

LUCAS EGA ACADEMY

CSS3

STYLES • ANIMATIONS • RESPONSIVE DESIGN • MODERN WEB



GREEN
#1A5C3A



GOLD
#CFA845



BASE
#0A00A10



ACCENT
#22B849N

```
.course {  
  display: grid; padding: 1.2rem;  
  border: 1px solid #1a5c3a; border-radius: 8px;  
  background: rgba(10,40,28,0.6);  
}  
.hum-hover { transition: all 0.3s ease; transform: scale(1.05);  
  box-shadow: 0 0 12px #cfa845; }  
}
```

Curso Completo de CSS3

© 2026 Esteban Gutiérrez

Colibrí Editorial EGA

Hecho en México

lucasegaacademy.com

Curso Completo de CSS3

Autor: Esteban Gutiérrez

lucasegaacademy.com

*Guía completa con explicaciones extensas por unidad,
prácticas y examen final.*

ÍNDICE

PRÓLOGO	4
UNIDAD 1: FUNDAMENTOS DE CSS	6
UNIDAD 2: COLOR, TEXTO Y FONDOS	23
UNIDAD 3: BORDES, ESPACIADO Y EL MODELO DE CAJA	40
UNIDAD 4: TAMAÑOS, DISPLAY Y POSICIONAMIENTO	55
UNIDAD 5: FLEXBOX Y PRIMEROS PASOS CON GRID	71
UNIDAD 6: GRID AVANZADO, PSEUDOCASES Y PSEUDO-ELEMENTOS ..	88
UNIDAD 7: MOVIMIENTO - TRANSICIONES, ANIMACIONES Y TRANSFORMA... ..	106
UNIDAD 8: SOMBRAS, LISTAS, TABLAS Y FORMULARIOS	122
UNIDAD 9: DISEÑO RESPONSIVE, UNIDADES Y VARIABLES	143
UNIDAD 10: PROYECTOS INTEGRADORES Y GRADUACIÓN	161
APENDICE A: REFERENCIA RÁPIDA DE PROPIEDADES CSS	230
APENDICE B: GLOSARIO	235
APENDICE C: PROYECTO FINAL INTEGRADOR	238
APENDICE D: GUIA DE DEPURACION RÁPIDA	240
APENDICE E: BANCO DE 30 MINI-RETOS	242
APENDICE F: PLANTILLAS LISTAS PARA COPIAR	249
APENDICE G: PREGUNTAS FRECUENTES	258
EXAMEN FINAL	263

PRÓLOGO

Bienvenido al libro "Curso Completo de CSS3" de Lucas EGA Academy (lucasegaacademy.com).

Si HTML es el esqueleto de una página web -los huesos, la estructura, los títulos, los párrafos y los botones-, entonces CSS es todo lo que lo viste, lo peina, lo maquilla y le da personalidad. Sin CSS, la web sería un documento plano, gris y aburrido. Con CSS, la misma página se convierte en un sitio profesional que puede enamorar a primera vista: colores vibrantes, tipografías elegantes, botones que reaccionan al pasar el mouse, tarjetas con sombra y animaciones que hacen que todo se sienta vivo.

Este libro es la versión escrita y extendida del curso de 30 clases de CSS3 de la academia. Aquí no solo aprenderás qué hace cada propiedad, sino por qué existe, cómo se combina con otras, qué errores comete la mayoría de la gente al empezar y cómo se usa en proyectos reales del día a día.

Cómo está organizado este libro

El libro está dividido en 10 unidades. Cada unidad agrupa 3 clases del curso en video y sigue siempre la misma estructura:

1. Una introducción extensa que te pone en contexto y te explica por qué el tema importa.
2. Una explicación detallada con analogías del mundo real, errores comunes que debes evitar y buenas prácticas de la industria.
3. Ejemplos paso a paso con código real que puedes copiar y probar en tu computadora.
4. Una sección de práctica con al menos 5 ejercicios y sus soluciones explicadas paso a paso.
5. Un resumen de la unidad con los puntos clave.

Al final encontrarás el EXAMEN FINAL: 50 preguntas de opción múltiple, verdadero/falso y preguntas abiertas con código, junto con todas las respuestas explicadas para que puedas autoevaluarte.

Cómo estudiar este libro

Te recomiendo este orden de trabajo para sacarle el máximo provecho:

Primero, lee la explicación detallada de cada unidad sin prisa. No te brinques las analogías: están ahí para que el concepto se te quede grabado, no solo memorizado. Segundo, copia los ejemplos a mano en tu editor de código y ejecútalos en tu navegador. La programación no se aprende leyendo, se aprende escribiendo y rompiendo cosas. Tercero, haz los ejercicios de práctica sin ver la solución. Si te atoras, regresa a la explicación, vuelve a intentarlo y solo al final compara con la solución propuesta. Cuarto, al terminar las 10 unidades, haz el examen final en una sentada, como si fuera real. Si sacas menos del 80 por ciento, repite las unidades donde fallaste y vuelve a intentarlo.

Qué necesitas antes de empezar

Para trabajar con este libro necesitas cuatro cosas: un editor de código (te recomiendo Visual Studio Code, que es gratis), un navegador moderno (Chrome, Edge o Firefox), una carpeta en tu computadora para guardar tus prácticas y, sobre todo, ganas de experimentar. Todo el código de este libro funciona sin instalar nada más: CSS se ejecuta directamente en el navegador.

Una nota importante: este libro se escribió con la idea de que ya tengas nociones básicas de HTML. Si todavía no sabes qué es una etiqueta, un atributo o cómo armar una página con encabezados y párrafos, te recomiendo pasar primero por el libro "Curso Completo de HTML5" de esta misma academia. Dicho esto, si ya entiendes lo básico, puedes empezar aquí sin problema: CSS es un lenguaje amable con los principiantes.

Una última cosa antes de empezar

Nadie nació sabiendo CSS. Todos los diseñadores que admiras empezaron poniendo un color de fondo que no les gustó, centrando un div que se negaba a centrarse y peleando con un menú que se veía bien en la computadora pero se rompía en el celular. La diferencia entre ellos y quien se rinde es una sola: siguieron practicando. Este libro te da el mapa, pero el camino lo recorres tú, línea de código por línea de código.

¡Vamos a darle estilo a la web!

UNIDAD 1: FUNDAMENTOS DE CSS

Introducción a la unidad

Imagina que vas a abrir una taquería. Rentas el local, pones las mesas, contratas al taquero y cuelgas un letrero que dice "TACOS". Eso es HTML: la estructura, el contenido, lo que existe. Pero si dejas el local con paredes grises, focos pelones y las mesas de plástico sin mantel, nadie va a querer sentarse aunque los tacos sean buenísimos. CSS es lo que convierte ese local en un lugar al que la gente quiere entrar: el color de las paredes, la iluminación cálida, los manteles, el diseño del menú, la música de fondo. Es exactamente el mismo negocio, pero se siente completamente distinto.

CSS son las siglas de "Cascading Style Sheets", que en español se traduce como "Hojas de Estilo en Cascada". Cada palabra importa. "Hojas de estilo" porque es un documento (o una sección de código) dedicado únicamente a describir cómo se ve todo. "En cascada" porque las reglas de estilo fluyen y se combinan siguiendo un orden de prioridad: si dos reglas compiten por darle estilo al mismo elemento, gana la más específica o la que llega después, como el agua que cae en cascada de un nivel a otro.

En esta unidad vas a construir la base de todo lo demás: qué es CSS y por qué cambió la historia de la web, las tres formas de incluir estilos en tu página y la más importante de todas las habilidades en CSS: los selectores, es decir, cómo apuntar con precisión al elemento exacto que quieres estilizar.

Cuando termines esta unidad entenderás por qué los profesionales separan el contenido (HTML) del diseño (CSS), sabrás cuándo usar cada forma de incluir estilos y podrás seleccionar cualquier elemento de una página usando etiquetas, clases e identificadores.

Explicación detallada

Qué es CSS y por qué existe

Para entender CSS hay que viajar a los inicios de la web. En los años 90, las páginas web se escribían solo con HTML, y el HTML de esa época incluía etiquetas de presentación mezcladas con el contenido: `<font>` para cambiar la letra, el atributo `bgcolor` para pintar fondos, `<center>` para centrar texto. Funcionaba, pero era un desastre de mantenimiento. Si

tenías un sitio con 100 páginas y querías cambiar el color de todos los títulos de azul a verde, tenías que abrir las 100 páginas una por una y editar cada etiqueta. Era lento, propenso a errores y absolutamente impráctico para sitios grandes.

CSS nació para resolver ese problema con una idea simple pero poderosa: separar el contenido de la presentación. El HTML dice qué es cada cosa ("esto es un título", "esto es un párrafo", "esto es un botón") y el CSS dice cómo se ve cada cosa ("los títulos son azules y grandes", "los párrafos tienen esta fuente", "los botones son redondos y naranjas"). Con esa separación, cambiar el color de todos los títulos de un sitio de 100 páginas se reduce a editar una sola línea en un solo archivo de CSS.

Esta separación trae tres beneficios gigantes. Primero, mantenimiento: cambias el diseño de todo el sitio desde un lugar central. Segundo, consistencia: todos los elementos iguales se ven iguales automáticamente, sin que tengas que acordarte de copiar el mismo estilo cien veces. Tercero, accesibilidad y rendimiento: el HTML queda limpio y semántico, lo que ayuda a los lectores de pantalla y a los buscadores, y el navegador puede guardar en caché el archivo CSS para que las páginas carguen más rápido.

CSS3 es la versión actual del lenguaje. A diferencia de versiones anteriores, CSS3 se desarrolló en módulos independientes (selectores, fondos, animaciones, flexbox, grid, etc.), lo que permitió que los navegadores adoptaran las novedades poco a poco. Todo lo que aprenderás en este libro es CSS3 moderno, el que se usa hoy en la industria.

Cómo piensa CSS: reglas, selectores, propiedades y valores

Todo CSS está hecho de reglas. Una regla tiene dos partes: el selector y el bloque de declaraciones.

```
p {  
  color: blue;  
  font-size: 18px;  
}
```

En este ejemplo, `p` es el selector: dice "esta regla aplica a todos los párrafos". Dentro de las llaves van las declaraciones, y cada declaración tiene una propiedad y un valor separados por dos puntos: `color: blue;` significa "la propiedad color toma el valor blue". Cada declaración termina con punto y coma. Si olvidas el punto y coma, la siguiente declaración puede romperse, y ese es uno de los errores más comunes de los principiantes.

Piensa en una regla CSS como una orden que le das a un pintor: "A todos los cuartos que sean recámaras (selector), píntales las paredes de azul (propiedad color: valor blue) y ponles lámparas de 18 watts (propiedad font-size: valor 18px)". El selector elige a quién, las declaraciones dicen qué hacer.

Las tres formas de incluir CSS

Hay tres formas de meter CSS en una página web, y es crucial que entiendas las tres porque te las vas a encontrar en código ajeno, aunque en tu propio trabajo casi siempre usarás solo una.

La primera forma es el estilo en línea, usando el atributo `style` directamente en la etiqueta HTML:

```
<p style="color: red; font-size: 20px;">Este párrafo es rojo y grande.</p>
```

Ventaja: es inmediato y tiene la máxima prioridad (casi siempre gana sobre las demás formas).
Desventaja: no se puede reutilizar. Si tienes 50 párrafos rojos y quieres cambiarlos a azul, tienes que editar los 50. Es como pintar cada cuarto con una brocha distinta en vez de usar la misma cubeta de pintura.

La segunda forma es el bloque `<style>` dentro del `<head>` del documento HTML:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    p {
      color: blue;
      font-size: 18px;
    }
  </style>
</head>
<body>
  <p>Todos los párrafos de esta página serán azules.</p>
  <p>Este también.</p>
</body>
</html>
```

Ventaja: una sola regla afecta a todos los elementos de la página, así que ya hay reutilización.

Desventaja: el estilo vive pegado a esa página. Si tienes 20 páginas, tendrías que copiar el bloque `<style>` en las 20, y cualquier cambio habría que repetirlo 20 veces.

La tercera forma es el archivo externo, y es la forma profesional:

```
<link rel="stylesheet" href="estilos.css">
```

Esta línea va en el `<head>` y le dice al navegador: "ve a buscar el archivo `estilos.css` y aplica sus reglas a esta página". El archivo `estilos.css` contiene solo CSS, sin HTML:

```
/* estilos.css */
p {
  color: green;
  font-size: 18px;
}
```

Ventaja: un solo archivo controla el diseño de todo el sitio. Cambias una línea y se actualizan las 100 páginas a la vez. Además el navegador guarda el archivo en caché, así que las páginas cargan más rápido. Desventaja: requiere que el archivo exista en la ruta correcta; si escribes mal el nombre o la carpeta, los estilos simplemente no se aplican y la página se ve "desnuda", lo cual es también una pista de diagnóstico muy útil.

Regla de oro: en tus proyectos usa siempre el archivo externo. Usa el bloque `<style>` solo para pruebas rápidas o páginas de un solo archivo. Usa el estilo en línea casi nunca, salvo para pruebas momentáneas o para estilos generados dinámicamente con JavaScript.

Un detalle importante sobre el elemento `<link>`: el atributo `rel="stylesheet"` le dice al navegador qué tipo de relación tiene el archivo con la página, y `href` indica la ruta. Si tu archivo CSS está en una carpeta llamada `css`, la ruta sería `href="css/estilos.css"`. Si está en la misma carpeta que el HTML, basta con `href="estilos.css"`. Los errores de ruta son la causa número uno de "mi CSS no funciona".

Selectores: el arte de apuntar con precisión

Los selectores son la parte más importante de CSS. Un selector responde a la pregunta: "¿a qué elementos les aplico esta regla?". Dominar los selectores es dominar CSS, porque todo lo demás (colores, tamaños, posiciones) son solo propiedades que ya conoces de nombre; lo difícil es aplicarlas exactamente donde quieres sin afectar lo que no quieres.

Los tres selectores básicos son: selector de etiqueta, selector de clase y selector de identificador (id).

El selector de etiqueta apunta a todos los elementos de ese tipo. Es el más amplio:

```
h1 {  
  color: navy;  
}
```

Esta regla pinta de azul marino TODOS los `<h1>` de la página. Úsalo para estilos generales y consistentes: "todos los párrafos se ven así, todos los títulos de nivel 2 se ven así". Es como una regla de la casa: aplica a todos por igual.

El selector de clase apunta a los elementos que tengan ese atributo `class`. En el CSS se escribe con un punto antes del nombre:

```
/* CSS */  
.destacado {  
  background-color: yellow;  
  font-weight: bold;  
}
```

```
<!-- HTML -->  
<p class="destacado">Este párrafo está destacado.</p>  
<p>Este párrafo es normal.</p>  
<div class="destacado">Este div también está destacado.</div>
```

Las clases son la herramienta más usada en CSS profesional, y hay tres razones. Primera: se pueden reutilizar. La misma clase puede aplicarse a 100 elementos distintos. Segunda: un elemento puede tener varias clases separadas por espacios (`class="tarjeta destacada grande"`), lo que permite componer estilos como piezas de LEGO. Tercera: funcionan con cualquier etiqueta, así que no te limitan.

Piensa en las clases como etiquetas adhesivas de colores que les pegas a las cosas: "todo lo que tenga la etiqueta amarilla se ve así". Puedes pegar la misma etiqueta en muchas cosas y una cosa puede tener varias etiquetas.

El selector de identificador (id) apunta al elemento único que tenga ese `id`. En el CSS se escribe con un numeral (#) antes del nombre:

```
/* CSS */
#menu-principal {
  background-color: #333;
  color: white;
}
```

```
<!-- HTML -->
<nav id="menu-principal">...</nav>
```

La regla de HTML es que cada `id` debe ser único en toda la página: no puede haber dos elementos con `id="menu-principal"`. Por eso los `ids` se usan para cosas que aparecen una sola vez: el menú principal, el encabezado del sitio, el pie de página, un formulario de contacto específico. En la práctica moderna, los `ids` se usan más en JavaScript (para encontrar un elemento exacto) que en CSS, donde las clases son preferidas por ser reutilizables. Pero debes conocerlos porque los verás en mucho código.

Analogía de los selectores: la escuela

Imagina una escuela. El selector de etiqueta es como decir "todos los alumnos de tercero": apuntas a un grupo completo por su tipo. El selector de clase es como decir "todos los que traen playera roja hoy": apuntas a quienes comparten una característica, sin importar de qué grupo sean, y mañana la playera puede ser de otro color. El selector de `id` es como decir "María González del grupo 3B": apuntas a una persona única e irrepetible.

La cascada y la especificidad: quién gana cuando hay conflicto

"Cascading" (cascada) significa que cuando varias reglas aplican al mismo elemento, el navegador decide cuál gana siguiendo un orden. Este es uno de los temas que más confunde a los principiantes, así que vamos con calma.

Hay tres fuentes de estilos en orden de prioridad, de menor a mayor: primero los estilos del navegador (cada navegador trae estilos por defecto: por ejemplo, los `<h1>` son grandes y con negritas aunque tú no escribas nada), después tus hojas de estilo (el archivo externo, el bloque `<style>`), y al final los estilos en línea (el atributo `style`). El estilo en línea casi siempre gana.

Pero dentro de tus propias hojas de estilo también hay conflictos. Mira este ejemplo:

```
p {  
  color: blue;  
}  
  
.destacado {  
  color: red;  
}  
  
#unico {  
  color: green;  
}
```

```
<p id="unico" class="destacado">¿De qué color soy?</p>
```

Las tres reglas aplican al mismo párrafo. ¿Quién gana? Aquí entra la especificidad, que es un sistema de puntos. Cada tipo de selector vale puntos distintos: el selector de etiqueta vale 1 punto, el selector de clase vale 10 puntos, el selector de id vale 100 puntos. El estilo en línea vale 1000 puntos. Gana la regla con más puntos. En el ejemplo: `p` tiene 1 punto, `.destacado` tiene 10 puntos, `#unico` tiene 100 puntos. Gana el id: el párrafo será verde.

Si dos reglas tienen los mismos puntos, gana la que aparece después en el código (la última escrita). Por eso el orden de tus reglas importa: las reglas más generales van primero y las más específicas después.

Existe además la declaración `!important`, que rompe todas las reglas y gana automáticamente:

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

Suena tentador, pero es una mala práctica usarla para resolver conflictos de especificidad, porque convierte tu CSS en una guerra de `!important` donde nadie entiende qué está pasando. Úsala solo en casos excepcionales, como sobrescribir estilos de una librería externa que no puedes modificar. Un buen CSS casi nunca necesita `!important`.

Errores comunes de principiantes (y cómo evitarlos)

Error 1: olvidar el punto y coma. Cada declaración termina con `;`. Si lo olvidas, la siguiente declaración se puede "pegar" a la anterior y dejar de funcionar. Acostúmbrate a ponerlo siempre, incluso en la última declaración del bloque.

Error 2: confundir clase con id. Recuerda: en el HTML escribes `class="nombre"` y en el CSS lo seleccionas con `.nombre` (punto). En el HTML escribes `id="nombre"` y en el CSS lo seleccionas con `#nombre` (numeral). Si en el CSS pones `.menu` pero en el HTML el elemento tiene `id="menu"`, no va a funcionar.

Error 3: repetir el mismo id en varios elementos. El navegador aplicará el estilo a todos (los navegadores son tolerantes), pero tu HTML será inválido y JavaScript se comportará de forma impredecible. Un id, un solo elemento.

Error 4: escribir mal la ruta del archivo CSS en el `<link>`. Si tus estilos no se aplican para nada, lo primero que debes revisar es que el `href` apunte al archivo correcto. Abre las herramientas de desarrollo del navegador (F12) y mira la pestaña de red o consola: ahí verás si el archivo CSS no se encontró (error 404).

Error 5: no entender por qué un estilo "no se aplica". El 90 por ciento de las veces es un problema de especificidad: otra regla con más puntos está ganando. Usa las herramientas de desarrollo (F12, pestaña de elementos) para ver qué reglas aplican a un elemento y cuáles están tachadas porque perdieron la batalla. Esa vista es tu mejor amiga para depurar CSS.

Buenas prácticas desde el día uno

Primera: un solo archivo CSS externo por proyecto (o unos pocos bien organizados), nunca estilos regados en atributos `style` por todo el HTML. Segunda: usa nombres de clase descriptivos y en minúsculas, con guiones para separar palabras: `.menu-principal`, `.tarjeta-producto`, `.boton-comprar`. Evita nombres como `.rojo` o `.grande` que describen la apariencia y no el propósito: si mañana el botón deja de ser rojo, el nombre `.rojo` se vuelve una mentira. Tercera: organiza tu CSS de lo general a lo específico: primero estilos base (body, títulos, párrafos), después componentes (botones, tarjetas, menús) y al final ajustes específicos. Cuarta: comenta tu código con `/* comentario */` para marcar secciones. Tu yo del futuro te lo agradecerá.

Ejemplos paso a paso

Ejemplo 1: Mi primera hoja de estilos externa

Vamos a crear una página completa con archivo CSS externo. Crea dos archivos en la misma carpeta: index.html y estilos.css.

Archivo index.html:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Mi primera página con CSS</title>
  <link rel="stylesheet" href="estilos.css">
</head>
<body>
  <h1>Bienvenidos a mi sitio</h1>
  <p>Este es mi primer párrafo con estilo.</p>
  <p>Este es el segundo párrafo.</p>
</body>
</html>
```

Archivo estilos.css:

```
body {
  background-color: #f0f0f0;
  font-family: Arial, sans-serif;
}

h1 {
  color: darkblue;
  text-align: center;
}

p {
  color: #333333;
  font-size: 18px;
}
```

Paso a paso: primero creamos el HTML con la estructura básica y enlazamos la hoja con `<link>`. Después escribimos tres reglas: una para el `body` (fondo gris claro y fuente Arial), una para el `h1` (azul oscuro y centrado) y una para los párrafos (gris oscuro y tamaño 18px). Al abrir `index.html` en el navegador verás la página con estilo. Prueba cambiar `darkblue` por `red` en el CSS, guarda y recarga: el cambio es instantáneo.

