles)**

✓ tamaño fijo
✓ exacto
✓ no depende de nada

p { font-size: 16px; }

✗ no escala con el usuario
✗ no es accesible

- **EM (relativo al padre)**

Depende del tamaño del elemento padre.

body { font-size: 16px; }

div { font-size: 2em; }

👉 resultado: 32px

⚠ puede multiplicarse si se anida.

- **REM (relativo al root)**

Depende del tamaño del <html>.

html { font-size: 16px; }

p { font-size: 2rem; }

👉 resultado: 32px

✓ consistente
✓ predecible
✓ accesible

DISPLAY: CÓMO SE COMPORTAN LOS ELEMENTOS

display

define cómo un elemento se muestra y ocupa espacio.

1 display: block

Ejemplos: , <p>, <h1>

✓ ocupa todo el ancho disponible
✓ inicia en nueva línea
✓ permite width, height, margin, padding

```
div {
  display: block;
}
```

2 display: inline

Ejemplos: , <a>,

✓ ocupa solo su contenido
✓ no inicia nueva línea
✓ NO acepta width ni height

```
span {
  display: inline;  }
```

3 display: inline-block

Combina lo mejor de ambos.

✓ permanece en línea
✓ permite width y height
✓ permite padding y margin

```
a {
  display: inline-block;
}
```

Idea clave

👉 display controla el flujo del documento.

DISPLAY: NONE vs VISIBILITY

A veces no queremos mostrar un elemento… pero hay **dos formas diferentes** de hacerlo.

 display: none

El elemento:

❌ desaparece completamente
❌ no ocupa espacio
❌ se elimina del flujo del documento

```
.menu {
  display: none;
}
```

👉 es como si no existiera.

 visibility: hidden

El elemento:

✓ sigue ocupando espacio
✓ permanece en el flujo
✓ solo se vuelve invisible

```
.menu {
  visibility: hidden;
}
```

👉 deja un espacio vacío.

👀 **Diferencia visual**

display: none → desaparece
visibility: hidden → se oculta pero deja espacio

POSITION: CONTROLANDO LA UBICACIÓN

position

podemos controlar su ubicación.

1 position: static (por defecto)

✓ todos los elementos comienzan aquí
✓ siguen el flujo normal
✓ no responden a top, left, right, bottom

```
div {
  position: static;
}
```

2 position: relative

✓ se mueve respecto a su posición original
✓ mantiene su espacio en el flujo

```
div {
  position: relative;
  top: 10px;
}
```

☞ se mueve, pero el espacio original queda reservado.

3 position: absolute

✓ sale del flujo normal
✓ se posiciona respecto al padre posicionado
✓ no ocupa espacio original

```
.box {
  position: absolute;
  top: 0;
  right: 0;
}
```

Idea clave

☞ absolute necesita un padre con position relativa.

POSITION: ABSOLUTE EN PROFUNDIDAD

Comportamiento clave

Cuando usamos:

```
.box {
  position: absolute;
}
```

el elemento:

❌ sale del flujo normal
❌ deja de ocupar espacio
❌ puede superponerse a otros elementos

🔑 **¿Respecto a qué se posiciona?**

Se posiciona respecto al:

✓ primer padre con position distinta de static
✓ si no existe → respecto al <body>

👇 **Ejemplo correcto**

```
.container {
  position: relative;
}

.box {
  position: absolute;
  top: 0;
  right: 0;
}
```

👉 la caja se ubica dentro del contenedor.

⚠️ **Ejemplo incorrecto**

Si el padre no tiene position:

👉 el elemento se moverá respecto a la pantalla.

POSITION: FIXED (ELEMENTOS FIJOS EN PANTALLA)

¿Qué hace position: fixed?

El elemento:

✓ se fija respecto a la ventana del navegador
✓ NO se mueve al hacer scroll
✓ sale del flujo normal

```
header {
  position: fixed;
  top: 0;
  width: 100%;
}
```

👉 permanece visible siempre.

👀 Ejemplos reales

✓ barra de navegación fija
✓ botón "volver arriba"
✓ chat flotante
✓ banners persistentes

⚠ Importante

El elemento fijo:

❌ no ocupa espacio en el flujo
❌ puede cubrir contenido

👉 por eso se usa margen superior en el body.