Ejemplo 2: Clases en acción

Ahora agreguemos clases para diferenciar elementos del mismo tipo. Modifica el HTML:

```
<h1 class="titulo-principal">Bienvenidos a mi sitio</h1>
<h1>Otro título normal</h1>
<p class="destacado">Este párrafo es importante.</p>
<p>Este párrafo es normal.</p>
<p class="destacado advertencia">Este tiene dos clases.</p>
```

Y agrega al CSS:

```
.titulo-principal {
  color: darkgreen;
  font-size: 40px;
  border-bottom: 3px solid darkgreen;
}

.destacado {
  background-color: #fff3cd;
  padding: 10px;
  border-left: 4px solid orange;
}

.advertencia {
  color: #856404;
  font-weight: bold;
}
```

Observa el último párrafo: tiene `class="destacado advertencia"`, así que recibe los estilos de ambas clases. El fondo amarillo claro y el borde naranja vienen de `.destacado`, y el color de texto y las negritas vienen de `.advertencia`. Así se componen estilos como piezas de LEGO.

Ejemplo 3: Id único y especificidad en la práctica

```
<nav id="menu-principal">
  <a href="#">Inicio</a>
  <a href="#">Cursos</a>
  <a href="#">Contacto</a>
</nav>
<p>Texto normal de la página.</p>
```

```
#menu-principal {
  background-color: #222222;
  padding: 15px;
}

#menu-principal a {
  color: white;
  margin-right: 20px;
  text-decoration: none;
}

p {
  color: gray;
}
```

Fíjate en el selector `#menu-principal a`: significa "los enlaces que están dentro del elemento con id `menu-principal`". Este tipo de selector combinado (descendiente) lo usarás constantemente. Y nota que aunque `p` dice gris, los enlaces del menú son blancos porque la regla del id tiene más especificidad... bueno, en realidad los enlaces ni siquiera son párrafos, pero el punto es el mismo: la regla más específica gana.

Ejemplo 4: Probando la cascada

Crea este HTML y CSS para ver la cascada en acción:

```
<p class="texto" id="parrafo1" style="color: purple;">¿De qué color soy?</p>
```

```
p {  
  color: blue;  
}  
  
.texto {  
  color: red;  
}  
  
#parrafo1 {  
  color: green;  
}
```

Respuesta: el párrafo será morado (purple), porque el estilo en línea (1000 puntos) le gana al id (100), a la clase (10) y a la etiqueta (1). Si quitas el atributo `style`, será verde (gana el id). Si quitas también el id, será rojo (gana la clase). Si quitas la clase, será azul (la etiqueta). Haz la prueba quitando una cosa a la vez: es el mejor ejercicio para entender la especificidad de una vez por todas.

Ejemplo 5: Comentarios y organización

Un archivo CSS profesional se ve así de organizado:

```

/* =====
ESTILOS GENERALES
===== */
body {
  font-family: Arial, sans-serif;
  background-color: #fafafa;
  color: #333;
}

h1, h2, h3 {
  color: #1a1a1a;
}

/* =====
COMPONENTES
===== */

/* Botones */
.boton {
  padding: 10px 20px;
  border-radius: 5px;
}

/* Tarjetas */
.tarjeta {
  background: white;
  padding: 20px;
}

```

Fíjate en el selector `h1, h2, h3`: la coma significa "aplica esta regla a h1 Y a h2 Y a h3". Es una forma de no repetir la misma regla tres veces. Los comentarios con `/* */` marcan secciones y no afectan nada: el navegador los ignora.

Práctica

Ejercicio 1

Crea una página HTML con un archivo CSS externo que cumpla: el fondo de la página es azul muy claro (`#e6f2ff`), todos los títulos `h1` son azul marino y están centrados, todos los párrafos tienen fuente Verdana y tamaño 16px.

SOLUCIÓN: Primero el HTML con el `<link>` en el head apuntando a `estilos.css`. Después el CSS con tres reglas: `body {background-color: #e6f2ff;}, h1 {color: navy; text-align: center;}` y `p {font-family: Verdana, sans-serif; font-size: 16px;}`. Lo importante aquí es practicar la conexión entre los dos archivos: si no ves los estilos, revisa que ambos archivos estén en la misma carpeta y que el `href` esté bien escrito.

Ejercicio 2

Dado este HTML, escribe el CSS usando clases para que el resultado sea: el título con clase "oferta" en rojo y tamaño 32px, el párrafo con clase "precio" en verde, negritas y tamaño 24px, y el párrafo normal en gris.

```
<h1 class="oferta">¡Oferta especial!</h1>
<p class="precio">$299</p>
<p>Válido hasta agotar existencias.</p>
```

SOLUCIÓN:

```
.oferta {
  color: red;
  font-size: 32px;
}

.precio {
  color: green;
  font-weight: bold;
  font-size: 24px;
}

p {
  color: gray;
}
```

Nota que la regla `p {color: gray;}` aplica a todos los párrafos, pero el párrafo con clase "precio" será verde porque la clase (10 puntos) le gana a la etiqueta (1 punto). Este es el patrón más común del CSS real: estilos generales con etiquetas y excepciones con clases.

Ejercicio 3

Corrige los errores en este CSS para que funcione:

```
.titulo-principal
  color: blue
  font-size: 30px;

#menu {
  background-color: black
  color: white;
}
```

SOLUCIÓN: Hay tres errores. Primero, al selector `.titulo-principal` le faltan las llaves de apertura y cierre. Segundo, a `color: blue` le falta el punto y coma al final. Tercero, a `background-color: black` le falta el punto y coma, lo que hace que `color: white` se pegue y se rompa. El CSS corregido:

```
.titulo-principal {
  color: blue;
  font-size: 30px;
}

#menu {
  background-color: black;
  color: white;
}
```

Ejercicio 4

Explica qué color tendrá cada párrafo y por qué:

```
<p class="a">Uno</p>
<p class="a b">Dos</p>
<p id="c" class="a b">Tres</p>
```

```
.a { color: red; }
.b { color: blue; }
#c { color: green; }
p { color: black; }
```

SOLUCIÓN: El párrafo "Uno" será rojo: solo aplica a (10 puntos) y p (1 punto); gana la clase. El párrafo "Dos" será azul: aplican a y b, ambas con 10 puntos; al empatar, gana la última escrita, que es b. El párrafo "Tres" será verde: el id #c tiene 100 puntos y le gana a todo. Este ejercicio resume la especificidad completa en cuatro líneas.

Ejercicio 5

Escribe el HTML y CSS para un menú de navegación usando un id "navegacion": fondo gris oscuro (#333), enlaces blancos sin subrayado, con 15px de separación entre ellos.

SOLUCIÓN:

```
<nav id="navegacion">
  <a href="#">Inicio</a>
  <a href="#">Cursos</a>
  <a href="#">Blog</a>
  <a href="#">Contacto</a>
</nav>
```

```
#navegacion {
  background-color: #333;
  padding: 15px;
}

#navegacion a {
  color: white;
  text-decoration: none;
  margin-right: 15px;
}
```

El selector #navegacion a es un selector descendiente: selecciona los <a> que están dentro del elemento con ese id. La propiedad text-decoration: none quita el subrayado típico de los enlaces.

Ejercicio 6

Crea tres cajas (divs) usando clases: todas comparten la clase "caja" (ancho 200px, alto 100px, margen 10px) pero cada una tiene además una clase de color distinta ("roja", "verde", "azul") que define su color de fondo.

SOLUCIÓN:

```
<div class="caja roja"></div>
<div class="caja verde"></div>
<div class="caja azul"></div>
```

```
.caja {
  width: 200px;
  height: 100px;
  margin: 10px;
}

.roja { background-color: red; }
.verde { background-color: green; }
.azul { background-color: blue; }
```

Este es el patrón de "clase base + clase modificadora" que usan los frameworks profesionales: la clase base define lo común y las modificadoras definen lo que cambia. Es limpio, reutilizable y fácil de mantener.

Resumen de la unidad

En esta unidad aprendiste que CSS es el lenguaje que controla la presentación de las páginas web, separando el contenido (HTML) del diseño para que los sitios sean fáciles de mantener, consistentes y rápidos. Conociste las tres formas de incluir CSS -estilo en línea, bloque `<style>` y archivo externo- y por qué el archivo externo es la forma profesional. Dominaste los tres selectores básicos: etiqueta (todos los de ese tipo), clase (los que comparten una característica, con punto) e id (uno único, con numeral). Entendiste la cascada y la especificidad: los puntos deciden quién gana (etiqueta 1, clase 10, id 100, en línea 1000) y en caso de empate gana la última regla escrita. También viste los errores más comunes -punto y coma olvidados, confundir clase con id, rutas mal escritas- y las buenas prácticas: nombres descriptivos, organización de lo general a lo específico y comentarios que marquen secciones.

Puntos clave para recordar: una regla CSS es selector + declaraciones; cada declaración es propiedad: valor; el archivo externo se enlaza con `<link rel="stylesheet" href="...">`; las clases se reutilizan y se combinan; el id es único; y `!important` casi nunca es la respuesta.

UNIDAD 2: COLOR, TEXTO Y FONDOS

Introducción a la unidad

Si la Unidad 1 fue aprender a tomar el pincel y apuntar a la pared correcta, esta unidad es aprender a mezclar los colores y a escribir bonito sobre la pared. El color y la tipografía son las dos decisiones visuales que más impacto tienen en cómo se percibe un sitio web. Puedes tener el layout más moderno del mundo, pero si los colores chocan y la letra es ilegible, el visitante se va en tres segundos. Al revés también funciona: una página sencilla con una paleta de colores armoniosa y una tipografía bien elegida se siente profesional de inmediato.

En esta unidad vas a dominar las tres formas de expresar colores en CSS (nombres, hexadecimal y RGB), las propiedades que controlan el texto (familia de fuente, tamaño, grosor, alineación, interlineado y más) y las propiedades de fondo (colores, imágenes, repetición y posición). Al terminar serás capaz de vestir cualquier página con una paleta coherente y una tipografía legible y atractiva.

Explicación detallada

Los colores en CSS: tres formas de decir lo mismo

En CSS puedes expresar un color de tres formas, y las tres significan exactamente lo mismo para el navegador. La diferencia está en la precisión y la comodidad.

La primera forma es el nombre del color, en inglés: red, blue, green, black, white, orange, purple, gray, etc. CSS reconoce 140 nombres de colores. Son fáciles de recordar y perfectos para prototipos rápidos:

```
h1 {  
  color: tomato;  
  background-color: lightyellow;  
}
```

La limitación es obvia: solo hay 140, y no puedes pedir "un azul un poquito más oscuro que el de ayer pero no tanto". Para eso están los otros dos formatos.

La segunda forma es el hexadecimal, que es el formato más usado en el trabajo profesional. Se escribe con un numeral seguido de 6 dígitos que representan la cantidad de rojo, verde y azul, en ese orden, en base 16:

```
color: #FF0000; /* rojo puro */
color: #00FF00; /* verde puro */
color: #0000FF; /* azul puro */
color: #FFFFFF; /* blanco */
color: #000000; /* negro */
```

Cada par de dígitos va de 00 (nada de ese color) a FF (todo ese color). Así, #FF0000 es todo el rojo y nada de verde ni azul: rojo puro. #808080 es mitad de cada uno: gris medio. Una vez que entiendes la lógica, puedes "leer" un hexadecimal y hacerte una idea del color sin necesidad de verlo.

Hay una forma abreviada: si los tres pares tienen dígitos repetidos, puedes escribir solo 3 dígitos. #FF0000 se puede escribir #F00, #FFFFFF como #FFF y #000000 como #000. El navegador los interpreta igual.

La tercera forma es RGB, que expresa lo mismo pero en decimal, con números del 0 al 255:

```
color: rgb(255, 0, 0); /* rojo puro, igual que #FF0000 */
color: rgb(0, 0, 0); /* negro */
color: rgb(128, 128, 128); /* gris medio */
```

Y existe su variante RGBA, que agrega un cuarto valor: la opacidad (alfa), de 0 (totalmente transparente) a 1 (totalmente opaco):

```
background-color: rgba(0, 0, 0, 0.5); /* negro al 50% de opacidad */
```

RGBA es muy útil para superponer colores sobre imágenes o crear efectos de capas sin necesidad de imágenes adicionales. Por ejemplo, un recuadro de texto semitransparente sobre una foto de fondo.

Hay también el formato HSL (tono, saturación, luminosidad), que a muchos diseñadores les resulta más intuitivo porque piensas en "qué tan vivo" y "qué tan claro" en vez de mezclar rojos, verdes y azules:

```
color: hsl(0, 100%, 50%);    /* rojo puro */  
color: hsl(120, 100%, 50%); /* verde puro */  
color: hsl(240, 100%, 50%); /* azul puro */
```

El tono va de 0 a 360 grados en una rueda de color (0 es rojo, 120 es verde, 240 es azul), la saturación de 0% (gris) a 100% (color puro) y la luminosidad de 0% (negro) a 100% (blanco). No es obligatorio dominarlo ahora, pero conviene saber que existe porque lo verás en código profesional.

Analogía de los colores: la paleta del pintor

Piensa en los tres formatos como tres formas de pedir pintura en la tienda. El nombre es como decir "dame un bote de rojo": rápido, pero no controlas el tono exacto. El hexadecimal es como dar el código exacto de la carta de colores: "el 4F8C2B", precisión total. El RGB es como decirle al dependiente la receta: "échale 255 de rojo, 128 de verde y 0 de azul". Los tres te dan pintura; eliges según cuánta precisión necesites.

Dónde se aplican los colores: color vs background-color

Las dos propiedades de color más usadas son `color` y `background-color`, y los principiantes las confunden al inicio. Es simple: `color` define el color del texto (y de elementos de texto como bordes de decoración heredada) dentro del elemento, mientras que `background-color` pinta el fondo del elemento:

```
p {  
  color: white;           /* el texto es blanco */  
  background-color: navy; /* el fondo del párrafo es azul marino */  
}
```

Un error clásico es poner `color: white` sin fondo oscuro detrás: el texto blanco sobre fondo blanco es invisible. Siempre que cambies el color del texto, pregúntate sobre qué fondo va a estar.

Tipografía: la voz visual de tu sitio

La tipografía es el arte de que el texto se vea bien, y en CSS se controla con un grupo de propiedades que trabajan juntas. Vamos una por una.

`font-family` define qué letra se usa. El truco es que no puedes garantizar que el visitante tenga instalada la fuente que quieres, así que se escribe una lista de opciones separadas por comas, de la más deseada a la más genérica:

```
body {  
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;  
}
```

El navegador prueba la primera; si no existe en el sistema, pasa a la segunda, y así hasta llegar a `sans-serif`, que es una familia genérica que siempre existe. Los nombres de fuente con espacios van entre comillas. Las familias genéricas son cinco: `serif` (con patitas, como Times New Roman, formal y tradicional), `sans-serif` (sin patitas, como Arial, moderna y limpia), `monospace` (ancho fijo, como Courier, para código), `cursive` (manuscrita) y `fantasy` (decorativa).

Regla práctica: usa `sans-serif` para el cuerpo del texto en pantalla (se lee mejor), `serif` para títulos elegantes o contenido editorial, y `monospace` para código. No mezcles más de dos o tres familias en un mismo sitio: la consistencia tipográfica es lo que hace que un diseño se vea profesional.

`font-size` define el tamaño del texto. Puedes usar píxeles (16px), pero en diseño moderno se prefieren unidades relativas como `em` o `rem` (las veremos a fondo en la Unidad 9). El tamaño base recomendado para el cuerpo de texto es 16px, que es además el valor por defecto de los navegadores.

`font-weight` controla el grosor: `normal`, `bold`, o valores numéricos del 100 al 900 (400 es normal, 700 es negrita). No todas las fuentes tienen todos los grosores; si pides un grosor que no existe, el navegador elige el más cercano.

`font-style` suele usarse para cursivas: `italic` pone el texto en itálica, `normal` lo quita (útil para quitarle la itálica por defecto a elementos como `<em>` o `<address>` si no la quieres).

Además de la fuente en sí, hay propiedades que controlan cómo se presenta el texto en la página. `text-align` alinea el texto: `left` (izquierda, lo normal), `right` (derecha), `center` (centrado) y `justify` (justificado, que estira las líneas para que ambos bordes queden parejos, como en los periódicos). Ojo: el justificado en pantallas angostas puede crear huecos feos entre palabras; úsalo con moderación.

line-height (interlineado) es una de las propiedades más subestimadas y más poderosas: define la altura de cada línea de texto. Un valor de 1.5 a 1.8 hace que los párrafos largos sean mucho más legibles que el valor por defecto:

```
p {  
  font-size: 16px;  
  line-height: 1.7;  
}
```

Se puede escribir sin unidades (1.7 significa "1.7 veces el tamaño de la fuente") y esa es la forma recomendada.

letter-spacing ajusta el espacio entre letras (útil en títulos en mayúsculas para darles aire) y word-spacing el espacio entre palabras. text-transform cambia la capitalización sin tocar el HTML: uppercase (TODO MAYÚSCULAS), lowercase (todo minúsculas), capitalize (Primera Letra De Cada Palabra). Es muy útil para títulos: escribes el texto normal en el HTML y dejas que el CSS lo ponga en mayúsculas.

text-decoration controla decoraciones de línea: underline (subrayado), line-through (tachado, para precios con descuento) y none (sin decoración, lo más usado para quitarle el subrayado a los enlaces). Y text-shadow agrega sombra al texto (la veremos con las sombras en la Unidad 8).

Fondos: más allá del color plano

background-color pinta el fondo de un elemento con un color sólido, y ya lo usaste en la Unidad 1. Pero los fondos en CSS pueden ser mucho más: imágenes, degradados, patrones que se repiten y composiciones con posición y tamaño controlados.

background-image coloca una imagen como fondo:

```
.hero {  
  background-image: url("fondo.jpg");  
}
```

La función url() recibe la ruta de la imagen, entre comillas (las comillas son opcionales pero recomendadas). La ruta puede ser relativa ("imagenes/fondo.jpg"), absoluta o una URL completa de internet. Si la imagen no se encuentra, simplemente no se muestra nada: no hay error visible, solo un fondo vacío, así que revisa bien la ruta.

Por defecto, una imagen de fondo se repite en mosaico para cubrir todo el elemento. Ese comportamiento se controla con `background-repeat: repeat` (repite en ambas direcciones, el valor por defecto), `repeat-x` (solo horizontal), `repeat-y` (solo vertical) y `no-repeat` (la imagen aparece una sola vez).

`background-position` decide dónde se coloca la imagen cuando no se repite: puedes usar palabras (`top`, `bottom`, `left`, `right`, `center` y combinaciones como `center center` o `top right`) o valores en píxeles o porcentajes (`background-position: 20px 50px;`).

`background-size` controla el tamaño de la imagen de fondo, y tiene dos valores mágicos: `cover` (la imagen crece hasta cubrir todo el elemento, aunque se recorte un poco; ideal para héroes y banners) y `contain` (la imagen se ajusta para verse completa dentro del elemento, aunque queden espacios vacíos). También puedes dar medidas exactas: `background-size: 300px 200px;`.

La forma abreviada `background` permite escribir varias de estas propiedades en una sola línea:

```
.hero {  
  background: #1a1a2e url("fondo.jpg") no-repeat center center / cover;  
}
```

El orden en la abreviatura es flexible pero hay convenciones; lo importante es que el color de fondo va primero (sirve como respaldo mientras la imagen carga o si no se encuentra) y el tamaño va después de la posición separado por una diagonal. Si la abreviatura te confunde, no hay problema en escribir cada propiedad por separado: es más largo pero más claro, y funciona igual.

Además de imágenes, CSS puede generar degradados directamente, sin archivos: `linear-gradient` crea una transición suave entre dos o más colores:

```
.degradado {  
  background: linear-gradient(to right, #ff7e5f, #feb47b);  
}
```

Esto pinta un degradado horizontal del naranja-rojizo al durazno. Puedes cambiar la dirección (`to bottom`, `to top left`, o un ángulo como `45deg`) y agregar más colores con paradas: `linear-gradient(45deg, red 0%, yellow 50%, green 100%)`. También existe `radial-gradient`, que irradia desde un centro hacia afuera, perfecto para efectos de luz o viñetas.

Errores comunes de esta unidad

Error 1: texto invisible por falta de contraste. El error más frecuente es poner texto claro sobre fondo claro o texto oscuro sobre fondo oscuro. Regla: el contraste entre texto y fondo debe ser fuerte. Blanco sobre negro, negro sobre blanco, blanco sobre azul marino: todo bien. Gris claro sobre blanco: mal.

Error 2: usar demasiadas fuentes y colores. Un sitio con 5 familias tipográficas y 8 colores se ve caótico. Límitate a 2 familias (una para títulos, una para cuerpo) y a una paleta de 3 a 4 colores: un color principal, un color secundario, un color de acento para botones y llamadas a la acción, y neutros (blanco, negro, grises) para fondos y texto.

Error 3: rutas de imágenes de fondo mal escritas. Recuerda que la ruta en `url()` se resuelve relativa al archivo CSS, no al HTML. Si tu CSS está en `css/estilos.css` y la imagen en `imagenes/fondo.jpg`, la ruta correcta desde el CSS es `url("../imagenes/fondo.jpg")`: el `../` significa "sube una carpeta". Este detalle causa más dolores de cabeza de los que imaginas.

Error 4: olvidar el color de respaldo detrás de una imagen de fondo. Si la imagen tarda en cargar o no se encuentra, el visitante verá el color de fondo. Pon siempre un `background-color` parecido al tono general de la imagen para que la transición sea suave.

Error 5: abusar de `text-transform: uppercase` en textos largos. Las mayúsculas en títulos cortos se ven bien; en párrafos completos son difíciles de leer y en internet se interpretan como si estuvieras gritando.

Buenas prácticas

Define tu paleta al inicio del proyecto y úsala de forma consistente. Una técnica profesional es declarar los colores como variables CSS (las veremos en la Unidad 9) para poder cambiar toda la paleta desde un solo lugar. Para la tipografía, establece una escala: por ejemplo, cuerpo 16px, h3 20px, h2 28px, h1 40px, con interlineados de 1.6 a 1.8 en párrafos. Y para fondos con imagen y texto encima, oscurece ligeramente la imagen (con un degradado semitransparente encima o con `background-blend-mode`) para garantizar que el texto siempre sea legible.

Ejemplos paso a paso

Ejemplo 1: Paleta de colores en los tres formatos

Crea una página que muestre la misma tarjeta en tres versiones, cada una usando un formato de color distinto. Verás que el resultado es idéntico.

```
<div class="tarjeta nombre">Color por nombre</div>
<div class="tarjeta hex">Color hexadecimal</div>
<div class="tarjeta rgb">Color RGB</div>
```

```
.tarjeta {
  width: 250px;
  padding: 20px;
  margin: 10px;
  color: white;
  font-family: Arial, sans-serif;
}

.nombre { background-color: steelblue; }
.hex { background-color: #4682b4; } /* steelblue en hexadecimal */
.rgb { background-color: rgb(70, 130, 180); } /* steelblue en RGB */
```

Los tres divs se ven exactamente igual porque `steelblue`, `#4682b4` y `rgb(70, 130, 180)` son el mismo color. Este ejercicio te entrena el ojo para reconocer equivalencias.

Ejemplo 2: Sistema tipográfico completo

Vamos a construir la tipografía base de un sitio ficticio, con escala y jerarquía claras:

```

body {
  font-family: "Segoe UI", Verdana, sans-serif;
  font-size: 16px;
  line-height: 1.7;
  color: #333333;
  background-color: #ffffff;
}

h1 {
  font-family: Georgia, "Times New Roman", serif;
  font-size: 40px;
  font-weight: 700;
  letter-spacing: 1px;
  color: #1a1a2e;
}

h2 {
  font-family: Georgia, serif;
  font-size: 28px;
  color: #16213e;
}

.subtitulo {
  font-size: 18px;
  color: #666666;
  font-style: italic;
}

.etiqueta {
  font-size: 12px;
  text-transform: uppercase;
  letter-spacing: 2px;
  color: #e94560;
  font-weight: bold;
}

```

Observa las decisiones: el cuerpo usa sans-serif para legibilidad en pantalla, los títulos usan serif para darles carácter editorial, el interlineado generoso (1.7) hace cómoda la lectura, y las etiquetas pequeñas en mayúsculas con espaciado entre letras se ven profesionales. Esta es exactamente la forma en que se diseña la tipografía de un sitio real.

Ejemplo 3: Hero con imagen de fondo

El "hero" es la sección grande de bienvenida que ves al entrar a un sitio. Vamos a construir uno:

```
<section class="hero">
  <h1>Aprende CSS desde cero</h1>
  <p>El curso gratis que te convierte en diseñador web.</p>
  <a href="#" class="boton">Empezar ahora</a>
</section>
```

```
.hero {
  background-color: #1a1a2e;
  background-image: url("hero-fondo.jpg");
  background-repeat: no-repeat;
  background-position: center center;
  background-size: cover;
  color: white;
  text-align: center;
  padding: 120px 20px;
}

.hero h1 {
  font-size: 48px;
  text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.6);
}

.hero p {
  font-size: 20px;
  margin-bottom: 30px;
}

.boton {
  background-color: #e94560;
  color: white;
  padding: 15px 40px;
  text-decoration: none;
  border-radius: 30px;
  font-weight: bold;
}
```

Detalles importantes: el background-color oscuro es el respaldo por si la imagen no carga. El text-shadow con negro semitransparente garantiza que el título se lea sobre cualquier

imagen. El botón es un enlace con clase que parece botón gracias al padding, el fondo de color y el borde redondeado.

Ejemplo 4: Degradados sin imágenes

```
.tarjeta-degradado {
  width: 300px;
  height: 180px;
  border-radius: 15px;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  color: white;
  padding: 25px;
  font-family: Arial, sans-serif;
}

.boton-degradado {
  background: linear-gradient(to right, #ff7e5f, #feb47b);
  border: none;
  color: white;
  padding: 12px 30px;
  border-radius: 25px;
  font-size: 16px;
  font-weight: bold;
}

.circulo-luz {
  width: 200px;
  height: 200px;
  border-radius: 50%;
  background: radial-gradient(circle, #fff8d6 0%, #ffd93d 70%, #ff9a3d 100%);
}
```

Tres usos típicos del degradado: tarjetas con fondo atractivo, botones con vida y efectos de luz radial. Todo sin una sola imagen, lo que significa carga instantánea.

Ejemplo 5: Tarjeta de cita con tipografía cuidada

```
<blockquote class="cita">
  "El diseño no es solo cómo se ve, es cómo funciona."
  <cite>- Principio de diseño web</cite>
</blockquote>
```

```

.cita {
  font-family: Georgia, serif;
  font-size: 24px;
  font-style: italic;
  line-height: 1.6;
  color: #2c3e50;
  border-left: 5px solid #e94560;
  padding: 20px 30px;
  background-color: #f9f9f9;
  max-width: 600px;
}

.cita cite {
  display: block;
  font-size: 14px;
  font-style: normal;
  text-transform: uppercase;
  letter-spacing: 1px;
  color: #888;
  margin-top: 15px;
}

```

Este ejemplo combina todo: tipografía serif itálica para la cita, borde lateral de acento, fondo sutil y autor en mayúsculas pequeñas. Es el tipo de componente que verás en blogs profesionales.

Práctica

Ejercicio 1

Escribe CSS que pinte el mismo tono de rojo puro usando los tres formatos (nombre, hexadecimal y RGB) en tres párrafos distintos con clases "r1", "r2" y "r3". El texto debe ser blanco en los tres.

SOLUCIÓN:

```

.r1 { background-color: red; color: white; }
.r2 { background-color: #FF0000; color: white; }
.r3 { background-color: rgb(255, 0, 0); color: white; }

```

Los tres son rojo puro. El texto blanco sobre rojo tiene buen contraste y se lee bien.

Ejercicio 2

Crea la tipografía base de un blog: cuerpo con fuente sans-serif, 17px, interlineado 1.8, color #2d2d2d; títulos h1 de 38px en serif color #111; subtítulos con clase "bajada" en itálica gris.

SOLUCIÓN:

```
body {  
  font-family: Arial, Helvetica, sans-serif;  
  font-size: 17px;  
  line-height: 1.8;  
  color: #2d2d2d;  
}  
  
h1 {  
  font-family: Georgia, "Times New Roman", serif;  
  font-size: 38px;  
  color: #111111;  
}  
  
.bajada {  
  font-style: italic;  
  color: #777777;  
  font-size: 19px;  
}
```

El interlineado 1.8 es generoso y hace muy cómoda la lectura de artículos largos, que es justo lo que un blog necesita.

Ejercicio 3

Haz un banner con degradado diagonal de #1e3c72 a #2a5298, texto blanco centrado, padding de 60px y esquinas redondeadas de 12px. Dentro, un título y un párrafo.

SOLUCIÓN:

```
.banner {
  background: linear-gradient(135deg, #1e3c72, #2a5298);
  color: white;
  text-align: center;
  padding: 60px 20px;
  border-radius: 12px;
  font-family: Arial, sans-serif;
}

.banner h2 {
  font-size: 32px;
  margin-bottom: 10px;
}
```

El degradado diagonal (135deg) se ve más dinámico que uno vertical. El blanco sobre esos azules oscuros tiene excelente contraste.

Ejercicio 4

Explica qué está mal en este código y corrígelo:

```
.aviso {
  color: #ffff;
  background-color: yellow;
  font-family: Arial
  text-align: center
}
```

SOLUCIÓN: Hay cuatro problemas. Primero, #ffff no es un hexadecimal válido: debe tener 3 o 6 dígitos (#fff o #ffffff). Segundo, texto amarillo claro (#fff es blanco) sobre fondo amarillo es ilegible: falta contraste. Tercero y cuarto, a font-family y text-align les faltan los puntos y coma. Versión corregida:

```
.aviso {
  color: #333333;
  background-color: #fff3cd;
  font-family: Arial, sans-serif;
  text-align: center;
}
```

Ahora el texto gris oscuro sobre amarillo pálido se lee perfectamente.

Ejercicio 5

Crea una clase "hero-oscuro" para una sección de bienvenida: imagen de fondo "portada.jpg" que no se repita, centrada y cubriendo todo el elemento; color de respaldo #222; texto blanco centrado con padding de 100px; y el h1 con sombra de texto negra semitransparente.

SOLUCIÓN:

```
.hero-oscuro {  
  background-color: #222222;  
  background-image: url("portada.jpg");  
  background-repeat: no-repeat;  
  background-position: center center;  
  background-size: cover;  
  color: white;  
  text-align: center;  
  padding: 100px 20px;  
}  
  
.hero-oscuro h1 {  
  text-shadow: 2px 2px 5px rgba(0, 0, 0, 0.7);  
}
```

La sombra con 0.7 de opacidad es lo suficientemente fuerte para garantizar legibilidad sobre fotos claras y oscuras.

Ejercicio 6

Diseña tres etiquetas de precio usando text-transform y letter-spacing: clase "precio" con el número en 36px negritas color #e94560, clase "antes" con el precio anterior tachado en gris, y clase "etiqueta-oferta" en mayúsculas con 3px de espaciado entre letras.

SOLUCIÓN:

```

.precio {
  font-size: 36px;
  font-weight: bold;
  color: #e94560;
}

.antes {
  color: #999999;
  text-decoration: line-through;
  font-size: 20px;
}

.etiqueta-oferta {
  text-transform: uppercase;
  letter-spacing: 3px;
  font-size: 12px;
  color: #e94560;
  font-weight: bold;
}

```

```

<span class="etiqueta-oferta">Oferta limitada</span>
<span class="antes">$599</span>
<span class="precio">$299</span>

```

El tachado (line-through) en el precio anterior es el patrón visual universal de los descuentos en tiendas en línea.

Resumen de la unidad

En esta unidad aprendiste las tres formas de expresar colores en CSS: nombres (140 colores, rápidos para prototipos), hexadecimal (el estándar profesional, 6 dígitos RGB en base 16, con forma abreviada de 3) y RGB/RGBA (decimal de 0 a 255, con canal de opacidad). Viste la diferencia entre color (texto) y background-color (fondo) y la importancia del contraste para la legibilidad. Dominaste la tipografía: font-family con lista de respaldo, font-size, font-weight, font-style, text-align, line-height (usa 1.5 a 1.8 sin unidades), letter-spacing, text-transform y text-decoration. Y exploraste los fondos: background-image con url(), background-repeat, background-position, background-size (cover y contain), la abreviatura background y los degradados con linear-gradient y radial-gradient.

Puntos clave: limita tu paleta a 3-4 colores y tus fuentes a 2 familias; pon siempre un color de respaldo detrás de las imágenes de fondo; recuerda que las rutas en `url()` son relativas al archivo CSS; y cuida el contraste como si tu sitio dependiera de ello, porque depende.

UNIDAD 3: BORDES, ESPACIADO Y EL MODELO DE CAJA

Introducción a la unidad

Hay una frase que todo diseñador web termina tatuándose en la memoria: "todo en CSS es una caja". Cada título, cada párrafo, cada imagen, cada botón que ves en una página es, para el navegador, una caja rectangular invisible con capas. Si entiendes cómo funcionan esas capas -qué es el contenido, qué es el padding, qué es el borde y qué es el margin- entonces entiendes el 80 por ciento de los problemas de layout que vas a enfrentar en tu vida: por qué un elemento se ve más grande de lo que esperabas, por qué dos cajas no se separan como quieres, por qué algo se desborda de su contenedor.

En esta unidad vas a dominar los bordes (grosor, estilo, color y esquinas redondeadas), las dos propiedades de espaciado más importantes de CSS (margin hacia afuera, padding hacia adentro) y el modelo de caja completo, incluyendo el truco de `box-sizing` que usan todos los profesionales para que las matemáticas cuadren. Al terminar, los tamaños en CSS dejarán de ser un misterio y se convertirán en una herramienta precisa.

Explicación detallada

Bordes: el marco de cada caja

La propiedad `border` dibuja una línea alrededor del elemento. Es una abreviatura que combina tres valores en este orden: grosor, estilo y color.

```
.caja {  
  border: 2px solid black;  
}
```

Esto dibuja un borde de 2 píxeles, de línea continua (`solid`), de color negro. Los estilos disponibles son: `solid` (línea continua), `dashed` (guiones), `dotted` (puntos), `double` (doble línea), `groove`, `ridge`, `inset`, `outset` (efectos 3D que hoy casi no se usan) y `none` (sin borde). En la práctica moderna verás casi siempre `solid`, `dashed` y `dotted`.

También puedes controlar cada aspecto por separado con `border-width`, `border-style` y `border-color`, y cada lado por separado con `border-top`, `border-right`, `border-bottom` y `border-left`:

```
.tarjeta {  
  border-top: 4px solid #e94560;  
  border-bottom: 1px dashed #cccccc;  
}
```

Esto es muy útil para detalles de diseño: una línea de acento arriba de una tarjeta, un separador punteado abajo de un elemento de lista.

La propiedad estrella de los bordes modernos es `border-radius`, que redondea las esquinas:

```
.boton {  
  border-radius: 10px;  
}  
  
.avatar {  
  border-radius: 50%;  
}
```

Con un valor en píxeles redondeas las esquinas; con `50%` conviertes un cuadrado en un círculo perfecto (muy usado para fotos de perfil). Puedes redondear cada esquina por separado: `border-radius: 10px 20px 30px 40px`; aplica a las esquinas superior-izquierda, superior-derecha, inferior-derecha e inferior-izquierda, en ese orden, como las manecillas del reloj. También existen las propiedades individuales: `border-top-left-radius`, etc.

Un detalle que confunde a los principiantes: el borde ocupa espacio y se dibuja entre el padding y el margin (lo veremos en el modelo de caja). Un borde de `5px` hace que la caja se vea `10px` más ancha en total (`5` de cada lado) a menos que uses `box-sizing: border-box`.

Margin y padding: el espacio hacia afuera y hacia adentro

Estas dos propiedades controlan el espacio, pero en direcciones opuestas, y confundirlas es el error más común de quien empieza.

`padding` es el espacio entre el contenido del elemento y su borde. Es espacio hacia adentro, dentro de la caja. Si tienes un botón con texto y le pones padding, el botón crece porque el texto "empuja" el borde hacia afuera:

```
.boton {  
  padding: 12px 24px;  
  background-color: #e94560;  
  color: white;  
}
```

margin es el espacio entre el borde del elemento y los elementos vecinos. Es espacio hacia afuera, fuera de la caja. Si tienes dos tarjetas y les pones margin, se separan entre sí:

```
.tarjeta {  
  margin: 20px;  
  background-color: white;  
}
```

Analogía: imagina una foto enmarcada colgada en la pared. El padding es el passepartout, ese cartón entre la foto y el marco: espacio dentro del cuadro. El borde es el marco de madera. El margin es la distancia entre ese cuadro y los otros cuadros de la pared: espacio fuera del cuadro. Si quieres que la foto "respire" dentro de su marco, aumentas el padding. Si quieres separar los cuadros entre sí, aumentas el margin.

Ambas propiedades aceptan de 1 a 4 valores, y esta es una sintaxis que debes memorizar porque la usarás todos los días:

```
/* Un valor: aplica a los 4 lados */  
margin: 20px;  
  
/* Dos valores: vertical | horizontal */  
margin: 10px 20px; /* 10px arriba y abajo, 20px izquierda y derecha */  
  
/* Tres valores: arriba | horizontal | abajo */  
margin: 10px 20px 30px;  
  
/* Cuatro valores: arriba | derecha | abajo | izquierda (sentido del reloj) */  
margin: 10px 20px 30px 40px;
```

El truco para recordar el orden de los cuatro valores es "las manecillas del reloj": empiezas arriba y giras. También existen las propiedades individuales: margin-top, margin-right, margin-bottom, margin-left, y lo mismo para padding.

Hay dos comportamientos especiales del margin que debes conocer. El primero es el margin automático para centrar: si un elemento de bloque tiene un ancho definido, `margin: 0 auto;` lo centra horizontalmente en su contenedor. Es la técnica clásica para centrar contenedores:

```
.contenedor {  
  width: 800px;  
  margin: 0 auto; /* 0 arriba/abajo, automático a los lados = centrado */  
}
```

El navegador reparte el espacio sobrante en partes iguales a izquierda y derecha, y el elemento queda centrado. Esto solo funciona con elementos de bloque que tengan un ancho menor al de su contenedor.

El segundo comportamiento especial es el colapso de márgenes verticales: cuando dos elementos de bloque están uno sobre otro, sus márgenes verticales no se suman, se traslapan y gana el mayor. Si un párrafo tiene `margin-bottom: 30px` y el siguiente tiene `margin-top: 20px`, la separación real entre ellos será de 30px, no de 50px. Esto sorprende al principio, pero es el comportamiento estándar y con la práctica lo anticipas sin pensar.

Otra diferencia clave: el padding nunca puede ser negativo y el fondo del elemento se extiende por el área del padding (si la caja tiene fondo de color, el padding también se pinta). El margin sí puede ser negativo (para traslapar elementos, una técnica avanzada) y siempre es transparente: nunca se pinta.

El modelo de caja: la anatomía de todo elemento

Llegamos al concepto central. Cada elemento en CSS es una caja formada por cuatro capas concéntricas, de adentro hacia afuera:

1. Content (contenido): el texto, la imagen o lo que sea que contiene el elemento. Su tamaño lo definen `width` y `height`.
2. Padding: el espacio entre el contenido y el borde. Se pinta con el fondo del elemento.
3. Border: la línea del borde. Su grosor suma al tamaño total.
4. Margin: el espacio transparente entre el borde y los vecinos. No se pinta nunca.

El tamaño total que ocupa un elemento en la página es: `width` + padding izquierdo y derecho + border izquierdo y derecho + margin izquierdo y derecho. Y aquí viene la trampa histórica de CSS: por defecto, `width` define solo el ancho del contenido, NO el ancho total de la caja.

Mira este ejemplo que ha confundido a generaciones de programadores:

```
.caja {  
  width: 300px;  
  padding: 20px;  
  border: 5px solid black;  
  margin: 10px;  
}
```

¿Cuánto mide de ancho en total esta caja? Hagamos la cuenta: contenido 300px + padding 20px de cada lado (40px) + borde 5px de cada lado (10px) = 350px de caja visible, más 10px de margin de cada lado = 370px de espacio total ocupado. El `width: 300px` que escribiste no es el ancho que ves en pantalla. Esta es la causa del 90 por ciento de los "¿por qué mi layout se rompió?".

La solución que usa toda la industria moderna es `box-sizing: border-box`:

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

Con `border-box`, el `width` que declares incluye el contenido, el padding y el borde. La caja anterior con `width: 300px` medirá exactamente 300px de ancho visible, y el navegador reducirá el área de contenido automáticamente para que cuadre. Las matemáticas se vuelven intuitivas: lo que escribes es lo que ves.

El selector `*` (asterisco) selecciona todos los elementos, así que esa regla aplica `border-box` a todo el documento. Es tan estándar que muchos frameworks y plantillas la incluyen por defecto. Mi recomendación: ponla siempre al inicio de tus hojas de estilo y olvídate del problema para siempre.

El valor por defecto es `content-box` (el comportamiento clásico que acabamos de ver). Existe por razones históricas y de compatibilidad, pero en proyectos nuevos no hay razón para usarlo.

Visualizando el modelo de caja

La mejor forma de ver el modelo de caja es con las herramientas de desarrollo del navegador. Presiona F12, ve a la pestaña de elementos, selecciona cualquier elemento y busca el diagrama del modelo de caja (en Chrome está en la pestaña de estilos, abajo del todo). Verás un diagrama con las cuatro capas y sus medidas exactas en píxeles, cada capa en un color distinto. Cuando algo no cuadra en tu layout, ese diagrama te dice exactamente qué capa está causando el problema.

Haz este experimento ahora mismo: crea una caja con `width`, `padding`, `border` y `margin`, ábrela en el navegador e inspecciona el diagrama. Luego agrega `box-sizing: border-box` y observa cómo cambian las medidas. Cinco minutos con esa herramienta valen más que una hora de teoría.

Errores comunes de esta unidad

Error 1: no usar `box-sizing: border-box` y pelear con anchos que no cuadran. Solución: agrégalo con el selector universal al inicio de cada proyecto.

Error 2: confundir `margin` con `padding`. Pregúntate: "¿quiero espacio dentro de la caja (entre el contenido y el borde) o fuera de la caja (entre esta caja y sus vecinas)?" `Dentro = padding`, `fuera = margin`.

Error 3: esperar que los márgenes verticales se sumen. Recuerda el colapso: entre dos bloques apilados, los márgenes verticales se traslapan y gana el mayor.

Error 4: usar `margin` para separar el contenido del borde de su propia caja. Eso es trabajo del `padding`. El `margin` separa cajas entre sí.

Error 5: olvidar que el borde suma al tamaño total (sin `border-box`). Un `width: 100%` más un borde de 1px se desborda de su contenedor por 2px. Con `border-box`, ese problema desaparece.

Buenas prácticas

Primera: `box-sizing: border-box` global, siempre. Segunda: usa `padding` para el "aire" interno de botones, tarjetas y contenedores; usa `margin` para separar componentes entre sí. Tercera: prefiere la sintaxis abreviada de 1 a 4 valores cuando los lados comparten lógica, y las propiedades individuales cuando necesitas ajustar un solo lado. Cuarta: para centrar un bloque con ancho definido, `margin: 0 auto` es tu mejor amigo. Quinta: nombra tus espaciados de forma consistente en el proyecto (por ejemplo, múltiplos de 8px: 8, 16, 24, 32) para que el diseño se sienta ordenado y rítmico.