🛠 Ejemplo solución

```
body {
  margin-top: 60px;
}
```

💡 Diferencia con absolute

absolute → relativo al contenedor
fixed → relativo a la pantalla

POSITION: RELATIVE vs ABSOLUTE vs FIXED

RELATIVE

✓ permanece en el flujo
✓ conserva su espacio
✓ se mueve desde su posición original

☞ útil como referencia para elementos absolute.

ABSOLUTE

✓ sale del flujo
✓ no ocupa espacio
✓ se posiciona respecto al padre posicionado

☞ útil para ubicar elementos dentro de contenedores.

FIXED

✓ sale del flujo
✓ se fija a la pantalla
✓ no se mueve con el scroll

☞ útil para navegación persistente.

👀 Ejemplo visual conceptual

RELATIVE → se mueve, pero su espacio queda
ABSOLUTE → flota dentro del contenedor
FIXED → flota respecto a la pantalla

Regla mental rápida

✓ relative = referencia
✓ absolute = ubicación precisa
✓ fixed = posición permanente

ESPECIFICIDAD EN CSS (QUIÉN TIENE MÁS PODER)

LA ESPECIFICIDAD

👉 es el “peso” de un selector.

⚖️ Jerarquía de especificidad

De menor a mayor prioridad:

- Selector universal *
- Etiquetas → p, h1, div
- Clases → .menu
- Atributos → [type="text"]
- ID → #header
- Inline styles
- !important 🔟

👇 Ejemplo

```
p { color: blue; }
.texto { color: green; }
#titulo { color: red; }
```

👉 gana el **ID**.

Cómo piensa el navegador

No importa el orden…

👉 gana el selector **más específico**.

⚠️ Error común

Usar IDs o !important para “forzar” estilos.

👉 esto genera CSS difícil de mantener.

💡 Regla profesional

✓ usar clases para estilos
✓ evitar !important
✓ mantener selectores simples

CÓMO CALCULAR LA ESPECIFICIDAD

☞ La especificidad no es aleatoria.
El navegador usa un sistema de **puntuación**.

Tipo de selector	Valor
Inline style	1000
ID	100
Clase / atributo	10
Etiqueta	1

EL ORDEN TAMBIÉN IMPORTA

Hasta ahora sabemos:

✓ la especificidad decide quién gana
✓ pero… ¿qué pasa si dos selectores tienen el mismo peso?

👉 entra en juego el:

orden de aparición

👇 **Ejemplo**

```
p {
  color: blue;
}
p {
  color: red;
}
```

👉 el texto será **rojo**.

 ¿Por qué?

Ambos selectores tienen la misma especificidad.

✓ gana el que aparece **último**.

👇 **Otro ejemplo**

```
.card p {
  color: green;
}
```

```
.card p {
  color: orange;
}
```

👉 el color será naranja.

⚠ **Esto explica muchos errores**

❌ estilos que cambian sin razón
❌ conflictos entre archivos CSS
❌ resultados inesperados

💡 **Regla profesional**

✓ organiza tu CSS
✓ evita duplicar reglas
✓ escribe estilos de lo general → a lo específico

!important: EL BOTÓN DE EMERGENCIA 🚨

A veces verás esta regla:

```
p {
  color: red !important;
}
```

👉 obliga al navegador a aplicar ese estilo.

¿Qué hace realmente?

!important:

✓ ignora la especificidad
✓ ignora el orden
✓ fuerza la prioridad máxima

👉 gana sobre casi todo.

👀 **Ejemplo**

```
p { color: blue; }

.texto {
  color: green !important;
}
```

👉 el texto será verde.

⚠ **Problema**

El abuso de !important genera:

❌ CSS imposible de mantener
❌ conflictos difíciles de resolver
❌ código desordenado

HERENCIA EN CSS (ESTILOS QUE SE TRANSMITEN)

☞ Algunos estilos se **heredan automáticamente** desde los elementos padre.

Esto significa que los hijos adoptan ciertas propiedades sin declararlas.

Ejemplo

```
body {
  color: darkblue;
  font-family: Arial;
}
```

Todo el texto dentro del body:

☞ heredará el color y la fuente.

Propiedades que SÍ se heredan

✓ color
✓ font-family
✓ font-size
✓ text-align
✓ line-height

☞ relacionadas con texto.

⚠ Propiedades que NO se heredan

✗ margin
✗ padding
✗ border
✗ background
✗ width / height

☞ relacionadas con caja y layout.

Ejemplo práctico

```
body {
  color: blue;
}
p {
  color: black;
}
```

☞ el párrafo será negro porque se sobrescribe.