Ejemplos paso a paso

Ejemplo 1: Bordes de todos los sabores

```
<div class="marco solido">Borde sólido</div>
<div class="marco punteado">Borde punteado</div>
<div class="marco rayado">Borde rayado</div>
<div class="marco grueso">Borde grueso con radio</div>
<div class="marco circulo"></div>
```

```
.marco {
  width: 250px;
  padding: 15px;
  margin: 10px;
  font-family: Arial, sans-serif;
  text-align: center;
}

.solido { border: 2px solid #333; }
.punteado { border: 3px dotted #e94560; }
.rayado { border: 3px dashed #2a9d8f; }
.grueso {
  border: 6px solid #1a1a2e;
  border-radius: 15px;
}

.circulo {
  width: 100px;
  height: 100px;
  background-color: #f4a261;
  border: 4px solid #e76f51;
  border-radius: 50%;
}
```

El último ejemplo muestra el truco del círculo: un cuadrado con `border-radius: 50%`. Es la técnica estándar para avatares y fotos de perfil.

Ejemplo 2: margin vs padding, la demostración definitiva

```
<div class="contenedor">
  <div class="caja con-padding">Padding grande</div>
  <div class="caja con-margin">Margin grande</div>
</div>
```

```
.contenedor {
  background-color: #eeeeee;
  padding: 20px;
  font-family: Arial, sans-serif;
}

.caja {
  background-color: #4a90d9;
  color: white;
  padding: 10px;
}

.con-padding {
  padding: 40px; /* la caja crece hacia adentro */
}

.con-margin {
  margin: 40px; /* la caja se separa de sus vecinas */
}
```

Prueba este código y observa: la primera caja se hace más grande (el fondo azul crece porque el padding se pinta), mientras que la segunda mantiene su tamaño pero se aleja de todo lo demás (el margin es transparente, se ve el gris del contenedor). Esta es la diferencia que debes sentir en los dedos.

Ejemplo 3: El problema del box-sizing y su solución

Primero, el problema. Dos cajas que deberían caber lado a lado en un contenedor de 620px:

```
<div class="fila">
  <div class="columna">Columna 1</div>
  <div class="columna">Columna 2</div>
</div>
```

```
.fila {
  width: 620px;
  background-color: #ddd;
  overflow: hidden;
}

.columna {
  width: 300px;
  padding: 20px;
  border: 5px solid #333;
  float: left;
  background-color: white;
}
```

Sin `box-sizing`, cada columna mide $300 + 40 + 10 = 350\text{px}$, total 700px , y no caben en los 620px : la segunda se cae abajo. Ahora agrega al inicio del CSS:

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

Magia: cada columna mide exactamente 300px (el navegador encoge el contenido para que cuadre) y las dos caben perfecto lado a lado. Este es el antes y después que todo desarrollador debe ver una vez en la vida.

Ejemplo 4: Tarjeta completa con las cuatro capas

Vamos a construir una tarjeta usando conscientemente cada capa del modelo de caja:

```
<article class="tarjeta">
  
  <h3>Audífonos Pro</h3>
  <p>Sonido de alta fidelidad con cancelación de ruido.</p>
  <span class="precio">$1,299</span>
</article>
```

```

.tarjeta {
  width: 280px;
  background-color: white;
  border: 1px solid #e0e0e0;      /* capa 3: borde sutil */
  border-radius: 12px;           /* esquinas redondeadas */
  padding: 20px;                 /* capa 2: aire interno */
  margin: 15px;                  /* capa 4: separación de vecinas */
  font-family: Arial, sans-serif;
  box-sizing: border-box;
}

.tarjeta img {
  width: 100%;
  border-radius: 8px;
}

.tarjeta h3 {
  margin: 15px 0 5px 0;          /* aire arriba y abajo del título */
  color: #1a1a2e;
}

.tarjeta p {
  color: #666;
  font-size: 14px;
  margin: 0 0 15px 0;
}

.precio {
  font-size: 24px;
  font-weight: bold;
  color: #e94560;
}

```

Cada decisión de espaciado está etiquetada con su capa. Nota cómo el `margin` del `h3` y del párrafo controla el ritmo vertical dentro de la tarjeta, mientras el `padding` de la tarjeta da el aire general. Esta es la forma profesional de pensar el espaciado.

Ejemplo 5: Centrado clásico con `margin auto`

```
<div class="pagina">
  <header class="encabezado">
    <h1>Mi sitio</h1>
  </header>
  <main class="contenido">
    <p>Contenido centrado de la página...</p>
  </main>
</div>
```

```
.pagina {
  font-family: Arial, sans-serif;
}

.encabezado {
  background-color: #1a1a2e;
  color: white;
  padding: 20px;
  text-align: center;
}

.contenido {
  width: 800px;
  max-width: 95%;
  margin: 30px auto; /* 30px arriba/abajo, centrado a los lados */
  background-color: white;
  padding: 30px;
  border: 1px solid #ddd;
  border-radius: 8px;
}
```

El `margin: 30px auto` centra el contenido horizontalmente y le da 30px de aire arriba y abajo. El `max-width: 95%` es un seguro para pantallas pequeñas: en celular, el contenedor no se desborda. Este patrón (ancho fijo + `margin auto` + `max-width`) es la base de casi todos los layouts centrados de la web.

Práctica

Ejercicio 1

Crea un botón con clase "btn": padding de 12px vertical y 30px horizontal, fondo #e94560, texto blanco, sin borde, esquinas redondeadas de 25px, fuente Arial negritas.

SOLUCIÓN:

```
.btn {  
  padding: 12px 30px;  
  background-color: #e94560;  
  color: white;  
  border: none;  
  border-radius: 25px;  
  font-family: Arial, sans-serif;  
  font-weight: bold;  
  font-size: 16px;  
}
```

El border-radius: 25px con un padding vertical de 12px crea el efecto "píldora" tan popular en botones modernos. El border: none quita el borde por defecto que los navegadores le ponen a los <button>.

Ejercicio 2

Haz tres cajas de 150x150px con 10px de separación entre ellas, cada una con un borde distinto: la primera sólido negro de 3px, la segunda punteado rojo de 4px, la tercera con esquinas redondeadas de 20px y borde azul de 2px.

SOLUCIÓN:

```
.caja {  
  width: 150px;  
  height: 150px;  
  margin: 10px;  
  display: inline-block;  
}  
  
.b1 { border: 3px solid black; }  
.b2 { border: 4px dotted red; }  
.b3 { border: 2px solid blue; border-radius: 20px; }
```

El `display: inline-block` (que veremos a fondo en la Unidad 4) permite que las cajas se pongan lado a lado. El `margin` de 10px las separa entre sí.

Ejercicio 3

Calcula el ancho total visible (sin contar `margin`) de esta caja `SIN box-sizing`, y luego explica qué cambia `CON box-sizing: border-box`.

```
.caja {  
  width: 400px;  
  padding: 25px;  
  border: 5px solid black;  
}
```

SOLUCIÓN: Sin `border-box`: contenido 400px + padding 25px × 2 = 50px + borde 5px × 2 = 10px. Total visible: 460px. El `width` declarado (400px) es solo el contenido. Con `box-sizing: border-box`: el total visible es exactamente 400px; el navegador reduce el área de contenido a 400 - 50 - 10 = 340px para que todo cuadre. Por eso `border-box` es el estándar: lo que escribes es lo que ves.

Ejercicio 4

Corrige este CSS para que las dos columnas quepan en un contenedor de 500px:

```
.col {  
  width: 250px;  
  padding: 15px;  
  border: 2px solid gray;  
  float: left;  
}
```

SOLUCIÓN: Sin `border-box` cada columna mide 250 + 30 + 4 = 284px, total 568px: no caben. La solución es agregar `box-sizing: border-box` para que cada columna mida exactamente 250px:

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

.col {
  width: 250px;
  padding: 15px;
  border: 2px solid gray;
  float: left;
}
```

Ahora cada columna mide 250px exactos y las dos suman 500px: caben perfecto.

Ejercicio 5

Crea una "alerta" con clase "alerta": fondo #fff3cd, borde izquierdo grueso de 5px sólido #ffc107, padding de 15px, margin inferior de 20px, esquinas redondeadas de 8px solo del lado derecho.

SOLUCIÓN:

```
.alerta {
  background-color: #fff3cd;
  border-left: 5px solid #ffc107;
  padding: 15px;
  margin-bottom: 20px;
  border-top-right-radius: 8px;
  border-bottom-right-radius: 8px;
  font-family: Arial, sans-serif;
  color: #856404;
}
```

El borde lateral grueso de color es el patrón visual estándar de las alertas y notificaciones. Las esquinas redondeadas solo a la derecha (border-top-right-radius y border-bottom-right-radius) mantienen el borde izquierdo recto y limpio.

Ejercicio 6

Explica por qué estos dos párrafos tienen 30px de separación y no 50px:

```
<p style="margin-bottom: 30px;">Primero</p>
<p style="margin-top: 20px;">Segundo</p>
```

SOLUCIÓN: Por el colapso de márgenes verticales. Cuando dos elementos de bloque están apilados, sus márgenes verticales adyacentes no se suman: se traslapan y prevalece el mayor. Aquí compiten 30px contra 20px, gana 30px, y la separación real es 30px. Si necesitaras 50px reales, tendrías que usar padding en un contenedor o ajustar los valores sabiendo que colapsan.

Resumen de la unidad

En esta unidad aprendiste que todo en CSS es una caja con cuatro capas: contenido, padding, borde y margin. Dominaste los bordes con la abreviatura `border: grosor estilo color`, los estilos `solid`, `dashed` y `dotted`, los bordes por lado y `border-radius` para esquinas redondeadas (con 50% para círculos perfectos). Entendiste la diferencia crucial entre `padding` (espacio interior, se pinta con el fondo) y `margin` (espacio exterior, siempre transparente), la sintaxis de 1 a 4 valores en el orden de las manecillas del reloj, el centrado con `margin: 0 auto` y el colapso de márgenes verticales. Y lo más importante: descubriste por qué `width` no siempre es el ancho visible y cómo `box-sizing: border-box` aplicado con el selector universal resuelve el problema para siempre.

Puntos clave: usa `border-box` global en cada proyecto; padding adentro, margin afuera; inspecciona el modelo de caja con F12 cuando algo no cuadre; y nombra tus espaciados de forma consistente para diseños ordenados.

UNIDAD 4: TAMAÑOS, DISPLAY Y POSICIONAMIENTO

Introducción a la unidad

Ya sabes vestir las cajas con colores, bordes y espaciados. Ahora viene la pregunta del millón: ¿dónde se pone cada caja y cuánto espacio ocupa? Esta es la parte de CSS que históricamente más dolores de cabeza ha causado, y también la que más satisfacción da cuando la dominas. Aquí aprenderás a controlar el tamaño de los elementos con precisión (incluyendo los límites que evitan desastres en celular), a entender los modos de visualización que determinan cómo se comporta cada caja (¿ocupa toda la línea? ¿se pone al lado de su vecina? ¿desaparece?) y a posicionar elementos en lugares exactos de la página, incluyendo esos menús y botones que se quedan fijos aunque hagas scroll.

Al terminar esta unidad podrás construir layouts de dos y tres columnas con técnicas clásicas, crear barras de navegación fijas, superponer elementos con control total y evitar los desbordamientos que rompen los diseños en pantallas pequeñas.

Explicación detallada

Ancho y alto: width, height y sus límites inteligentes

Las propiedades `width` (ancho) y `height` (alto) definen el tamaño de un elemento. Hasta aquí todo simple. Lo interesante está en los límites: `max-width`, `min-width`, `max-height` y `min-height`.

`max-width` es una de las propiedades más importantes del diseño responsive, y la regla de oro es esta: las imágenes y los contenedores flexibles deben llevar `max-width: 100%`. ¿Por qué? Imagina una imagen de 1200px de ancho en un celular de 360px: sin límite, la imagen se desborda y rompe el layout, obligando al usuario a hacer scroll horizontal (la peor experiencia posible). Con `max-width: 100%`, la imagen nunca será más ancha que su contenedor: en pantallas grandes se ve a su tamaño natural y en pantallas pequeñas se encoge sola.

```
img {  
  max-width: 100%;  
  height: auto;  
}
```

El `height: auto` mantiene la proporción de la imagen al encogerse: si el ancho se reduce a la mitad, el alto también, y la imagen no se deforma. Esta combinación de dos líneas debería estar en casi todos tus proyectos.

`min-width` y `min-height` hacen lo contrario: garantizan un tamaño mínimo. Un botón con `min-width: 120px` nunca será más angosto que eso aunque su texto sea corto, lo que mantiene los botones de un formulario con tamaños consistentes.

Para contenedores de texto, el patrón profesional es:

```
.contenedor {  
  width: 100%;  
  max-width: 1100px;  
  margin: 0 auto;  
}
```

Esto crea un contenedor que en pantallas pequeñas ocupa todo el ancho disponible (con el aire que le des con padding) y en pantallas grandes no crece más allá de 1100px, quedando centrado. Es la base del 90 por ciento de los sitios web que ves todos los días.

Un detalle sobre `height`: en general, evita poner alturas fijas a elementos que contienen texto. Si el texto crece (porque el usuario agranda la letra o porque la pantalla es angosta y las líneas se parten), un alto fijo puede hacer que el texto se desborde o se corte. Deja que la altura la determine el contenido de forma natural, y usa `min-height` cuando necesites garantizar un mínimo.

Display: cómo se comporta cada caja

La propiedad `display` define el modo de visualización de un elemento, es decir, cómo se comporta en el flujo de la página. Los cuatro valores fundamentales son `block`, `inline`, `inline-block` y `none`.

block (bloque): el elemento ocupa todo el ancho disponible de su contenedor y siempre empieza en una línea nueva, empujando a sus vecinos arriba y abajo. Los títulos, párrafos, divs y secciones son bloque por defecto. Piensa en los bloques como vagones de tren: cada uno ocupa su propia vía (línea) completa.

inline (en línea): el elemento ocupa solo el espacio de su contenido y se pone al lado de sus vecinos en la misma línea, como las palabras de un párrafo. Los enlaces (<a>), el texto en negritas () y las imágenes son inline por defecto... bueno, las imágenes son un caso especial (replaced inline), pero la idea se mantiene. Limitación importante: a los elementos inline NO se les puede dar `width`, `height`, ni márgenes verticales efectivos. Si intentas ponerle `width: 200px` a un `<span>`, el navegador lo ignora.

inline-block: el punto medio perfecto. El elemento se pone al lado de sus vecinos como si fuera inline, PERO acepta `width`, `height`, `padding` y márgenes como si fuera block. Es la herramienta clásica para poner cajas lado a lado (botones en una fila, tarjetas en una galería) antes de que existieran flexbox y grid.

none: el elemento desaparece por completo de la página, como si no existiera en el HTML. No ocupa espacio, no se ve, y los demás elementos se acomodan como si no estuviera. Es diferente de `visibility: hidden`, que oculta el elemento pero conserva su espacio (queda un hueco invisible). `display: none` se usa muchísimo con JavaScript para mostrar y ocultar menús, pestañas y ventanas emergentes.

Hay más valores de `display` (`flex`, `grid`, `table`, etc.), y de hecho `flex` y `grid` son valores de `display` que convierten al elemento en un contenedor flexible o en una cuadrícula. Los veremos a fondo en las Unidades 5 y 6, pero es importante que entiendas desde ahora que pertenecen a esta misma propiedad.

Un truco muy usado: convertir elementos inline en bloque o viceversa según convenga. Por ejemplo, los enlaces de un menú vertical se suelen poner como `display: block` para que cada uno ocupe toda la línea y sea fácil de dar clic:

```
.menu-vertical a {  
  display: block;  
  padding: 12px;  
  border-bottom: 1px solid #ddd;  
}
```

Y a la inversa, una lista (es bloque por defecto) se puede horizontalizar con `display: inline-block` para hacer un menú horizontal clásico.

Posicionamiento: sacando elementos del flujo normal

Por defecto, todos los elementos siguen el "flujo normal": aparecen uno después de otro según el orden del HTML, de arriba hacia abajo (bloques) o de izquierda a derecha (inlines). La propiedad `position` te permite sacar elementos de ese flujo y colocarlos donde quieras.

Tiene cinco valores: `static`, `relative`, `absolute`, `fixed` y `sticky`.

`static` es el valor por defecto: el elemento sigue el flujo normal y las propiedades de desplazamiento (`top`, `right`, `bottom`, `left`) no le afectan. Casi nunca lo escribes explícitamente, pero conviene saber que existe.

`relative` (relativo): el elemento sigue ocupando su lugar en el flujo (los demás no se mueven), pero visualmente se desplaza desde su posición original según los valores de `top`, `right`, `bottom` y `left`:

```
.desplazado {  
  position: relative;  
  top: 20px;  
  left: 30px;  
}
```

Esto mueve la caja 20px hacia abajo y 30px hacia la derecha desde donde estaría normalmente, pero su espacio original se conserva: los vecinos no se recorren. Su uso más importante, sin embargo, es otro: un elemento con `position: relative` se convierte en el punto de referencia para los hijos con `position: absolute`. Este es uno de los patrones más usados en CSS y lo veremos en el siguiente punto.

`absolute` (absoluto): el elemento sale por completo del flujo (los demás se acomodan como si no existiera) y se posiciona respecto a su ancestro posicionado más cercano, es decir, el antepasado más próximo que tenga un `position` distinto de `static`. Si no hay ninguno, se posiciona respecto al documento (el `<body>`).

```
.tarjeta {  
  position: relative; /* el punto de referencia */  
}  
  
.etiqueta-oferta {  
  position: absolute;  
  top: 10px;  
  right: 10px; /* pegada a la esquina superior derecha de la tarjeta */  
}
```

Este patrón (padre relative + hijo absolute) es la forma estándar de colocar insignias, etiquetas de descuento, botones de cerrar y cualquier elemento que deba ir "pegado" a una esquina o borde de otro. Si olvidas poner `relative` en el padre, el hijo absolute se posicionará respecto a la página completa y aparecerá en un lugar inesperado: es uno de los errores más comunes.

`fixed` (fijo): el elemento sale del flujo y se posiciona respecto a la ventana del navegador (el viewport), no respecto a la página. Eso significa que aunque hagas scroll, el elemento se queda clavado en su lugar. Es la técnica de las barras de navegación fijas, los botones flotantes de WhatsApp y los banners de cookies:

```
.menu-fijo {  
  position: fixed;  
  top: 0;  
  left: 0;  
  width: 100%;  
  background-color: #1a1a2e;  
  color: white;  
  padding: 15px;  
}
```

Precaución: como el elemento fijo sale del flujo, el contenido de la página puede quedar escondido debajo de él. Si tu menú fijo mide 60px de alto, dale al contenido un `padding-top` o `margin-top` de 60px (o un poco más) para compensar.

`sticky` (pegajoso): el híbrido inteligente. El elemento se comporta como `relative` hasta que el scroll lo lleva a la posición indicada, y entonces se "pega" como si fuera `fixed`:

```
.encabezado-tabla {  
  position: sticky;  
  top: 0;  
}
```

Es perfecto para encabezados de tabla que se quedan visibles al hacer scroll, o para menús que aparecen fijos solo después de pasar cierta sección. No sale del flujo, así que no necesitas compensar espacios como con `fixed`.

z-index: quién queda encima de quién

Cuando posicionas elementos (o los traslapas con márgenes negativos), a veces se enciman. La propiedad `z-index` decide el orden de apilamiento: el elemento con `z-index` mayor queda encima del de `z-index` menor.

```
.modal {  
  position: fixed;  
  z-index: 1000; /* encima de todo */  
}  
  
.menu-fijo {  
  position: fixed;  
  z-index: 100; /* encima del contenido, debajo del modal */  
}
```

Detalle importante: `z-index` solo funciona en elementos posicionados (con `position` distinto de `static`) o en elementos `flex/grid`. Si tu `z-index` "no funciona", lo más probable es que al elemento le falte un `position`. Los profesionales suelen usar escalas de cientos (100, 200, 1000) para dejar espacio entre capas por si necesitan insertar algo en medio después.

Errores comunes de esta unidad

Error 1: imágenes que rompen el layout en celular. Solución: `img {max-width: 100%; height: auto;}` como regla global.

Error 2: intentar dar `width` o márgenes verticales a un elemento `inline` y que no funcione. Solución: cambia su `display` a `inline-block` o `block` según lo que necesites.

Error 3: el hijo `absolute` que aparece en un lugar raro. Causa: olvidaste poner `position: relative` en el padre. El `absolute` busca al ancestro posicionado más cercano; si no hay ninguno, usa la página completa.

Error 4: contenido escondido debajo de un menú `fixed`. Solución: compensa con `padding-top` o `margin-top` en el contenido igual a la altura del menú fijo.

Error 5: alturas fijas en cajas con texto que luego se desborda. Solución: evita `height` fijo en contenedores de texto; usa `min-height` si necesitas un mínimo.

Error 6: `z-index` que no hace nada. Causa: al elemento le falta `position` (`relative`, `absolute`, `fixed` o `sticky`).

Buenas prácticas

Usa `max-width: 100%` en imágenes y contenedores como regla global. Prefiere `min-height` sobre `height` para contenido variable. Reserva `position: absolute` para decoraciones y elementos anclados a un padre `relative`; no construyas layouts completos con `absolute` porque se vuelven frágiles. Usa `fixed` con moderación (cada elemento fijo es espacio permanente de pantalla, valioso en celulares) y compensa siempre el espacio que ocupa. Documenta tus capas de `z-index` con comentarios para no perder la cuenta.

Ejemplos paso a paso

Ejemplo 1: Galería responsive sin frameworks

```
<div class="galeria">
  
  
  
  
</div>
```

```
.galeria {
  max-width: 1000px;
  margin: 0 auto;
  font-size: 0; /* elimina el espacio entre inline-blocks */
}

.galeria img {
  display: inline-block;
  width: 23%;
  margin: 1%;
  max-width: 100%;
  height: auto;
  border-radius: 8px;
  font-size: 16px; /* restaura el tamaño de fuente si hubiera texto */
}
```

Cuatro imágenes por fila ($23\% + 2\%$ de márgenes = 25% cada una). El truco de `font-size: 0` en el contenedor elimina el pequeño espacio que los navegadores ponen entre elementos `inline-block` por los saltos de línea del HTML. En pantallas pequeñas las imágenes se encogen

solamente gracias al porcentaje. (En la Unidad 5 verás cómo hacer esto mismo con flexbox de forma más elegante.)

Ejemplo 2: Menú de navegación fijo

```
<nav class="menu-fijo">
  <a href="#">Inicio</a>
  <a href="#">Cursos</a>
  <a href="#">Blog</a>
  <a href="#">Contacto</a>
</nav>
<main class="contenido">
  <h1>Bienvenido</h1>
  <p>Mucho contenido aquí...</p>
</main>
```

```
.menu-fijo {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  background-color: #1a1a2e;
  padding: 15px 20px;
  z-index: 100;
}

.menu-fijo a {
  color: white;
  text-decoration: none;
  margin-right: 25px;
  display: inline-block;
}

.contenido {
  padding-top: 70px; /* compensa la altura del menú fijo */
  max-width: 900px;
  margin: 0 auto;
  font-family: Arial, sans-serif;
}
```

El `padding-top: 70px` en el contenido es la compensación: sin él, el título quedaría escondido debajo del menú. El `z-index: 100` asegura que el menú quede sobre el contenido al hacer scroll.

Ejemplo 3: Tarjeta con insignia de descuento (relative + absolute)

```
<article class="producto">
  <span class="descuento">-30%</span>
  
  <h3>Tenis Runner Pro</h3>
  <p class="precio">$1,399 <span class="antes">$1,999</span></p>
</article>
```

```

.producto {
  position: relative; /* punto de referencia */
  width: 260px;
  border: 1px solid #ddd;
  border-radius: 12px;
  padding: 15px;
  font-family: Arial, sans-serif;
}

.descuento {
  position: absolute;
  top: 10px;
  right: 10px;
  background-color: #e94560;
  color: white;
  font-weight: bold;
  padding: 5px 12px;
  border-radius: 20px;
  z-index: 10;
}

.producto img {
  width: 100%;
  border-radius: 8px;
}

.antes {
  text-decoration: line-through;
  color: #999;
  font-size: 14px;
}

```

La insignia "-30%" queda clavada en la esquina superior derecha de la tarjeta gracias al patrón padre-relative/hijo-absolute. Este es exactamente el componente que usan las tiendas en línea.

Ejemplo 4: Botón flotante de ayuda

```
<button class="ayuda">?</button>
```

```
.ayuda {
  position: fixed;
  bottom: 25px;
  right: 25px;
  width: 55px;
  height: 55px;
  border-radius: 50%;
  border: none;
  background-color: #2a9d8f;
  color: white;
  font-size: 24px;
  font-weight: bold;
  cursor: pointer;
  z-index: 500;
}
```

Un botón circular fijo en la esquina inferior derecha, como los botones de chat de los sitios modernos. `bottom` y `right` lo anclan a esa esquina sin importar el scroll. El `cursor: pointer` cambia el cursor a la manita para indicar que es clicable.

Ejemplo 5: Encabezado pegajoso con sticky

```
<section class="seccion">
  <h2 class="titulo-seccion">Capítulo 1</h2>
  <p>Párrafos y párrafos de contenido...</p>
</section>
<section class="seccion">
  <h2 class="titulo-seccion">Capítulo 2</h2>
  <p>Más contenido...</p>
</section>
```

```
.titulo-seccion {
  position: sticky;
  top: 0;
  background-color: #1a1a2e;
  color: white;
  padding: 12px 20px;
  margin: 0;
  z-index: 50;
}

.seccion {
  font-family: Arial, sans-serif;
  max-width: 800px;
  margin: 0 auto;
}
```

Al hacer scroll, cada título de capítulo se queda pegado arriba mientras lees su sección, y cuando llega el siguiente capítulo lo empuja y toma su lugar. Es un efecto muy pulido que se logra con tres líneas.

Práctica

Ejercicio 1

Escribe el CSS global para que ninguna imagen rompa el layout en pantallas pequeñas, manteniendo sus proporciones.

SOLUCIÓN:

```
img {
  max-width: 100%;
  height: auto;
}
```

Dos líneas que previenen el 99 por ciento de los desbordamientos causados por imágenes. El `height: auto` es crucial: sin él, al encogerse el ancho la imagen se deformaría.

Ejercicio 2

Crea un contenedor de 900px máximo, centrado, que en pantallas pequeñas ocupe el 95% del ancho.

SOLUCIÓN:

```
.contenedor {  
  width: 95%;  
  max-width: 900px;  
  margin: 0 auto;  
}
```

En pantallas de menos de ~947px, el contenedor mide el 95% del viewport; en pantallas más grandes, se queda en 900px centrado. Es el patrón base de casi todos los sitios.

Ejercicio 3

Convierte esta lista vertical en un menú horizontal donde cada enlace sea un bloque clicable con padding:

```
<ul class="menu">  
  <li><a href="#">Inicio</a></li>  
  <li><a href="#">Cursos</a></li>  
  <li><a href="#">Contacto</a></li>  
</ul>
```

SOLUCIÓN:

```

.menu {
  list-style: none;
  padding: 0;
  background-color: #1a1a2e;
}

.menu li {
  display: inline-block;
}

.menu a {
  display: block;
  color: white;
  text-decoration: none;
  padding: 15px 25px;
}

```

Los `li` pasan a `inline-block` para ponerse en fila, y los enlaces a `block` para que todo el área del `padding` sea clicable, no solo el texto. El `list-style: none` quita las viñetas (lo veremos en la Unidad 8).

Ejercicio 4

Explica qué está mal aquí y corrígelo: la insignia "NUEVO" debería aparecer en la esquina superior derecha de la tarjeta, pero aparece en la esquina de la página.

```

.tarjeta {
  width: 300px;
  border: 1px solid #ddd;
  padding: 20px;
}

.insignia {
  position: absolute;
  top: 10px;
  right: 10px;
  background: red;
  color: white;
}

```

SOLUCIÓN: Falta `position: relative` en `.tarjeta`. Sin un ancestro posicionado, el `absolute` toma como referencia la página completa. Corrección: agregar `position: relative;` a `.tarjeta`. Este es el error número uno con `absolute`.

Ejercicio 5

Crea una ventana modal centrada que cubra la pantalla con un fondo semitransparente. Debe quedar sobre todo (z-index alto) y no moverse con el scroll.

SOLUCIÓN:

```
.fondo-modal {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.6);
  z-index: 1000;
}

.ventana-modal {
  position: fixed;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  background-color: white;
  padding: 30px;
  border-radius: 12px;
  z-index: 1001;
  width: 90%;
  max-width: 450px;
}
```

El fondo usa `rgba(0, 0, 0, 0.6)` para oscurecer la página sin ocultarla del todo. La ventana se centra con el truco `top: 50%; left: 50%; transform: translate(-50%, -50%)`: la llevas al centro y luego la recorres la mitad de su propio tamaño hacia atrás. (Los transforms los veremos en la Unidad 7, pero este truco de centrado es tan útil que vale la pena conocerlo desde ahora.)

Ejercicio 6

Haz que el encabezado de una tabla larga se quede visible al hacer scroll, sin usar fixed.

SOLUCIÓN: Usa `position: sticky`:

```
thead th {  
  position: sticky;  
  top: 0;  
  background-color: #1a1a2e;  
  color: white;  
}
```

El `background-color` es obligatorio: sin él, el contenido que pasa por debajo se transparentaría a través del encabezado. Sticky es ideal aquí porque no saca el elemento del flujo y no requiere compensar espacios.

Resumen de la unidad

En esta unidad aprendiste a controlar tamaños con `width` y `height`, y sobre todo con sus límites inteligentes: `max-width: 100%` para imágenes y contenedores que no deben desbordarse, `min-width / min-height` para mínimos garantizados, y el patrón `width: 100% + max-width + margin: 0 auto` para contenedores centrados y responsive. Dominaste los cuatro displays fundamentales: `block` (toda la línea), `inline` (solo su contenido, sin `width/height`), `inline-block` (lo mejor de ambos) y `none` (desaparece por completo, distinto de `visibility: hidden`). Y aprendiste los cinco posicionamientos: `static` (flujo normal), `relative` (se desplaza conservando su lugar, y sirve de referencia), `absolute` (fuera del flujo, respecto al ancestro posicionado), `fixed` (clavado al viewport, requiere compensar espacio) y `sticky` (híbrido que se pega al hacer scroll). Finalmente, viste que `z-index` controla el apilamiento pero solo en elementos posicionados.

Puntos clave: el patrón padre-relative/hijo-absolute para insignias y decoraciones; compensa siempre el espacio de los elementos `fixed`; y cuando el `z-index` no funcione, revisa que el elemento tenga `position`.

UNIDAD 5: FLEXBOX Y PRIMEROS PASOS CON GRID

Introducción a la unidad

Durante años, centrar un elemento verticalmente en CSS era un chiste recurrente entre programadores: había una docena de trucos, ninguno obvio, y todos se rompían en cuanto cambiaba algo. Flexbox llegó para terminar con ese sufrimiento y con muchos otros. Es un sistema de layout diseñado para distribuir espacio entre elementos en una sola dirección (una fila o una columna), y hace triviales cosas que antes eran pesadillas: centrar en ambos ejes, repartir elementos con espacios iguales, hacer que unas cajas crezcan y otras se encojan según el espacio disponible.

Grid, por su parte, es el sistema para layouts en dos dimensiones: filas Y columnas al mismo tiempo. Es la herramienta para armar la estructura completa de una página (encabezado, barra lateral, contenido, pie) o galerías y tableros complejos.

En esta unidad dominarás flexbox a fondo (contenedor e hijos, ejes, alineación, dirección y espaciado) y darás tus primeros pasos con grid creando cuadrículas de columnas. Al terminar, el centrado perfecto te tomará tres líneas y los layouts dejarán de darte miedo.

Explicación detallada

Flexbox: la idea central

Flexbox trabaja con dos roles: el contenedor flex (el padre, al que le pones `display: flex`) y los ítems flex (los hijos directos, que se vuelven flexibles automáticamente). Todo lo que sigue se entiende mejor con una analogía: el contenedor es una caja elástica y los ítems son objetos dentro de ella que se acomodan, se estiran o se encogen según las reglas que definas.

```
<div class="contenedor">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>
```

```
.contenedor {  
  display: flex;  
}  
  
.item {  
  padding: 20px;  
  background-color: #4a90d9;  
  margin: 5px;  
  color: white;  
}
```

Con solo `display: flex`, los tres divs (que normalmente se apilarían uno sobre otro) se ponen en fila automáticamente. Eso es todo lo que se necesita para empezar.

Flexbox piensa en dos ejes: el eje principal (main axis) y el eje transversal (cross axis). Por defecto, el eje principal es horizontal (de izquierda a derecha) y el transversal es vertical (de arriba hacia abajo). Casi todas las propiedades de flexbox se refieren a uno de estos dos ejes, así que grábate esta imagen mental: una flecha horizontal (principal) cruzada por una flecha vertical (transversal).

Propiedades del contenedor flex

`justify-content` controla la alineación de los ítems sobre el eje principal (horizontal, por defecto). Sus valores más usados:

- `flex-start`: los ítems se agrupan al inicio (izquierda). Es el valor por defecto.
- `flex-end`: se agrupan al final (derecha).
- `center`: se centran. El famoso centrado horizontal en una línea.
- `space-between`: el primer ítem pegado al inicio, el último pegado al final, y el espacio sobrante repartido equitativamente entre los ítems. Perfecto para barras de navegación con el logo a la izquierda y el menú a la derecha.
- `space-around`: cada ítem tiene la misma cantidad de espacio a su alrededor (el espacio entre dos ítems es el doble que el de los extremos).
- `space-evenly`: el espacio entre todos los ítems y los bordes es exactamente igual.

`align-items` controla la alineación sobre el eje transversal (vertical, por defecto):

- `stretch`: los ítems se estiran para llenar la altura del contenedor. Es el valor por defecto, y por eso los ítems flex suelen verse todos de la misma altura sin que hagas nada.

- `flex-start`: se alinean arriba.
- `flex-end`: se alinean abajo.
- `center`: se centran verticalmente. Combinado con `justify-content: center`, logras el centrado perfecto en ambos ejes con dos líneas.

El centrado perfecto, la receta más famosa de flexbox:

```
.centrado {  
  display: flex;  
  justify-content: center;  
  align-items: center;  
  height: 300px;  
}
```

Cualquier cosa que pongas dentro de `.centrado` quedará perfectamente centrada, horizontal y verticalmente. Tres líneas para resolver lo que antes tomaba trucos oscuros. Memorízala: la usarás cientos de veces.

`flex-direction` cambia la dirección del eje principal: `row` (fila, por defecto), `row-reverse` (fila al revés), `column` (columna: los ítems se apilan verticalmente) y `column-reverse`.

Cuando cambias a `column`, los ejes se intercambian: `justify-content` ahora controla la vertical y `align-items` la horizontal. Este es el punto que más confunde: las propiedades no significan "horizontal" o "vertical" en absoluto, significan "eje principal" y "eje transversal", y los ejes dependen de `flex-direction`.

`flex-wrap` decide qué pasa cuando los ítems no caben en una línea: `nowrap` (por defecto, los ítems se encogen para caber, aunque se vean apretados), `wrap` (los ítems que no caben bajan a la siguiente línea, como el texto de un párrafo) y `wrap-reverse`. Con `wrap` puedes hacer galerías y rejillas que se adaptan solas al ancho disponible.

`gap` pone espacio uniforme entre los ítems, sin necesidad de márgenes:

```
.contenedor {  
  display: flex;  
  gap: 15px;  
}
```

Es más limpio que ponerle margin a cada ítem porque no agrega espacio extra en los bordes del contenedor (el margin del primer y último ítem sí lo haría). Existe también `row-gap` y `column-gap` por si quieres espaciados distintos en cada dirección. `gap` funciona también en grid, y es una de las propiedades favoritas de los profesionales modernos.

Propiedades de los ítems flex

Hasta ahora vimos propiedades del contenedor. Pero cada hijo también puede tener sus propias reglas.

`flex-grow` define cuánto crece un ítem para ocupar el espacio sobrante. Si todos tienen `flex-grow: 1`, el espacio se reparte en partes iguales. Si uno tiene `flex-grow: 2` y otro `flex-grow: 1`, el primero recibe el doble de espacio extra que el segundo. Es proporcional, no absoluto: los números solo importan en relación con los demás.

`flex-shrink` es lo contrario: cuánto se encoge un ítem cuando falta espacio. El valor por defecto es 1 (todos se encogen por igual). Con `flex-shrink: 0`, un ítem se niega a encogerse y mantiene su tamaño aunque los demás se aprieten.

`flex-basis` define el tamaño inicial del ítem antes de que se reparta el espacio sobrante o faltante. Puedes pensarlo como el "width sugerido" dentro de un contenedor flex:

`flex-basis: 200px` significa "empieza con 200px y luego crece o encoge según las reglas".

La abreviatura `flex` combina las tres en el orden grow, shrink, basis: `flex: 1` significa `flex-grow: 1; flex-shrink: 1; flex-basis: 0%`. Es la forma más común de decir "reparte el espacio equitativamente":

```
.item {
  flex: 1; /* todos los ítems iguales */
}
```

`align-self` permite que un ítem individual ignore el `align-items` del contenedor y se alinee a su manera: `align-self: flex-start`, `center`, `flex-end`, etc. Útil cuando quieres que un solo ítem se comporte distinto a sus hermanos.

`order` cambia el orden visual de los ítems sin tocar el HTML. Todos los ítems tienen `order: 0` por defecto; un ítem con `order: -1` se pondrá primero, uno con `order: 2` irá después de los de orden 0 y 1. Úsalo con cuidado: cambia solo lo visual, no el orden del documento para lectores de pantalla ni para la navegación con teclado, así que no lo uses para reordenar contenido importante de forma que confunda.

Grid básico: cuadrículas en dos dimensiones

Si flexbox es para una dimensión (una fila o una columna), grid es para dos: filas y columnas al mismo tiempo. La idea central: defines una cuadrícula en el contenedor y los hijos se colocan solos en las celdas.

```
<div class="rejilla">
  <div class="celda">1</div>
  <div class="celda">2</div>
  <div class="celda">3</div>
  <div class="celda">4</div>
  <div class="celda">5</div>
  <div class="celda">6</div>
</div>
```

```
.rejilla {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  gap: 15px;
}

.celda {
  background-color: #6c5ce7;
  color: white;
  padding: 30px;
  text-align: center;
}
```

`grid-template-columns: 1fr 1fr 1fr` crea tres columnas iguales. La unidad `fr` (fracción) es la estrella de grid: representa una fracción del espacio disponible. `1fr 1fr 1fr` = tres columnas que se reparten el espacio en partes iguales. `2fr 1fr` = la primera columna el doble de ancha que la segunda. Puedes mezclar unidades: `200px 1fr 1fr` crea una columna fija de 200px y dos que se reparten el resto.

Los seis divs se colocan automáticamente: 1, 2, 3 en la primera fila; 4, 5, 6 en la segunda. Las filas se crean solas según se necesiten (filas implícitas), con altura automática según el contenido. Si quieres controlar las filas explícitamente, usas `grid-template-rows` con la misma lógica.

La función `repeat()` evita escribir columnas repetidas:

```
grid-template-columns: repeat(3, 1fr); /* igual que 1fr 1fr 1fr */
grid-template-columns: repeat(4, 200px); /* 4 columnas de 200px */
```

Y `gap` (o `grid-gap` en sintaxis antigua) separa las celdas, igual que en flexbox.

¿Cuándo usar flexbox y cuándo grid? Regla práctica: si tu layout es esencialmente una línea (una barra de navegación, una fila de botones, una lista de tarjetas en una fila), usa flexbox. Si es una cuadrícula real con filas y columnas que deben alinearse en ambas direcciones (el esqueleto de la página, una galería, un tablero), usa grid. Y no son excluyentes: es muy común tener un layout grid para la página completa y flexbox dentro de cada celda para alinear su contenido. Los profesionales los combinan constantemente.

Errores comunes de esta unidad

Error 1: aplicar propiedades de ítem al contenedor o viceversa. `justify-content` y `align-items` van en el contenedor; `flex-grow`, `align-self` y `order` van en los hijos. Si "no funciona", revisa en qué elemento pusiste la propiedad.

Error 2: olvidar que al cambiar `flex-direction` a `column`, los ejes se intercambian. Si tu `justify-content: center` de pronto centra vertical en vez de horizontal, es porque cambiaste la dirección.

Error 3: usar `float` para layouts modernos. Float fue la técnica de los 2000; hoy flexbox y grid lo reemplazan en casi todo. Si ves un tutorial usando floats para columnas, es un tutorial viejo.

Error 4: en grid, esperar que `1fr` funcione si el contenedor no tiene ancho definido. Las fracciones se calculan sobre el espacio disponible del contenedor; asegúrate de que el contenedor tenga un ancho (normalmente el 100% de su padre, que es lo natural en bloques).

Error 5: abusar de `order` para reordenar contenido importante, rompiendo la accesibilidad. El orden visual debe coincidir con el orden lógico del documento siempre que sea posible.

Buenas prácticas

Para centrar: el combo `display: flex; justify-content: center; align-items: center;`. Para espaciado entre ítems: `gap` en vez de márgenes. Para layouts de página: grid para la estructura general, flexbox para los componentes internos. Nombra tus clases por su función (`.barra-navegacion`, `.galeria-productos`) y no por la técnica (`.flex-container`), porque mañana podrías cambiar la técnica y el nombre quedaría obsoleto.

Ejemplos paso a paso

Ejemplo 1: Barra de navegación con space-between

```
<nav class="barra">
  <span class="logo">MiSitio</span>
  <div class="enlaces">
    <a href="#">Inicio</a>
    <a href="#">Cursos</a>
    <a href="#">Contacto</a>
  </div>
</nav>
```

```
.barra {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: #1a1a2e;
  padding: 15px 30px;
}

.logo {
  color: white;
  font-size: 24px;
  font-weight: bold;
  font-family: Arial, sans-serif;
}

.enlaces {
  display: flex;
  gap: 25px;
}

.enlaces a {
  color: white;
  text-decoration: none;
  font-family: Arial, sans-serif;
}
```

El logo queda a la izquierda y los enlaces a la derecha, todo verticalmente centrado. space-between empuja los dos grupos a los extremos. Este es el layout de navegación más común de la web moderna.

Ejemplo 2: Tres tarjetas iguales con flex

```
<div class="precios">
  <div class="plan">
    <h3>Básico</h3>
    <p class="monto">$99</p>
    <p>10 cursos</p>
    <button>Elegir</button>
  </div>
  <div class="plan destacado">
    <h3>Pro</h3>
    <p class="monto">$199</p>
    <p>Cursos ilimitados</p>
    <button>Elegir</button>
  </div>
  <div class="plan">
    <h3>Empresas</h3>
    <p class="monto">$499</p>
    <p>Todo + soporte</p>
    <button>Elegir</button>
  </div>
</div>
```

```

.precios {
  display: flex;
  gap: 20px;
  max-width: 1000px;
  margin: 0 auto;
  font-family: Arial, sans-serif;
}

.plan {
  flex: 1;
  border: 1px solid #ddd;
  border-radius: 12px;
  padding: 30px;
  text-align: center;
}

.plan.destacado {
  border: 2px solid #e94560;
  transform: scale(1.05);
}

.monto {
  font-size: 36px;
  font-weight: bold;
  color: #1a1a2e;
}

.plan button {
  background-color: #1a1a2e;
  color: white;
  border: none;
  padding: 12px 30px;
  border-radius: 25px;
  cursor: pointer;
}

.plan.destacado button {
  background-color: #e94560;
}

```

Las tres tarjetas miden lo mismo gracias a `flex: 1`, sin importar cuánto texto tenga cada una (además, `align-items: stretch` por defecto iguala sus alturas). La destacada tiene borde de color y está ligeramente agrandada. Es la clásica tabla de precios.

Ejemplo 3: Layout en columna para un formulario centrado

```
<form class="formulario">
  <h2>Regístrate</h2>
  <input type="text" placeholder="Nombre">
  <input type="email" placeholder="Correo">
  <input type="password" placeholder="Contraseña">
  <button type="submit">Crear cuenta</button>
</form>
```