EL FLUJO NORMAL DEL DOCUMENTO

El flujo normal (normal flow)

☞ es la forma natural en que el navegador coloca los elementos.

Cómo funciona

Los elementos se colocan:

✓ de arriba hacia abajo
✓ de izquierda a derecha
✓ siguiendo el orden del HTML

Comportamiento por defecto

Elementos block → ocupan toda la línea

Elementos inline → fluyen dentro de la línea

☞ el diseño se construye naturalmente.

Ejemplo

<h1>Título</h1>

Párrafo

Otro párrafo

☞ se apilan verticalmente.

Cuándo se rompe el flujo

El flujo normal cambia cuando usamos:

✓ float
✓ position: absolute
✓ position: fixed
✓ display: none

☞ los elementos salen del flujo.

Idea profesional

☞ primero entender el flujo
☞ luego modificarlo

FLOAT: CREANDO COLUMNAS Y DISEÑOS CLÁSICOS

Antes de Flexbox y Grid, los layouts se construían con:

float

☞ permite que un elemento flote hacia un lado.

Cómo funciona

```
img {
  float: left;
}
```

El elemento:

✓ se mueve a la izquierda
✓ permite que el texto lo rodee

☞ **Valores principales**

float: left;

float: right;

float: none;

👀 **Ejemplo clásico (texto rodeando imagen)**

```
img {
  float: left;
  margin-right: 15px;
}
```

Crear columnas

```
.col {
  width: 50%;
  float: left;
}
```

☞ crea dos columnas.

⚠ **Problema común**

El contenedor padre "pierde" su altura 😲

Esto ocurre porque los elementos flotados salen del flujo normal.

CLEAR: SOLUCIONANDO EL PROBLEMA DEL FLOAT

Cuando usamos float, los elementos salen del flujo normal…

☞ y el contenedor puede colapsar (perder altura).

😱 Problema típico

```
.col {
  float: left;
}
```

El contenedor padre:

❌ no reconoce la altura
❌ otros elementos se superponen

✅ Solución: clear

La propiedad clear evita que elementos floten a su lado.

```
.footer {
  clear: both;
}
```

Valores:

✓ left
✓ right
✓ both

★ Solución profesional: clearfix

```
.container::after {
  content: "";
  display: block;
  clear: both;
}
```

☞ obliga al contenedor a reconocer su altura.

Cuándo usar clear

✓ después de columnas flotadas
✓ cuando el layout se rompe
✓ para restablecer el flujo

💡 Idea profesional

Float crea layouts.
Clear restaura el orden.

CENTRADO HORIZONTAL PROFESIONAL

Una de las preguntas más comunes:

☞ **¿Cómo centro un elemento correctamente?**

✅ **Centrar un contenedor (bloque)**

```
.container {
  width: 600px;
  margin: 0 auto;
}
```

 ¿Por qué funciona?

auto reparte el espacio a izquierda y derecha

el elemento queda centrado

⚠ **Requisitos**

✓ el elemento debe tener ancho definido
✓ debe ser block

👀 **Ejemplo práctico**

```
.card {
  width: 300px;
  margin: 40px auto;
}
```

☞ tarjeta centrada horizontalmente.

 Centrar texto

```
text-align: center;
```

☞ centra contenido inline dentro del contenedor.

 Diferencia importante

margin: auto → centra cajas
text-align: center → centra contenido

💡 **Tip profesional**

Si no se centra…

☞ revisa si definiste el width.

CENTRADO VERTICAL (SIN FLEXBOX 😎)

Centrar verticalmente ha sido históricamente uno de los retos más difíciles en CSS.

👉 Pero existen técnicas clásicas que funcionan.

MÉTODO 1 — Usar line-height (texto en una sola línea)

```
.box {
  height: 100px;
  line-height: 100px;
  text-align: center;
}
```

✓ centra verticalmente texto
✕ solo funciona con una línea

MÉTODO 2 — Usar position

```
.container {
  position: relative;
  height: 200px;
}
.box {
  position: absolute;
  top: 50%;
  transform: translateY(-50%);
}
```

✓ centra cualquier contenido
✓ método profesional clásico

MÉTODO 3 — vertical-align (elementos inline)

```
img {
  vertical-align: middle;
}
```

✓ útil para imágenes y texto
✕ no funciona en bloques normales

⚠ Error común

margin: auto NO centra verticalmente.

Idea profesional

Antes de Flexbox, esto era un arte 😄

CONTROLANDO EL DESBORDE (OVERFLOW)

A veces el contenido es más grande que su contenedor…

👉 y se desborda.

La propiedad:

overflow

controla qué ocurre en ese caso.

👇 **Ejemplo base**

```
.box {
  width: 200px;
  height: 100px;
}
```

Si el contenido excede el tamaño…

😮 se sale del contenedor.

🎛️ **Valores de overflow**

- **hidden**

Oculta el contenido extra.

overflow: hidden;

- **scroll**

Agrega barras de desplazamiento.

overflow: scroll;

- **auto**

Muestra scroll solo si es necesario.

overflow: auto;

👉 el más usado.

- **visible (por defecto)**

Permite que el contenido se salga.

PROPIEDAD Z-INDEX (CONTROL DE SUPERPOSICIÓN)

Cuando los elementos se superponen, necesitamos decidir:

☞ ¿cuál queda encima?

Para eso usamos:

z-index

Requisito IMPORTANTE

z-index solo funciona en elementos con:

✓ position: relative
✓ position: absolute
✓ position: fixed

Ejemplo

```
.box1 {
  position: relative;
  z-index: 1;
}
```

```
.box2 {
  position: relative;
  z-index: 2;
}
```

☞ .box2 quedará encima.

Cómo funciona

Mayor valor → más cerca del usuario
Menor valor → más al fondo

Usos reales

✓ menús desplegables
✓ modales
✓ tooltips
✓ superposición de imágenes

SOMBRAS Y PROFUNDIDAD VISUAL

Las sombras permiten crear **profundidad** y separar elementos visualmente.

☞ hacen que la interfaz se vea moderna y profesional.

box-shadow (sombras en cajas)

```
.card {
  box-shadow: 0 4px 10px rgba(0,0,0,0.2);
}
```

Sintaxis

box-shadow: horizontal vertical blur color;

Ejemplo:

horizontal → 0

vertical → 4px

blur → 10px

color → sombra

👀 **Resultado**

✓ efecto flotante
✓ sensación de profundidad
✓ separación visual

Sombra más suave

box-shadow: 0 2px 5px rgba(0,0,0,0.15);

☞ apariencia elegante.

text-shadow (sombras en texto)

```
h1 {
  text-shadow: 2px 2px 4px gray;
}
```

✓ mejora contraste
✓ útil sobre fondos claros

BORDES REDONDEADOS Y ESTÉTICA MODERNA

Los bordes redondeados ayudan a crear interfaces más suaves y modernas.

☞ son estándar en el diseño actual.

border-radius

```
.box {
  border-radius: 10px;
}
```

✓ suaviza las esquinas
✓ mejora la estética visual

Redondear completamente

```
.box {
  border-radius: 50%;
}
```

☞ crea círculos perfectos.

Redondear esquinas específicas

```
.box {
  border-radius: 10px 0 10px 0;
}
```

Orden:

superior-izq | superior-der | inferior-der | inferior-izq

Botones modernos

```
button {
  border-radius: 8px;
}
```

☞ apariencia profesional inmediata.

PSEUDOCLASES: INTERACCIÓN CON EL USUARIO

Las pseudo-clases permiten aplicar estilos cuando ocurre una acción del usuario.

👉 hacen las interfaces **interactivas**.

- **:hover — cuando el cursor pasa encima**

```
button:hover {
  background: blue;
}
```

✓ cambia al pasar el mouse
✓ mejora la interacción visual

- **:active — cuando se hace clic**

```
button:active {
  transform: scale(0.95);
}
```

👉 simula presión del botón.

- **:focus — cuando el elemento está seleccionado**

```
input:focus {
  border-color: blue;
}
```

✓ útil en formularios
✓ mejora accesibilidad
✓ indica selección activa

- **:visited — enlaces visitados**

```
a:visited {
  color: purple;
}
```

- **:link — enlaces no visitados**

```
a:link {
  color: blue;
}
```

🎯 **Orden correcto para enlaces**

- a:link
- a:visited
- a:hover
- a:active

👉 recuerda: **LVHA**

TRANSICIONES SUAVES (INTERACCIONES PROFESIONALES)

Las transiciones permiten que los cambios visuales ocurran **suavemente** en lugar de forma brusca.

☞ mejoran la experiencia del usuario.

Sintaxis básica

transition: propiedad duración;

Ejemplo

```
button {
  background: black;
  color: white;
  transition: background 0.3s;
}
button:hover {
  background: gray;
}
```

☞ el cambio será suave.