```
.formulario {
  display: flex;
  flex-direction: column;
  gap: 15px;
  max-width: 400px;
  margin: 50px auto;
  padding: 40px;
  border: 1px solid #ddd;
  border-radius: 12px;
  font-family: Arial, sans-serif;
}

.formulario h2 {
  text-align: center;
  margin: 0 0 10px 0;
}

.formulario input {
  padding: 12px;
  border: 1px solid #ccc;
  border-radius: 8px;
  font-size: 16px;
}

.formulario button {
  padding: 14px;
  background-color: #2a9d8f;
  color: white;
  border: none;
  border-radius: 8px;
  font-size: 16px;
  font-weight: bold;
  cursor: pointer;
}
```

Con `flex-direction: column` y `gap`, los campos se apilan con espaciado perfecto sin un solo `margin`. Antes de `flexbox`, este formulario requería márgenes en cada campo y trucos para que ocuparan todo el ancho.

Ejemplo 4: Galería con grid y repeat

```
<div class="galeria">
  <div class="foto">1</div>
  <div class="foto">2</div>
  <div class="foto">3</div>
  <div class="foto">4</div>
  <div class="foto">5</div>
  <div class="foto">6</div>
  <div class="foto">7</div>
  <div class="foto">8</div>
</div>
```

```
.galeria {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  gap: 15px;
  max-width: 1100px;
  margin: 0 auto;
}

.foto {
  background-color: #6c5ce7;
  color: white;
  height: 150px;
  display: flex;
  justify-content: center;
  align-items: center;
  font-size: 28px;
  font-family: Arial, sans-serif;
  border-radius: 10px;
}
```

Ocho celdas en 4 columnas: grid las acomoda en 2 filas automáticamente. Y nota la combinación: grid para la cuadrícula, flexbox dentro de cada celda para centrar el número. Así trabajan juntos en el mundo real.

Ejemplo 5: Layout de página con grid (vista previa)

```
<div class="pagina">
  <header class="enc">Encabezado</header>
  <aside class="lat">Barra lateral</aside>
  <main class="cont">Contenido principal</main>
  <footer class="pie">Pie de página</footer>
</div>
```

```
.pagina {
  display: grid;
  grid-template-columns: 220px 1fr;
  gap: 15px;
  max-width: 1100px;
  margin: 0 auto;
  font-family: Arial, sans-serif;
}

.enc, .pie {
  grid-column: 1 / -1;
  background-color: #1a1a2e;
  color: white;
  padding: 20px;
  border-radius: 8px;
}

.lat {
  background-color: #f0f0f0;
  padding: 20px;
  border-radius: 8px;
}

.cont {
  background-color: white;
  border: 1px solid #ddd;
  padding: 20px;
  border-radius: 8px;
  min-height: 300px;
}
```

Dos columnas: 220px fijos para la barra lateral y el resto para el contenido. El encabezado y el pie ocupan ambas columnas con `grid-column: 1 / -1` (desde la primera línea hasta la última). Es el esqueleto de miles de sitios reales, en 30 líneas.

Práctica

Ejercicio 1

Centra un texto perfecta y horizontalmente dentro de un div de 400px de alto usando flexbox.

SOLUCIÓN:

```
.caja {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 400px;
  background-color: #f0f0f0;
}
```

`justify-content: center` centra en el eje principal (horizontal) y `align-items: center` en el transversal (vertical). Sin flexbox esto requería trucos con tablas o posicionamiento absoluto.

Ejercicio 2

Haz una fila de 4 botones con el mismo tamaño que ocupen todo el ancho disponible con 10px de separación entre ellos.

SOLUCIÓN:

```
.fila {
  display: flex;
  gap: 10px;
}

.fila button {
  flex: 1;
  padding: 15px;
}
```

El `flex: 1` en cada botón hace que los cuatro crezcan por igual hasta llenar el contenedor. El `gap` los separa sin espacios extra en los bordes.

Ejercicio 3

Crea un layout donde una barra lateral mida 250px y el contenido ocupe el resto, usando flexbox.

SOLUCIÓN:

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

.barra-lateral {
  flex: 0 0 250px; /* no crece, no se encoge, base de 250px */
}

.contenido {
  flex: 1; /* ocupa todo el espacio restante */
}
```

`flex: 0 0 250px` es la forma de decir "quédate en 250px pase lo que pase": `grow 0` (no crezcas), `shrink 0` (no te encojas), `basis 250px`. El contenido con `flex: 1` toma todo lo demás.

Ejercicio 4

Explica qué hace este código y qué vería el usuario:

```
.menu {
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 10px;
}
```

SOLUCIÓN: Convierte al contenedor en flex vertical (columna). Los ítems se apilan uno sobre otro, centrados horizontalmente (`align-items: center` en dirección columna controla el eje horizontal), con 10px de separación entre ellos. El usuario vería una lista vertical centrada, como un menú de botones apilados al centro de la página.

Ejercicio 5

Haz una cuadrícula de 3 columnas iguales con 20px de separación usando grid, donde la primera celda ocupe dos columnas.

SOLUCIÓN:

```
.rejilla {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 20px;
}

.rejilla .primera {
  grid-column: span 2;
}
```

`grid-column: span 2` hace que esa celda abarque dos columnas de la cuadrícula. Las demás celdas se acomodan automáticamente en los espacios restantes.

Ejercicio 6

Corrige este código: las tarjetas deberían estar en fila con espacio entre ellas, pero se ven apiladas y pegadas.

```
.tarjetas {
  display: flex;
}

.tarjeta {
  width: 30%;
  margin: 0;
}
```

SOLUCIÓN: Falta el espaciado. La forma moderna es agregar `gap` al contenedor (o `justify-content: space-between`). Además, con `width: 30%` y sin `gap`, tres tarjetas suman 90% y queda un 10% vacío al final. Mejor:

```
.tarjetas {  
  display: flex;  
  gap: 5%;  
}  
.tarjeta {  
  flex: 1;  
}
```

Con `flex: 1` y `gap`, las tarjetas se reparten el espacio equitativamente con separación uniforme. Regla: prefiere `flex: 1` + `gap` sobre porcentajes manuales.

Resumen de la unidad

En esta unidad dominaste flexbox, el sistema de layout unidimensional: el contenedor (`display: flex`) y sus ítems, los dos ejes (principal y transversal), `justify-content` (eje principal: `flex-start`, `flex-end`, `center`, `space-between`, `space-around`, `space-evenly`), `align-items` (eje transversal: `stretch`, `flex-start`, `flex-end`, `center`), `flex-direction` (`row`, `column` y sus reversos, que intercambian los ejes), `flex-wrap` (`wrap` para múltiples líneas), `gap` (espaciado limpio), y las propiedades de los ítems: `flex-grow`, `flex-shrink`, `flex-basis`, la abreviatura `flex`, `align-self` y `order`. Aprendiste la receta del centrado perfecto en tres líneas. Y diste tus primeros pasos con grid: `display: grid`, `grid-template-columns` con la unidad `fr`, la función `repeat()`, y la regla de oro: flexbox para líneas, grid para cuadrículas, combinados cuando convenga.

Puntos clave: memoriza el centrado con `flex`; usa `gap` en vez de márgenes entre ítems; recuerda que `flex-direction: column` intercambia los ejes; y no uses floats para layouts modernos.

UNIDAD 6: GRID AVANZADO, PSEUDOCCLASES Y PSEUDO-ELEMENTOS

Introducción a la unidad

En la unidad anterior aprendiste a crear cuadrículas simples con grid. Ahora vas a llevarlo al siguiente nivel: dibujar layouts completos con nombres de áreas, como si fuera un plano de arquitecto, y controlar el tamaño de filas y columnas con funciones inteligentes que se adaptan solas. Después entrarás al mundo de los selectores dinámicos: las pseudoclasses, que aplican estilos según el estado o la posición de un elemento (cuando el mouse pasa encima, cuando es el primer hijo, cuando un campo está enfocado), y los pseudo-elementos, que te permiten estilizar partes de un elemento o insertar decoraciones sin tocar el HTML.

Estos tres temas comparten una idea: escribir menos HTML y lograr más con CSS. Un buen desarrollador frontend no llena su HTML de divs decorativos; usa grid para la estructura, pseudoclasses para los estados y pseudo-elementos para los adornos. Al terminar esta unidad tu CSS será más limpio, más potente y más profesional.

Explicación detallada

Grid avanzado: dibujando layouts con áreas

La forma más visual y poderosa de usar grid es `grid-template-areas`: literalmente dibujas tu layout con palabras, fila por fila, como un mapa.

```
<div class="pagina">
  <header class="cabecera">Encabezado</header>
  <nav class="menu">Menú</nav>
  <main class="contenido">Contenido</main>
  <aside class="lateral">Lateral</aside>
  <footer class="pie">Pie</footer>
</div>
```

```

.pagina {
  display: grid;
  grid-template-columns: 200px 1fr 200px;
  grid-template-rows: auto 1fr auto;
  grid-template-areas:
    "cabecera cabecera cabecera"
    "menu contenido lateral"
    "pie pie pie";
  gap: 15px;
  min-height: 100vh;
}

.cabecera { grid-area: cabecera; }
.menu { grid-area: menu; }
.contenido { grid-area: contenido; }
.lateral { grid-area: lateral; }
.pie { grid-area: pie; }

```

Lee el `grid-template-areas` como un dibujo: la primera fila tiene tres celdas llamadas "cabecera" (el encabezado abarca las tres columnas), la segunda fila tiene "menu", "contenido" y "lateral", y la tercera fila es todo "pie". Cada elemento se coloca en su área con `grid-area`. Si mañana quieres que la barra lateral vaya a la izquierda, solo cambias el dibujo: "lateral contenido menu". No tocas el HTML. Esa es la magia: el layout se vuelve un texto visual fácil de leer y modificar.

Reglas del dibujo: cada fila va entre comillas, todas las filas deben tener el mismo número de celdas, y los nombres se repiten para indicar que un área abarca varias celdas. Puedes dejar celdas vacías con un punto: "menu contenido ." dejaría la tercera celda de esa fila vacía.

Además de las áreas, grid tiene funciones de tamaño inteligentes. `minmax(min, max)` define un rango: la columna o fila no será menor que el mínimo ni mayor que el máximo:

```

grid-template-columns: 200px minmax(300px, 1fr) 200px;

```

La columna central mide mínimo 300px y crece hasta ocupar el espacio restante. Es perfecto para contenido que no debe comprimirse demasiado pero sí aprovechar pantallas grandes.

`auto-fit` y `auto-fill` combinados con `repeat()` crean cuadrículas responsive sin media queries:

```
.galeria {  
  display: grid;  
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));  
  gap: 20px;  
}
```

Esta es una de las líneas más famosas del CSS moderno. Léela así: "crea tantas columnas como quepan, cada una de mínimo 250px, y reparte el espacio sobrante equitativamente". En una pantalla ancha caben 4 columnas; en una tablet, 2; en un celular, 1. Todo automático, sin escribir un solo media query. La diferencia entre `auto-fit` y `auto-fill` es sutil: con `auto-fit`, las columnas vacías colapsan y las ocupadas se expanden; con `auto-fill`, se reservan los espacios aunque estén vacíos. Para galerías, `auto-fit` suele ser lo que quieres.

También puedes colocar ítems manualmente con `grid-column` y `grid-row` usando números de línea:

```
.destacado {  
  grid-column: 1 / 3; /* desde la línea 1 hasta la línea 3 = abarca 2 columnas */  
  grid-row: 1 / 2;  
}
```

Las líneas de la cuadrícula se numeran desde 1. `1 / 3` significa "empieza en la línea 1 y termina en la línea 3", abarcando las columnas 1 y 2. Es útil para elementos destacados que deben ocupar más espacio que los demás.

Pseudoclases: estilos que reaccionan

Una pseudoclase es una palabra clave que se agrega al selector con dos puntos (`:`) y que selecciona elementos según un estado especial o una posición, no según su nombre o clase. Son la base de la interactividad en CSS puro.

Las pseudoclases de interacción responden a lo que hace el usuario:

- `:hover` - cuando el puntero del mouse está sobre el elemento. La más usada de todas: botones que cambian de color, tarjetas que se elevan, menús que se despliegan.
- `:active` - mientras el elemento está siendo presionado (entre el clic y soltar).
- `:focus` - cuando el elemento tiene el foco, típicamente un campo de formulario donde estás escribiendo o un botón al que llegaste con el tabulador. Esencial para accesibilidad: nunca elimines el indicador de foco sin reemplazarlo por otro estilo visible.

- `:visited` - enlaces que el usuario ya visitó (el navegador los muestra de otro color por defecto, morado históricamente).
- `:link` - enlaces no visitados.

```
.boton {
  background-color: #1a1a2e;
  color: white;
  padding: 12px 30px;
  border: none;
  border-radius: 8px;
  cursor: pointer;
}

.boton:hover {
  background-color: #e94560;
}

.boton:active {
  transform: scale(0.97);
}
```

El botón es azul oscuro, se vuelve rojo al pasar el mouse y se encoge ligeramente al presionarlo. Tres estados, cero JavaScript. Esta es la interactividad básica que todo sitio necesita.

Las pseudoclases de posición seleccionan elementos según su lugar entre sus hermanos:

- `:first-child` - el primer hijo de su padre.
- `:last-child` - el último hijo.
- `:nth-child(n)` - el hijo número n. Acepta números (`:nth-child(2)`), palabras (`:nth-child(odd)` para impares, `:nth-child(even)` para pares) y fórmulas (`:nth-child(3n)` para cada tercero).
- `:nth-child(2n+1)` selecciona los impares con fórmula; `:nth-child(-n+3)` selecciona los tres primeros.

El caso de uso clásico es el "cebreado" de tablas y listas: filas alternadas de colores para facilitar la lectura:

```
tbody tr:nth-child(even) {  
  background-color: #f5f5f5;  
}
```

Otras pseudoclasas útiles: `:first-of-type` y `:last-of-type` (el primero/último de su tipo de etiqueta entre sus hermanos, útil cuando hay etiquetas mezcladas), `:not(selector)` (selecciona lo que NO coincide: `p:not(.destacado)` selecciona los párrafos sin la clase destacado), `:checked` (checkboxes y radios marcados, base de muchos trucos de CSS puro como menús hamburguesa sin JavaScript), `:disabled` (campos deshabilitados) y `:empty` (elementos sin contenido).

Un detalle de sintaxis que causa errores: `en:nth-child(2)` el conteo incluye a TODOS los hermanos, no solo a los del mismo tipo. Si tienes un `<h2>` seguido de párrafos y escribes `p:nth-child(2)`, no selecciona el segundo párrafo: selecciona los párrafos que además sean el segundo hijo de su padre. Para "el segundo párrafo entre sus hermanos sin importar qué haya antes", usa `:nth-of-type(2)`.

Pseudo-elementos: estilo sin tocar el HTML

Los pseudo-elementos se escriben con doble dos puntos (`::`) y seleccionan una parte de un elemento o crean un elemento virtual que no existe en el HTML. Los dos más importantes son `::before` y `::after`.

`::before` inserta contenido antes del contenido del elemento; `::after` lo inserta después. El contenido se define con la propiedad `content`:

```
.aviso::before {  
  content: "! ";  
}  
  
.cita::after {  
  content: " ";  
  font-size: 30px;  
  color: #e94560;  
}
```

Esto agrega un símbolo de advertencia antes de cada aviso y una comilla decorativa después de cada cita, sin modificar el HTML. Los pseudo-elementos son cajas reales: puedes darles display, colores, tamaños, posicionarlos, todo. Se usan para iconos decorativos, viñetas personalizadas, líneas ornamentales, tooltips y cientos de detalles visuales.

Un patrón clásico: la línea decorativa bajo un título:

```
.titulo-seccion {
  text-align: center;
}

.titulo-seccion::after {
  content: "";
  display: block;
  width: 60px;
  height: 4px;
  background-color: #e94560;
  margin: 10px auto 0;
  border-radius: 2px;
}
```

Con `content: ""` (vacío) creas una cajita invisible a la que le das tamaño y color: una línea roja centrada bajo el título. Sin este truco necesitarías un `<div>` extra en el HTML solo para la decoración.

Otros pseudo-elementos útiles:

- `::first-letter` - estiliza la primera letra del texto (la clásica letra capital de los periódicos y libros).
- `::first-line` - estiliza la primera línea del párrafo.
- `::selection` - estiliza el texto que el usuario selecciona con el mouse (puedes cambiar el típico fondo azul por los colores de tu marca).
- `::placeholder` - estiliza el texto de ayuda dentro de los campos de formulario (`placeholder="Escribe tu nombre"`).
- `::marker` - estiliza las viñetas o números de las listas.

```

p::first-letter {
  font-size: 48px;
  float: left;
  line-height: 1;
  margin-right: 8px;
  color: #e94560;
  font-weight: bold;
}

::selection {
  background-color: #e94560;
  color: white;
}

input::placeholder {
  color: #aaa;
  font-style: italic;
}

```

Diferencia clave entre pseudoclase y pseudo-elemento: la pseudoclase (:hover) selecciona el elemento completo según su estado; el pseudo-elemento (::before) selecciona una parte del elemento o crea contenido nuevo. Sintaxis: un dos puntos para pseudoclases, dos para pseudo-elementos. (Los navegadores aceptan :before con un solo dos puntos por compatibilidad histórica, pero la forma correcta moderna es ::before.)

Errores comunes de esta unidad

Error 1: dibujar `grid-template-areas` con filas de distinto número de celdas. Todas las filas deben tener el mismo número de columnas o el layout se rompe.

Error 2: olvidar `content` en `::before` / `::after`. Sin la propiedad `content` (aunque sea vacía: `content: ""`), el pseudo-elemento no se genera y nada aparece.

Error 3: confundir `:nth-child` con `:nth-of-type`. Recuerda: `:nth-child` cuenta todos los hermanos; `:nth-of-type` cuenta solo los de ese tipo de etiqueta.

Error 4: quitar el `:focus` visible sin reemplazo. Muchos diseñadores ponen `outline: none` porque "se ve feo", dejando a los usuarios de teclado sin saber dónde están. Si lo quitas, pon un estilo de foco alternativo (un borde de color, una sombra).

Error 5: usar `auto-fill` cuando querías `auto-fit` en galerías, dejando columnas vacías fantasma en pantallas anchas.

Buenas prácticas

Dibuja tus layouts con `grid-template-areas` cuando la estructura sea compleja: el código se vuelve autodocumentado. Usa `repeat(auto-fit, minmax(...))` para rejillas responsive sin media queries. Da siempre feedback visual en `:hover` y `:active` para elementos clicables (el usuario necesita saber que algo es interactivo). Mantén un `:focus` visible y bonito. Y usa `::before` / `::after` para decoraciones en vez de ensuciar el HTML con divs vacíos.

Ejemplos paso a paso

Ejemplo 1: Layout completo con áreas nombradas

```
<div class="sitio">
  <header class="cab">Mi Tienda</header>
  <nav class="men">Inicio | Productos | Contacto</nav>
  <main class="con">
    <h2>Bienvenido</h2>
    <p>Contenido principal de la página...</p>
  </main>
  <aside class="lat">
    <h3>Ofertas</h3>
    <p>Descuentos de la semana.</p>
  </aside>
  <footer class="pie">© 2026 Mi Tienda</footer>
</div>
```

```

.sitio {
  display: grid;
  grid-template-columns: 180px 1fr 220px;
  grid-template-rows: auto auto 1fr auto;
  grid-template-areas:
    "cab cab cab"
    "men men men"
    "lat con con"
    "pie pie pie";
  gap: 15px;
  max-width: 1200px;
  margin: 0 auto;
  min-height: 100vh;
  font-family: Arial, sans-serif;
}

.cab { grid-area: cab; background: #1a1a2e; color: white; padding: 20px; }
.men { grid-area: men; background: #16213e; color: white; padding: 12px 20px; }
.con { grid-area: con; padding: 20px; }
.lat { grid-area: lat; background: #f0f0f0; padding: 20px; }
.pie { grid-area: pie; background: #1a1a2e; color: white; padding: 15px;
text-align: center; }

```

Mira el dibujo: encabezado y menú ocupan todo el ancho, luego tres columnas (lateral, contenido, contenido), y el pie todo el ancho. Si quieres el lateral a la derecha, cambias una línea del dibujo. Así de fácil se reestructura una página completa.

Ejemplo 2: Galería responsive automática

```

<div class="galeria">
  <div class="foto">Foto 1</div>
  <div class="foto">Foto 2</div>
  <div class="foto">Foto 3</div>
  <div class="foto">Foto 4</div>
  <div class="foto">Foto 5</div>
  <div class="foto">Foto 6</div>
</div>

```

```

.galeria {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
  gap: 20px;
  padding: 20px;
}

.foto {
  background: linear-gradient(135deg, #667eea, #764ba2);
  height: 160px;
  border-radius: 12px;
  display: flex;
  justify-content: center;
  align-items: center;
  color: white;
  font-family: Arial, sans-serif;
  font-size: 20px;
}

```

Sin media queries, sin JavaScript: la galería se adapta sola. Prueba cambiar el ancho de la ventana del navegador y observa cómo las columnas aparecen y desaparecen mágicamente.

Ejemplo 3: Botones con todos los estados

```

<button class="btn">Comprar ahora</button>
<button class="btn" disabled>Agotado</button>
<input type="text" class="campo" placeholder="Escribe tu nombre">

```

```
.btn {
  background-color: #2a9d8f;
  color: white;
  border: none;
  padding: 14px 35px;
  font-size: 16px;
  border-radius: 8px;
  cursor: pointer;
  font-family: Arial, sans-serif;
}

.btn:hover {
  background-color: #21867a;
}

.btn:active {
  transform: scale(0.96);
}

.btn:disabled {
  background-color: #cccccc;
  cursor: not-allowed;
}

.campo {
  padding: 12px;
  border: 2px solid #ddd;
  border-radius: 8px;
  font-size: 16px;
}

.campo:focus {
  border-color: #2a9d8f;
  outline: none;
  box-shadow: 0 0 0 3px rgba(42, 157, 143, 0.2);
}

.campo::placeholder {
  color: #aaa;
  font-style: italic;
}
```

Cuatro estados del botón (normal, hover, active, disabled) y dos del campo (focus con anillo de color en vez del outline por defecto, placeholder estilizado). Nota que al quitar el `outline` en `:focus` lo reemplazamos con un `box-shadow` visible: accesibilidad preservada.