Sintaxis completa

transition: propiedad duración tipo retardo;

Ejemplo:

transition: all 0.3s ease;

Propiedades comunes

✓ color
✓ background
✓ transform
✓ opacity
✓ box-shadow

INTRODUCCIÓN A FLEXBOX (EL SIGUIENTE NIVEL)

Hasta ahora hemos construido layouts usando:

✓ flujo normal
✓ float
✓ position
✓ centrado manual

👉 pero estos métodos pueden ser complejos.

🚀 ¿Qué es Flexbox?

Flexbox es un sistema de diseño que permite:

✅ alinear elementos fácilmente
✅ distribuir espacio automáticamente
✅ crear layouts flexibles
✅ centrar elementos sin trucos

👉 fue creado para resolver problemas clásicos del diseño.

⚠️ Antes de Flexbox…

❌ centrar verticalmente era complicado
❌ crear columnas requería float
❌ alinear elementos era difícil

Con Flexbox podemos:

✓ centrar horizontal y verticalmente
✓ crear columnas y filas fácilmente
✓ controlar espacios entre elementos
✓ adaptar diseños dinámicamente

👀 Ejemplo simple

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

👉 activa el modo flexible.

💡 Idea clave

Flexbox no reemplaza CSS…

👉 lo potencia.

EJES EN FLEXBOX (LA CLAVE PARA ENTENDERLO)

Para dominar Flexbox debes entender sus **dos ejes**.

👉 TODO se alinea en relación a estos ejes.

Eje principal (Main Axis)

Es la dirección en la que los elementos se organizan.

Por defecto:

➡ horizontal (fila)

Eje transversal (Cross Axis)

Es perpendicular al eje principal.

Por defecto:

⬇ vertical

👀 **Visualización**

Si flex-direction: row:

Main axis → ➡
Cross axis → ⬇

⚠ **IMPORTANTE**

Si cambias la dirección…

los ejes cambian también 😲

👇 **Ejemplo**

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

Ahora:

Main axis → ⬇
Cross axis → ➡

Idea profesional

👉 No pienses en horizontal o vertical.

👉 Piensa en **main axis** y **cross axis**.

flex-direction: DEFINIENDO LA DIRECCIÓN

La propiedad **flex-direction** define cómo se organizan los elementos dentro del contenedor flexible.

- **Valores principales**

➡ row (por defecto)

Elementos en fila horizontal.

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

⬅ row-reverse

Fila horizontal invertida.

```
flex-direction: row-reverse;
```

⬇ column

Elementos en columna vertical.

```
flex-direction: column;
}
```

⬆ column-reverse

Columna invertida.

```
flex-direction: column-reverse;
```

👀 Qué cambia

✓ dirección del contenido
✓ orden visual
✓ orientación del eje principal

justify-content: ALINEACIÓN EN EL EJE PRINCIPAL

Ahora que conocemos el eje principal (main axis)…

☞ usamos **justify-content** para alinear los elementos a lo largo de ese eje.

- **Valores principales**

◆ **flex-start (por defecto)**

Alinea al inicio.

justify-content: flex-start;

◆**center**

Centra los elementos.

justify-content: center;

◆ **flex-end**

Alinea al final.

justify-content: flex-end;

◆ **space-between**

Espacio entre elementos.

justify-content: space-between;

☞ extremos pegados, espacio interno.

◆ **space-around**

Espacio alrededor.

justify-content: space-around;

◆ **space-evenly**

Espacio igual entre todos.

justify-content: space-evenly;

align-items: ALINEACIÓN EN EL EJE TRANSVERSAL

Ya sabemos alinear en el eje principal…

Ahora alineamos en el **eje transversal (cross axis)**.

☞ usamos **align-items**.

- **Valores principales**

◆ **stretch (por defecto)**

Los elementos se estiran para llenar el contenedor.

align-items: stretch;

◆ **flex-start**

Alinea al inicio del eje transversal.

align-items: flex-start;

◆ **center**

Centra los elementos.

align-items: center;

☞ centra verticalmente si es row.

◆ **flex-end**

Alinea al final.

align-items: flex-end;

◆ **baseline**

Alinea según la línea base del texto.

align-items: baseline;

gap: ESPACIADO ENTRE ELEMENTOS FLEX

Antes, para separar elementos usábamos:

❌ margin individual
❌ trucos con :last-child
❌ cálculos manuales

👉 ahora usamos:

gap

```
.container {
  display: flex;
  gap: 20px;
}
```

✓ separación uniforme
✓ más limpio
✓ fácil de mantener

¿Qué hace?