Ejemplo 4: Lista de precios cebreada con `nth-child`

```
<ul class="precios">
  <li>Laptop <span>$15,999</span></li>
  <li>Mouse <span>$499</span></li>
  <li>Teclado <span>$899</span></li>
  <li>Monitor <span>$4,299</span></li>
  <li>Audífonos <span>$1,299</span></li>
</ul>
```

```
.precios {
  list-style: none;
  padding: 0;
  max-width: 400px;
  font-family: Arial, sans-serif;
}

.precios li {
  display: flex;
  justify-content: space-between;
  padding: 12px 15px;
}

.precios li:nth-child(even) {
  background-color: #f5f5f5;
}

.precios li:nth-child(odd) {
  background-color: white;
}

.precios li:hover {
  background-color: #e3f2fd;
}

.precios li:first-child {
  border-top: 2px solid #1a1a2e;
}

.precios li:last-child {
  border-bottom: 2px solid #1a1a2e;
}
```

Filas alternadas para legibilidad, resaltado al pasar el mouse, y bordes solo en la primera y última fila. Todo con pseudoclases, sin clases extra en el HTML.

Ejemplo 5: Título decorado y letra capital

```
<article class="nota">
  <h2 class="titulo-nota">El arte del CSS</h2>
  <p class="entrada">CSS transformó la web de documentos grises...</p>
  <p>Hoy en día, cada sitio que admiras...</p>
</article>
```

```
.titulo-nota {
  text-align: center;
  font-family: Georgia, serif;
  color: #1a1a2e;
}

.titulo-nota::after {
  content: "";
  display: block;
  width: 70px;
  height: 4px;
  background-color: #e94560;
  margin: 12px auto 0;
  border-radius: 2px;
}

.entrada::first-letter {
  font-size: 52px;
  float: left;
  line-height: 0.9;
  margin: 4px 8px 0 0;
  color: #e94560;
  font-weight: bold;
  font-family: Georgia, serif;
}

::selection {
  background-color: #e94560;
  color: white;
}
```

Línea decorativa bajo el título, letra capital en el primer párrafo y selección de texto con los colores de la marca. Tres detalles editoriales que elevan el diseño, todos con pseudo-elementos.

Práctica

Ejercicio 1

Dibuja con grid-template-areas un layout de blog: encabezado completo arriba, luego contenido (2/3) y barra lateral (1/3), y pie completo abajo.

SOLUCIÓN:

```
.blog {  
  display: grid;  
  grid-template-columns: 2fr 1fr;  
  grid-template-areas:  
    "cabecera cabecera"  
    "contenido lateral"  
    "pie pie";  
  gap: 20px;  
}  
  
.cabecera { grid-area: cabecera; }  
.contenido { grid-area: contenido; }  
.lateral { grid-area: lateral; }  
.pie { grid-area: pie; }
```

Con 2fr 1fr, el contenido mide el doble que la barra lateral. El dibujo deja clarísima la estructura de un vistazo.

Ejercicio 2

Crea una galería que muestre automáticamente 2, 3 o 4 columnas según el ancho, con celdas de mínimo 200px.

SOLUCIÓN:

```
.galeria {  
  display: grid;  
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));  
  gap: 15px;  
}
```

Una sola línea hace todo el trabajo responsive. En el celular (menos de ~430px) habrá 1 columna, luego 2, 3 y 4 según crezca la pantalla.

Ejercicio 3

Haz que los enlaces de un menú cambien así: normales en azul, al pasar el mouse en rojo y subrayados, al presionar en verde.

SOLUCIÓN:

```
.menu a {  
  color: blue;  
  text-decoration: none;  
}  
  
.menu a:hover {  
  color: red;  
  text-decoration: underline;  
}  
  
.menu a:active {  
  color: green;  
}
```

El orden importa: :hover después del estilo base y :active después de :hover, porque en el momento del clic ambos aplican y debe ganar el active.

Ejercicio 4

Sin tocar el HTML, agrega una palomita verde ([ok]) antes de cada elemento de una lista con clase "tareass".

SOLUCIÓN:

```
.tareass li {  
  list-style: none;  
}  
  
.tareass li::before {  
  content: "[ok] ";  
  color: green;  
  font-weight: bold;  
  margin-right: 8px;  
}
```

Primero quitamos las viñetas originales con `list-style: none`, luego insertamos la palomita con `::before`. El HTML no se tocó para nada.

Ejercicio 5

Explica la diferencia entre estos dos selectores y qué selecciona cada uno:

```
p:nth-child(2) { color: red; }  
p:nth-of-type(2) { color: blue; }
```

Dado este HTML:

```
<article>  
  <h2>Título</h2>  
  <p>Primero</p>  
  <p>Segundo</p>  
</article>
```

SOLUCIÓN: `p:nth-child(2)` selecciona los párrafos que sean el segundo hijo de su padre. Aquí el segundo hijo es el primer `<p>` ("Primero"), así que "Primero" será rojo. `p:nth-of-type(2)` selecciona el segundo párrafo entre los párrafos, sin importar su posición general: "Segundo" será azul. Resultado: "Primero" en rojo, "Segundo" en azul. Esta distinción es una de las trampas clásicas de los exámenes de CSS.

Ejercicio 6

Crea un campo de búsqueda que al recibir foco muestre un borde azul de 2px con un resplandor suave, y cuyo placeholder sea gris en itálica.

SOLUCIÓN:

```
.buscador {  
  padding: 12px 15px;  
  border: 1px solid #ccc;  
  border-radius: 25px;  
  font-size: 15px;  
  width: 250px;  
}  
  
.buscador:focus {  
  outline: none;  
  border-color: #4a90d9;  
  box-shadow: 0 0 8px rgba(74, 144, 217, 0.4);  
}  
  
.buscador::placeholder {  
  color: #999;  
  font-style: italic;  
}
```

El resplandor con `box-shadow` reemplaza al `outline` por defecto manteniendo la accesibilidad: el usuario de teclado sigue viendo claramente dónde está el foco.

Resumen de la unidad

En esta unidad llevaste `grid` al nivel avanzado con `grid-template-areas` (dibujar layouts con nombres de áreas), `minmax()` (rangos de tamaño), `repeat(auto-fit, minmax(...))` (rejillas responsive automáticas sin media queries) y el posicionamiento manual con `grid-column` / `grid-row`. Dominaste las pseudoclases de interacción (`:hover`, `:active`, `:focus`, `:visited`, `:link`), las de posición (`:first-child`, `:last-child`, `:nth-child` con números, `odd/even` y fórmulas, `:nth-of-type`, `:not`, `:checked`, `:disabled`) y la trampa clásica entre `nth-child` y `nth-of-type`. Y aprendiste los pseudo-elementos: `::before` y `::after` (que requieren `content`), `::first-letter`, `::first-line`, `::selection`, `::placeholder` y `::marker`, con la regla de oro: decoraciones en CSS, no `divs` vacíos en el HTML.

Puntos clave: dibuja layouts complejos con áreas nombradas; usa `auto-fit` + `minmax` para galerías responsive; da feedback visual en `hover/active/focus`; nunca elimines el foco sin reemplazo; y recuerda que `::before` / `::after` no existen sin `content`.

UNIDAD 7: MOVIMIENTO - TRANSICIONES, ANIMACIONES Y TRANSFORMACIONES

Introducción a la unidad

Una página estática informa; una página con movimiento bien usado enamora. No hablamos de animaciones exageradas que marean, sino de esos detalles que hacen que un sitio se sienta vivo y pulido: el botón que cambia de color con suavidad en vez de golpe, la tarjeta que se eleva ligeramente al pasar el mouse, el menú que se desliza en vez de aparecer de golpe, el indicador de carga que gira mientras esperas. Estos micro-movimientos son la diferencia entre un sitio que se siente amateur y uno que se siente profesional, y CSS te permite crearlos sin una sola línea de JavaScript.

En esta unidad aprenderás las tres herramientas de movimiento en CSS: las transiciones (cambios suaves entre dos estados), las animaciones con keyframes (secuencias complejas que se ejecutan solas) y las transformaciones (mover, rotar, escalar e inclinar elementos). Al terminar podrás darle vida a cualquier interfaz con efectos elegantes y profesionales.

Explicación detallada

Transiciones: la suavidad entre dos estados

Una transición le dice al navegador: "cuando esta propiedad cambie de valor, no lo hagas de golpe; hazlo gradualmente durante X segundos". Es el efecto más simple y más usado del CSS animado.

```
.boton {  
  background-color: #1a1a2e;  
  color: white;  
  padding: 12px 30px;  
  transition: background-color 0.3s;  
}  
  
.boton:hover {  
  background-color: #e94560;  
}
```

Sin la transición, el botón pasaría de azul a rojo instantáneamente al pasar el mouse: un cambio brusco. Con `transition: background-color 0.3s`, el cambio se difumina suavemente durante 0.3 segundos. Es una diferencia sutil que el usuario percibe como "calidad" sin saber por qué.

La sintaxis completa de `transition` acepta cuatro valores: propiedad, duración, función de tiempo y retardo:

```
transition: background-color 0.3s ease-in-out 0.1s;
```

- Propiedad: qué propiedad se anima (`background-color`, `all` para todas, aunque animar `all` puede ser costoso en rendimiento si cambian muchas cosas).
- Duración: cuánto tarda (en segundos `s` o milisegundos `ms`). Los valores típicos van de 0.2s a 0.5s para interacciones.
- Función de tiempo (timing function): cómo se distribuye la velocidad durante la animación. `ease` (empieza lento, acelera, termina lento; el valor por defecto y el más natural), `linear` (velocidad constante, para rotaciones continuas), `ease-in` (empieza lento y acelera), `ease-out` (empieza rápido y frena, ideal para elementos que aparecen) y `ease-in-out` (lento al inicio y al final).
- Retardo (delay): cuánto espera antes de empezar. Útil para efectos en cascada donde varios elementos se animan uno tras otro.

Puedes animar múltiples propiedades separándolas con comas:

```
.tarjeta {  
  transition: transform 0.3s ease, box-shadow 0.3s ease;  
}  
  
.tarjeta:hover {  
  transform: translateY(-8px);  
  box-shadow: 0 12px 24px rgba(0, 0, 0, 0.2);  
}
```

La tarjeta se eleva 8px y su sombra crece al pasar el mouse, todo suavizado. Este es el efecto "levitar" que ves en las tarjetas de productos de las tiendas modernas.

Analogía: una transición es como un dimmer de luz. Sin dimmer, el interruptor prende o apaga de golpe. Con dimmer, la luz sube o baja gradualmente. La transición es el dimmer de tus cambios de estilo.

Limitación importante: las transiciones solo funcionan entre dos estados definidos (normal y hover, por ejemplo) y solo cuando algo dispara el cambio. No pueden ejecutarse solas en bucle ni tener pasos intermedios. Para eso están las animaciones.

Animaciones con @keyframes: secuencias que se ejecutan solas

Si las transiciones son "de A a B suavemente", las animaciones son "haz esta coreografía completa, con los pasos que yo defina, las veces que yo diga". Se definen en dos partes: los keyframes (la coreografía) y la propiedad `animation` (la orden de ejecutarla).

```
@keyframes flotar {
  0% {
    transform: translateY(0);
  }
  50% {
    transform: translateY(-15px);
  }
  100% {
    transform: translateY(0);
  }
}

.globo {
  animation: flotar 3s ease-in-out infinite;
}
```

`@keyframes flotar` define los pasos: empieza en su lugar (0%), sube 15px a la mitad (50%) y regresa al final (100%). La propiedad `animation` lo ejecuta: nombre (flotar), duración (3s), función de tiempo (ease-in-out) y repeticiones (infinite para siempre). El resultado: el elemento flota suavemente de arriba abajo eternamente, como un globo.

También puedes usar `from` y `to` en vez de porcentajes cuando solo hay dos pasos:

```
@keyframes aparecer {
  from {
    opacity: 0;
    transform: translateY(20px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}
```

La propiedad `animation` acepta hasta ocho valores, en este orden: nombre, duración, timing-function, delay, iteration-count (número de repeticiones o `infinite`), direction, fill-mode y play-state. Los más usados además de los básicos:

- `animation-direction`: `normal` (siempre hacia adelante), `reverse` (siempre hacia atrás), `alternate` (adelante y atrás alternadamente: ideal para efectos de vaivén sin escribir keyframes de regreso).
- `animation-fill-mode`: qué pasa antes y después de la animación. `forwards` hace que el elemento conserve el estado final al terminar (sin esto, al terminar vuelve de golpe al estado inicial). `backwards` aplica el estado inicial durante el retardo. `both` combina ambos.
- `animation-play-state`: `paused` pausa la animación. Muy usado para pausar una animación al pasar el mouse: `.elemento:hover {animation-play-state: paused;}`.

Ejemplo completo: un indicador de carga (spinner), el elemento animado más común de la web:

```

.spinner {
  width: 50px;
  height: 50px;
  border: 5px solid #f0f0f0;
  border-top: 5px solid #e94560;
  border-radius: 50%;
  animation: girar 1s linear infinite;
}

@keyframes girar {
  from { transform: rotate(0deg); }
  to { transform: rotate(360deg); }
}

```

Un círculo con solo el borde superior de color, girando linealmente para siempre. `linear` es importante aquí: con `ease`, el giro aceleraría y frenaría en cada vuelta y se vería raro.

Transformaciones: mover, rotar, escalar e inclinar

`transform` modifica la apariencia visual de un elemento (posición, tamaño, rotación) sin afectar el flujo del documento: los demás elementos no se enteran del cambio y no se recorren. Es como mover una calcomanía sobre una foto: la calcomanía cambia de lugar pero la foto sigue igual.

Las cuatro funciones básicas:

- `translate(x, y)`: mueve el elemento. `translate(50px, 20px)` lo mueve 50px a la derecha y 20px abajo. Existen `translateX()` y `translateY()` por separado. Es la forma preferida de mover elementos animados porque tiene el mejor rendimiento (el navegador la acelera con la GPU).
- `scale(x, y)`: cambia el tamaño. `scale(1.5)` agranda al 150%; `scale(0.5)` encoge a la mitad; `scale(2, 1)` duplica el ancho sin tocar el alto. Valores mayores a 1 agrandan, menores a 1 encogen, negativos voltean (efecto espejo).
- `rotate(ángulo)`: gira el elemento. `rotate(45deg)` lo gira 45 grados en sentido horario; `rotate(-45deg)` en sentido contrario. Los ángulos también pueden expresarse en vueltas (`0.5turn` = media vuelta = `180deg`).
- `skew(x, y)`: inclina el elemento, como empujar un rectángulo para convertirlo en paralelogramo. `skew(10deg, 0)` inclina horizontalmente. Se usa para efectos decorativos y cintas.

Se pueden combinar varias en una sola declaración, y el orden importa:

```
transform: translateX(50px) rotate(45deg) scale(1.2);
```

Esto primero mueve, luego rota, luego escala. Cambiar el orden da resultados distintos porque cada transformación se aplica sobre el resultado de la anterior.

`transform-origin` define el punto desde el cual se aplican las transformaciones: por defecto es el centro del elemento (50% 50%). Si quieres que una puerta gire desde su bisagra (el borde izquierdo), usas `transform-origin: left center`. Si quieres que un elemento crezca desde la esquina superior izquierda, `transform-origin: top left`.

Combinar transformaciones con transiciones es la receta de la mayoría de los efectos hover elegantes:

```
.zoom {  
  transition: transform 0.3s ease;  
}  
  
.zoom:hover {  
  transform: scale(1.1);  
}
```

La imagen crece un 10% suavemente al pasar el mouse. Simple, elegante, profesional. Y como `transform` no afecta el flujo, los elementos vecinos no se mueven ni un píxel.

Rendimiento y accesibilidad en animaciones

No todas las propiedades se animan igual de bien. Las más eficientes son `transform` y `opacity`, porque el navegador puede animarlas con la tarjeta gráfica sin recalcular el layout. Animar `width`, `height`, `margin` o `top / left` obliga al navegador a recalcular la posición de todo en cada frame, lo que en páginas complejas causa tirones. Regla de oro: para mover o escalar, usa `transform`; para aparecer/desaparecer, usa `opacity`; evita animar propiedades de layout.

Sobre accesibilidad: algunas personas sufren mareos o náuseas con las animaciones (trastornos vestibulares). CSS tiene una media query para respetar su preferencia del sistema:

```
@media (prefers-reduced-motion: reduce) {  
  * {  
    animation: none;  
    transition: none;  
  }  
}
```

Con esto, si el usuario configuró "reducir movimiento" en su sistema operativo, tu sitio desactiva las animaciones automáticamente. Es un detalle profesional que pocos implementan y que marca la diferencia.

Errores comunes de esta unidad

Error 1: la animación "salta" al terminar porque falta `animation-fill-mode: forwards`. Sin `fill-mode`, al terminar la animación el elemento regresa de golpe a su estado inicial.

Error 2: transiciones que no se ven porque la propiedad no cambia entre dos estados definidos. La transición necesita un "antes" y un "después": si solo defines el `hover` sin estado base con transición, o viceversa, no hay nada que suavizar.

Error 3: animar `width` / `height` / `top` / `left` y que la página se sienta lenta. Usa `transform` y `opacity` para animaciones fluidas.

Error 4: `transform` que "no funciona" en elementos `inline`. Las transformaciones no aplican a elementos puramente `inline`; cambia el `display` a `inline-block` o `block` primero.

Error 5: abusar de las animaciones infinitas. Un elemento flotando está bien; diez elementos parpadeando a distintas velocidades es una tortura visual. Menos es más.

Buenas prácticas

Mantén las transiciones de interacción entre 0.2s y 0.4s: más rápido se siente brusco, más lento se siente pesado. Usa `ease-out` para elementos que aparecen y `ease-in-out` para movimientos de vaivén. Prefiere `transform` y `opacity` por rendimiento. Implementa `prefers-reduced-motion` en proyectos serios. Y recuerda: la mejor animación es la que el usuario percibe como "esto se siente bien" sin notar conscientemente que hubo una animación.

Ejemplos paso a paso

Ejemplo 1: Botón con transición completa

```
<button class="btn-magico">Pasa el mouse</button>
```

```
.btn-magico {  
  background-color: #1a1a2e;  
  color: white;  
  border: none;  
  padding: 15px 40px;  
  font-size: 16px;  
  border-radius: 8px;  
  cursor: pointer;  
  transition: background-color 0.3s ease, transform 0.3s ease, box-shadow 0.3s ease;  
}  
  
.btn-magico:hover {  
  background-color: #e94560;  
  transform: translateY(-3px);  
  box-shadow: 0 8px 20px rgba(233, 69, 96, 0.4);  
}  
  
.btn-magico:active {  
  transform: translateY(0) scale(0.98);  
  box-shadow: 0 2px 8px rgba(233, 69, 96, 0.4);  
}
```

Tres estados con personalidad: en reposo es sobrio, al hover se eleva con sombra de color, al presionar se hunde ligeramente. Todo suavizado. Este botón se siente "caro".

Ejemplo 2: Tarjetas que levitan

```

<div class="catalogo">
  <div class="producto">
    <h3>Laptop</h3>
    <p>$15,999</p>
  </div>
  <div class="producto">
    <h3>Mouse</h3>
    <p>$499</p>
  </div>
  <div class="producto">
    <h3>Teclado</h3>
    <p>$899</p>
  </div>
</div>

```

```

.catalogo {
  display: flex;
  gap: 20px;
  padding: 40px;
  font-family: Arial, sans-serif;
}

.producto {
  flex: 1;
  background: white;
  border: 1px solid #eee;
  border-radius: 12px;
  padding: 30px;
  text-align: center;
  transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.producto:hover {
  transform: translateY(-10px);
  box-shadow: 0 15px 30px rgba(0, 0, 0, 0.15);
}

```

El efecto levitar completo: la tarjeta sube 10px y proyecta una sombra más grande y difusa, como si se despegara de la página. Es el patrón de las tiendas en línea modernas.

Ejemplo 3: Spinner de carga y puntos suspensivos animados

```
<div class="spinner"></div>
<div class="cargando">Cargando<span class="puntos"></span></div>
```

```
.spinner {
  width: 45px;
  height: 45px;
  border: 5px solid #eeeeee;
  border-top-color: #e94560;
  border-radius: 50%;
  animation: girar 0.8s linear infinite;
}

@keyframes girar {
  to { transform: rotate(360deg); }
}

.cargando {
  font-family: Arial, sans-serif;
  font-size: 18px;
  color: #666;
  margin-top: 20px;
}

.puntos::after {
  content: "";
  animation: puntos 1.5s steps(4) infinite;
}

@keyframes puntos {
  0% { content: ""; }
  25% { content: "."; }
  50% { content: ".."; }
  75% { content: "..."; }
}
```

Dos indicadores de carga clásicos: el spinner giratorio y los puntos suspensivos que aparecen uno por uno. El truco de animar `content` con `steps()` es una joya poco conocida de CSS puro.

Ejemplo 4: Entrada escalonada de elementos

```
<ul class="lista-animada">
  <li>Primer elemento</li>
  <li>Segundo elemento</li>
  <li>Tercer elemento</li>
  <li>Cuarto elemento</li>
</ul>
```

```
.lista-animada {
  list-style: none;
  font-family: Arial, sans-serif;
}

.lista-animada li {
  background: #f0f4ff;
  margin: 10px 0;
  padding: 15px 20px;
  border-radius: 8px;
  opacity: 0;
  animation: entrar 0.5s ease forwards;
}

.lista-animada li:nth-child(1) { animation-delay: 0.1s; }
.lista-animada li:nth-child(2) { animation-delay: 0.2s; }
.lista-animada li:nth-child(3) { animation-delay: 0.3s; }
.lista-animada li:nth-child(4) { animation-delay: 0.4s; }

@keyframes entrar {
  from {
    opacity: 0;
    transform: translateX(-30px);
  }
  to {
    opacity: 1;
    transform: translateX(0);
  }
}
```

Cada elemento entra deslizándose desde la izquierda con un retardo progresivo, creando un efecto de cascada. El `forwards` es esencial: sin él, los elementos volverían a `opacity: 0` al terminar y desaparecerían. El `nth-child` controla el retardo de cada uno.