Define el espacio **entre los elementos flex**.

🎯 **Espacio horizontal y vertical**

gap: 10px 20px;

👉 10px entre filas
👉 20px entre columnas

flex-wrap: CONTROLANDO EL SALTO DE LÍNEA

Por defecto, los elementos flex intentan mantenerse en **una sola línea**.

👉 Esto puede causar desbordes 😱

Comportamiento por defecto

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

✓ los elementos se comprimen
✓ NO saltan de línea

✅ **Permitir salto de línea**

flex-wrap: wrap;

👉 los elementos bajan a la siguiente fila si no caben.

Evitar salto

flex-wrap: nowrap;

👉 comportamiento por defecto.

Wrap invertido

flex-wrap: wrap-reverse;

👉 los elementos bajan pero en orden invertido.

flex-grow, flex-shrink y flex-basis (CÓMO CRECEN LOS ELEMENTOS)

Flexbox permite que los elementos **crezcan o se reduzcan** para adaptarse al espacio disponible.

☞ estas propiedades se aplican a los **elementos hijos**.

- **flex-grow — cuánto puede crecer**

Define cuánto espacio extra puede ocupar un elemento.

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

✓ todos crecen por igual
✓ ocupan el espacio disponible

Ejemplo

```
.item1 { flex-grow: 1; }
.item2 { flex-grow: 2; }
```

☞ item2 crecerá el doble.

- **flex-shrink — cuánto puede reducirse**

Define cuánto puede encogerse.

```
.item {
  flex-shrink: 1;
}
```

☞ evita desbordes.

- **flex-basis — tamaño base inicial**

Define el tamaño inicial antes de crecer o reducirse.

```
.item {
  flex-basis: 200px;
}
```

☞ punto de partida.

max-width, min-width Y CONTROL RESPONSIVO

Estas propiedades NO pertenecen a Flexbox.

👉 Son parte del **CSS adaptable** y se usan junto con Flexbox para crear diseños fluidos.

max-width — ancho máximo permitido

```
.container {
  max-width: 1200px;
}
```

✓ evita que el contenido se estire demasiado
✓ mantiene legibilidad
✓ diseño profesional

👉 muy usado en contenedores centrales.

min-width — ancho mínimo permitido

```
.card {
  min-width: 250px;
}
```

✓ evita que el elemento se haga demasiado pequeño
✓ protege el diseño

Ejemplo combinado

```
.container {
  width: 90%;
  max-width: 1200px;
}
```

👉 fluido en pantallas pequeñas
👉 limitado en pantallas grandes

max-height y min-height

Funcionan igual pero en altura.

```
.box {
  min-height: 200px;
}
```

CIERRE DE LA CLASE

🚀 **Hoy dimos el salto a CSS profesional**

Ahora puedes:

✅ controlar el flujo visual
✅ alinear elementos con precisión
✅ crear layouts flexibles
✅ centrar elementos sin trucos
✅ distribuir espacio dinámicamente
✅ construir interfaces modernas

🎯 **Dominamos hoy**

✓ Box Model avanzado
✓ posicionamiento y flujo
✓ especificidad y cascada
✓ interacción con pseudo-clases
✓ Flexbox y alineación moderna
✓ control adaptable con max/min

Lo más importante

👉 CSS no es memorizar propiedades.
👉 CSS es entender **cómo se comporta el diseño**.

🔥 **Próxima clase**

👉 Profundizaremos en:

INTRODUCCIÓN A SASS (CSS PROFESIONAL)

TAREA “ESPECIAL” 😎🎭

🎉 **TAREA PARA EL GRUPO**

Para fomentar la integración del curso y fortalecer el espíritu universitario…

👉 quiero que manden al grupo:

📸 **SUS FOTOS MÁS VERGONZOSAS DEL CARNAVAL** 🎭

😈 **Requisitos**

✔ con espuma hasta el alma
✔ disfraz cuestionable
✔ cara de “¿por qué hice esto?”
✔ evidencia de supervivencia carnavalera

▶

🏆 **Puntos extra**

⭐ si no te reconocen
⭐ si tu familia no sabe que existió esa foto
⭐ si tus amigos te etiquetaron sin permiso

⚠ **Importante**

❌ no fotos comprometedoras
❌ no contenido inapropiado
❌ no culpar al profesor después

👉 solo diversión sana 😉