Ejemplo 5: Insignia giratoria y corazón latiendo

```
<span class="insignia-nueva">NUEVO</span>
<span class="corazon"><3</span>
```

```
.insignia-nueva {
  display: inline-block;
  background-color: #e94560;
  color: white;
  font-family: Arial, sans-serif;
  font-weight: bold;
  font-size: 12px;
  padding: 6px 14px;
  border-radius: 4px;
  transform: rotate(-8deg);
  box-shadow: 2px 2px 6px rgba(0, 0, 0, 0.2);
}

.corazon {
  display: inline-block;
  font-size: 40px;
  animation: latir 1.2s ease-in-out infinite;
  transform-origin: center;
}

@keyframes latir {
  0%, 100% { transform: scale(1); }
  25% { transform: scale(1.25); }
  50% { transform: scale(1); }
  75% { transform: scale(1.15); }
}
```

La insignia inclinada con `rotate(-8deg)` parece una etiqueta pegada a mano, y el corazón late con un ritmo de doble pulso (como un corazón real: lub-dub). El `transform-origin: center` asegura que el escalado parta del centro.

Práctica

Ejercicio 1

Haz que un enlace cambie de color de azul a rojo suavemente en 0.4 segundos al pasar el mouse.

SOLUCIÓN:

```
a {
  color: blue;
  transition: color 0.4s ease;
}

a:hover {
  color: red;
}
```

La transición va en el estado base (no en el hover) para que funcione en ambas direcciones: al entrar y al salir el mouse.

Ejercicio 2

Crea una caja que al pasar el mouse rote 15 grados y crezca un 10%, con transición suave de 0.3s.

SOLUCIÓN:

```
.caja {
  width: 150px;
  height: 150px;
  background-color: #4a90d9;
  transition: transform 0.3s ease;
}

.caja:hover {
  transform: rotate(15deg) scale(1.1);
}
```

El orden en el transform (rotar luego escalar) se aplica de izquierda a derecha sobre el resultado anterior.

Ejercicio 3

Escribe los keyframes para una animación llamada "rebotar" que haga que un elemento baje 30px y regrese, y aplícala en bucle infinito de 1 segundo.

SOLUCIÓN:

```
@keyframes rebotar {
  0%, 100% { transform: translateY(0); }
  50% { transform: translateY(30px); }
}

.pelota {
  animation: rebotar 1s ease-in-out infinite;
}
```

Agrupar 0%, 100% con el mismo valor crea el ciclo perfecto sin saltos. El ease-in-out hace que el movimiento acelere al bajar y frene al subir, como una pelota real.

Ejercicio 4

Explica por qué esta animación desaparece al terminar y cómo arreglarlo:

```
.aviso {
  animation: aparecer 1s ease;
}

@keyframes aparecer {
  from { opacity: 0; }
  to { opacity: 1; }
}
```

SOLUCIÓN: Al terminar la animación, el elemento vuelve a su estado inicial definido en el CSS... pero aquí el estado inicial no define opacity, así que vuelve al valor por defecto (1) - espera, en este caso no desaparecería. El problema real: si el CSS base tuviera opacity: 0 (para ocultar el elemento antes de animar), al terminar la animación regresaría a 0 y desaparecería de golpe. La solución es animation-fill-mode: forwards para conservar el estado final:

```
.aviso {  
  opacity: 0;  
  animation: aparecer 1s ease forwards;  
}
```

Ahora el elemento empieza invisible, aparece suavemente y se queda visible.

Ejercicio 5

Crea un botón que al pasar el mouse se desplace 5px hacia arriba y su sombra crezca, y al presionarlo regrese a su lugar. Todo con transiciones de 0.25s.

SOLUCIÓN:

```
.btn-subir {  
  padding: 14px 35px;  
  background-color: #2a9d8f;  
  color: white;  
  border: none;  
  border-radius: 8px;  
  cursor: pointer;  
  transition: transform 0.25s ease, box-shadow 0.25s ease;  
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.15);  
}  
  
.btn-subir:hover {  
  transform: translateY(-5px);  
  box-shadow: 0 10px 20px rgba(0, 0, 0, 0.2);  
}  
  
.btn-subir:active {  
  transform: translateY(0);  
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.15);  
}
```

Definir la sombra base en el estado normal es importante para que la transición tenga un punto de partida.

Ejercicio 6

Haz que tres círculos parpadeen en secuencia (uno tras otro) infinitamente, como un indicador de "escribiendo...".

SOLUCIÓN:

```
.escribiendo span {
  display: inline-block;
  width: 12px;
  height: 12px;
  background-color: #999;
  border-radius: 50%;
  margin: 0 3px;
  animation: parpadear 1.2s infinite;
}

.escribiendo span:nth-child(2) { animation-delay: 0.2s; }
.escribiendo span:nth-child(3) { animation-delay: 0.4s; }

@keyframes parpadear {
  0%, 100% { opacity: 0.3; }
  50% { opacity: 1; }
}
```

Los retardos escalonados (0s, 0.2s, 0.4s) crean la ola de parpadeo. Se anima `opacity` (eficiente) en vez de encender y apagar con `display` (que no se puede animar).

Resumen de la unidad

En esta unidad aprendiste las tres herramientas de movimiento en CSS. Las transiciones suavizan cambios entre dos estados con `transition: propiedad duración timing delay`, usando funciones como `ease`, `linear` y `ease-in-out`. Las animaciones con `@keyframes` definen coreografías completas con pasos en porcentajes o `from / to`, ejecutadas con `animation: nombre duración timing delay repeticiones dirección relleno`, incluyendo `infinite`, `alternate` y el crucial `fill-mode: forwards`. Las transformaciones modifican lo visual sin afectar el flujo: `translate` (mover, la más eficiente), `scale` (tamaño), `rotate` (giro) y `skew` (inclinación), combinables y controlables con `transform-origin`. También viste las reglas de rendimiento (`anima transform` y `opacity`, no propiedades de layout) y accesibilidad (`prefers-reduced-motion`).

Puntos clave: la transición va en el estado base; `forwards` evita el salto al terminar; combina `transform + transition` para los hovers elegantes; y la mejor animación es la que se siente sin notarse.

UNIDAD 8: SOMBRAS, LISTAS, TABLAS Y FORMULARIOS

Introducción a la unidad

Hasta ahora has construido la estructura y el movimiento. Esta unidad se trata del acabado fino: esos detalles que separan un diseño plano de uno con profundidad y un formulario genérico de uno que provoca llenarlo. Aprenderás a dar profundidad con sombras en cajas y texto, a domar las listas y tablas (dos elementos HTML con estilos por defecto tercos que hay que resetear antes de vestir), y a estilizar formularios completos: campos, botones, estados de foco y validación.

Los formularios merecen atención especial porque son el punto donde tu sitio conversa con el usuario: registros, inicios de sesión, contactos, compras. Un formulario bien diseñado aumenta las conversiones; uno descuidado las destruye. Al terminar esta unidad podrás crear tablas legibles, listas elegantes y formularios profesionales de principio a fin.

Explicación detallada

Sombras: profundidad instantánea

La sombra es el truco más rápido para pasar del diseño plano al diseño con profundidad. CSS tiene dos propiedades: `box-shadow` para cajas y `text-shadow` para texto.

`box-shadow` acepta hasta seis valores en este orden: desplazamiento-x, desplazamiento-y, desenfoque (blur), expansión (spread), color, y la palabra `inset` para sombras internas:

```
.tarjeta {  
  box-shadow: 5px 5px 10px gray;  
}
```

Esto proyecta una sombra gris 5px a la derecha y 5px abajo, con 10px de desenfoque. Desglosemos cada valor:

- Desplazamiento X: qué tan a la derecha (positivo) o izquierda (negativo) cae la sombra.
- Desplazamiento Y: qué tan abajo (positivo) o arriba (negativo) cae.

- Blur (desenfoque): qué tan difusa es la sombra. 0 es una sombra nítida con bordes duros; valores grandes crean sombras suaves y realistas.
- Spread (expansión, opcional): cuánto crece o encoge la sombra respecto al tamaño de la caja. Positivo la agranda, negativo la encoge.
- Color: normalmente negro o gris con transparencia (`rgba(0, 0, 0, 0.2)`). La transparencia es clave: una sombra sólida se ve falsa; una semitransparente se ve natural.
- `inset`: convierte la sombra en interna, como si la luz viniera desde adentro del elemento (efecto de hundido o de campo presionado).

La receta de sombra elegante que usan los sistemas de diseño modernos:

```
box-shadow: 0 4px 12px rgba(0, 0, 0, 0.12);
```

Desplazamiento X en 0 (la sombra cae recta hacia abajo, como la luz del mediodía), un poco de Y, blur generoso y negro al 12% de opacidad. Para el efecto "levitar" al hover, aumenta Y y el blur:

```
.tarjeta {
  box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);
  transition: box-shadow 0.3s ease;
}

.tarjeta:hover {
  box-shadow: 0 12px 28px rgba(0, 0, 0, 0.18);
}
```

Puedes apilar múltiples sombras separándolas con comas, lo que permite efectos sofisticados como anillos concéntricos o sombras en varias direcciones:

```
box-shadow: 0 2px 4px rgba(0,0,0,0.1), 0 8px 16px rgba(0,0,0,0.1);
```

`text-shadow` funciona igual pero sin `spread` ni `inset`: desplazamiento-x, desplazamiento-y, blur y color:

```
.titulo {
  text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
}
```

Sus dos usos principales: legibilidad de texto sobre imágenes (una sombra oscura semitransparente hace que el texto blanco se lea sobre cualquier foto) y efectos decorativos (texto con relieve, neón, etc.). Un efecto neón, por ejemplo, apila varias sombras del mismo color:

```
.neon {  
  color: #fff;  
  text-shadow: 0 0 5px #0ff, 0 0 10px #0ff, 0 0 20px #0ff;  
}
```

Listas: del HTML aburrido al menú elegante

Las listas HTML (, ,) vienen con estilos por defecto del navegador: viñetas o números, y un padding izquierdo generoso. El primer paso para estilizarlas casi siempre es resetear:

```
.lista-limpia {  
  list-style: none;  
  padding: 0;  
  margin: 0;  
}
```

`list-style: none` quita las viñetas o números. Pero ojo: solo quita el marcador, el `padding-left` por defecto (unos 40px) sigue ahí, por eso hay que resetearlo también. Este trío de líneas aparece al inicio de prácticamente cada menú de navegación del mundo.

La propiedad completa es `list-style`, abreviatura de tres valores: tipo, posición e imagen:

```
list-style: square inside url("vineta.png");
```

- `list-style-type`: el tipo de marcador. Para no ordenadas: `disc` (punto sólido, el defecto), `circle` (círculo hueco), `square` (cuadrado). Para ordenadas: `decimal` (1, 2, 3), `upper-roman` (I, II, III), `lower-alpha` (a, b, c), entre otros.
- `list-style-position`: `outside` (el marcador queda fuera del flujo del texto, el defecto; las líneas largas se alinean bajo el texto, no bajo la viñeta) o `inside` (el marcador queda dentro, como parte del texto).

- `list-style-image`: reemplaza el marcador con una imagen: `list-style-image: url("check.png")`. Útil pero limitado (no puedes controlar el tamaño fácilmente); muchos profesionales prefieren el truco de `:before` con `content` que viste en la Unidad 6.

Para menús horizontales, el patrón es: resetear la lista, poner los `li` en `inline-block` o usar `flex` en el `ul`:

```
.menu {
  list-style: none;
  padding: 0;
  margin: 0;
  display: flex;
  gap: 5px;
  background-color: #1a1a2e;
}

.menu a {
  display: block;
  color: white;
  text-decoration: none;
  padding: 14px 22px;
}

.menu a:hover {
  background-color: #e94560;
}
```

Y para listas de contenido elegante (precios, características, pasos), el patrón es: resetear, dar padding y bordes a cada `li`, y usar `:nth-child` o `:before` para detalles.

Tablas: orden visual para datos

Las tablas HTML también traen estilos por defecto incómodos: celdas con bordes separados por espacios (el famoso "doble borde"). El primer paso es siempre:

```
table {
  border-collapse: collapse;
  width: 100%;
}
```

`border-collapse: collapse` fusiona los bordes de las celdas adyacentes en uno solo. Sin esto, cada celda dibuja su propio borde y se ven líneas dobles con huecos. Es la línea más importante del CSS de tablas.

A partir de ahí, la receta de tabla profesional:

```
.tabla {
  border-collapse: collapse;
  width: 100%;
  font-family: Arial, sans-serif;
  font-size: 15px;
}

.tabla th {
  background-color: #1a1a2e;
  color: white;
  text-align: left;
  padding: 14px 16px;
}

.tabla td {
  padding: 12px 16px;
  border-bottom: 1px solid #e0e0e0;
}

.tabla tbody tr:nth-child(even) {
  background-color: #f8f9fa;
}

.tabla tbody tr:hover {
  background-color: #e3f2fd;
}
```

Encabezado oscuro con texto blanco, celdas con padding generoso separadas por líneas sutiles (solo borde inferior, más elegante que la cuadrícula completa), filas alternadas para legibilidad y resaltado al hover para seguir la fila con la vista. Esta es la tabla que verás en paneles de administración profesionales.

Detalles finos: `text-align: left` en los encabezados (el centrado en tablas dificulta el escaneo visual), `white-space: nowrap` en columnas que no deben partirse (como fechas o precios), y para tablas largas, el `position: sticky` en el `thead` que viste en la Unidad 4.

Formularios: donde el diseño se vuelve conversación

Estilizar formularios es un arte porque los controles (input, select, textarea, button) traen estilos nativos del sistema operativo que varían entre navegadores. La base es normalizarlos:

```
input, select, textarea {  
  font-family: inherit;  
  font-size: 16px;  
  padding: 12px 15px;  
  border: 1px solid #ccc;  
  border-radius: 8px;  
  width: 100%;  
  box-sizing: border-box;  
}
```

font-family: inherit es importante: por defecto, los campos usan la fuente del sistema, no la de tu página. El font-size: 16px tiene un motivo práctico: en iPhone, los campos con fuente menor a 16px provocan que el navegador haga zoom automático al enfocarlos, una experiencia molesta. El width: 100% con box-sizing: border-box hace que los campos llenen su contenedor sin desbordarse por el padding y el borde.

Los estados son la mitad del diseño de formularios:

```
.campo:focus {  
  outline: none;  
  border-color: #4a90d9;  
  box-shadow: 0 0 0 3px rgba(74, 144, 217, 0.25);  
}
```

El campo enfocado se ilumina con borde de color y un anillo suave: el usuario siempre sabe dónde está escribiendo. (Recuerda: si quitas el outline, reemplázalo, como aquí.)

HTML5 trae validación nativa con pseudoclases que CSS puede estilizar::valid e::invalid (según si el valor cumple las reglas como required, type="email", minlength), y :required/:optional:

```
.campo:invalid {
  border-color: #e94560;
}

.campo:valid {
  border-color: #2a9d8f;
}
```

Con esto, el campo se pone rojo si el correo está mal escrito y verde cuando es válido, sin JavaScript. Un detalle: `:invalid` aplica desde que la página carga (el campo vacío e inválido se vería rojo inmediatamente), así que muchos profesionales combinan con `:not(:placeholder-shown)` o aplican estos estilos solo después de la interacción. Es un refinamiento avanzado que vale la pena conocer.

Los botones de envío merecen su propio diseño: color sólido de acento, padding generoso, bordes redondeados, y estados hover/active como viste en la Unidad 7. Y no olvides las etiquetas (`<label>`): deben estar visualmente asociadas a sus campos (arriba o al lado, nunca lejos), con `display: block` y margen inferior para formularios verticales.

El checkbox y el radio son los controles más difíciles de estilizar porque cada navegador los dibuja a su manera. La técnica profesional es ocultarlos (`opacity: 0` o `appearance: none`) y dibujar uno personalizado con `:before` en el label, usando `:checked` para el estado marcado. Es un patrón avanzado; lo importante es que sepas que existe y que los controles nativos sin estilizar se ven inconsistentes entre navegadores.

Por último, el layout del formulario: usa flexbox en columna con `gap` (como viste en la Unidad 5) para apilar campos con espaciado perfecto, agrupa campos relacionados (nombre y apellido lado a lado con flex en fila), y pon los mensajes de error justo debajo del campo correspondiente en color rojo con un tamaño de letra ligeramente menor.

Errores comunes de esta unidad

Error 1: sombras sólidas negras que se ven falsas. Usa siempre transparencia: `rgba(0, 0, 0, 0.15)` en vez de `black` o `gray` sólido.

Error 2: quitar las viñetas con `list-style: none` pero olvidar resetear el `padding-left`, dejando un hueco fantasma a la izquierda de la lista.

Error 3: tablas con doble borde por olvidar `border-collapse: collapse`.

Error 4: campos de formulario con fuente menor a 16px que provocan zoom automático en iPhone.

Error 5: quitar el `outline` del `:focus` sin poner un estilo alternativo, dejando a los usuarios de teclado perdidos.

Error 6: `width: 100%` en campos sin `box-sizing: border-box`, haciendo que el padding y el borde desborden el contenedor.

Buenas prácticas

Para sombras: X en 0, Y moderado, blur generoso, negro con 10-20% de opacidad. Para listas: resetea siempre (`list-style`, `padding`, `margin`) antes de vestir. Para tablas: `border-collapse` primero, encabezado contrastante, cebreado y hover. Para formularios: 16px mínimo en campos, foco visible y bonito, etiquetas asociadas, mensajes de error junto al campo, y layout con `flex + gap`. Y en todo: la consistencia (mismos radios de borde, mismos espaciados, misma paleta) es lo que hace que un conjunto de componentes se sienta como un sistema de diseño y no como piezas sueltas.

Ejemplos paso a paso

Ejemplo 1: Sistema de sombras por niveles

```
<div class="demo-sombras">
  <div class="nivel n1">Nivel 1</div>
  <div class="nivel n2">Nivel 2</div>
  <div class="nivel n3">Nivel 3</div>
  <div class="nivel n4">Nivel 4</div>
</div>
```

```

.demo-sombras {
  display: flex;
  gap: 25px;
  padding: 40px;
  background-color: #f5f5f5;
  font-family: Arial, sans-serif;
}

.nivel {
  width: 140px;
  height: 100px;
  background: white;
  border-radius: 10px;
  display: flex;
  justify-content: center;
  align-items: center;
}

.n1 { box-shadow: 0 1px 3px rgba(0, 0, 0, 0.12); }
.n2 { box-shadow: 0 4px 12px rgba(0, 0, 0, 0.14); }
.n3 { box-shadow: 0 10px 25px rgba(0, 0, 0, 0.16); }
.n4 { box-shadow: 0 20px 45px rgba(0, 0, 0, 0.2); }

```

Cuatro niveles de elevación: del sutil (tarjetas en reposo) al dramático (modales, elementos flotantes). Los sistemas de diseño profesionales definen estos niveles como variables y los usan consistentemente: a mayor importancia o cercanía, mayor sombra.

Ejemplo 2: Menú lateral elegante

```

<ul class="menu-lateral">
  <li><a href="#" class="activo">Dashboard</a></li>
  <li><a href="#">Productos</a></li>
  <li><a href="#">Pedidos</a></li>
  <li><a href="#">Clientes</a></li>
  <li><a href="#">Configuración</a></li>
</ul>

```

```

.menu-lateral {
  list-style: none;
  padding: 0;
  margin: 0;
  width: 240px;
  background-color: #1a1a2e;
  border-radius: 12px;
  padding: 15px;
  font-family: Arial, sans-serif;
}

.menu-lateral a {
  display: block;
  color: #b8b8c8;
  text-decoration: none;
  padding: 13px 18px;
  border-radius: 8px;
  transition: background-color 0.25s, color 0.25s;
}

.menu-lateral a:hover {
  background-color: rgba(255, 255, 255, 0.08);
  color: white;
}

.menu-lateral a.activo {
  background-color: #e94560;
  color: white;
  font-weight: bold;
}

```

El patrón completo: reset de lista, enlaces como bloques clicables, hover sutil y estado activo destacado. Es el menú lateral de un panel de administración real.

Ejemplo 3: Tabla de datos profesional

```
<table class="datos">
  <thead>
    <tr>
      <th>Producto</th>
      <th>Categoría</th>
      <th>Precio</th>
      <th>Stock</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Laptop Pro 15</td>
      <td>Computadoras</td>
      <td>$24,999</td>
      <td>12</td>
    </tr>
    <tr>
      <td>Mouse inalámbrico</td>
      <td>Accesorios</td>
      <td>$599</td>
      <td>85</td>
    </tr>
    <tr>
      <td>Teclado mecánico</td>
      <td>Accesorios</td>
      <td>$1,899</td>
      <td>34</td>
    </tr>
    <tr>
      <td>Monitor 27"</td>
      <td>Monitores</td>
      <td>$6,499</td>
      <td>20</td>
    </tr>
  </tbody>
</table>
```

```

.datos {
  border-collapse: collapse;
  width: 100%;
  max-width: 800px;
  font-family: Arial, sans-serif;
  font-size: 15px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.08);
  border-radius: 10px;
  overflow: hidden;
}

.datos th {
  background-color: #1a1a2e;
  color: white;
  text-align: left;
  padding: 15px 18px;
  font-weight: 600;
}

.datos td {
  padding: 13px 18px;
  border-bottom: 1px solid #eeeeee;
}

.datos tbody tr:nth-child(even) {
  background-color: #f8f9fa;
}

.datos tbody tr:hover {
  background-color: #e8f0fe;
}

.datos tbody tr:last-child td {
  border-bottom: none;
}

```

Todos los patrones juntos: collapse, encabezado oscuro, cebreado, hover, y el truco de border-radius + overflow: hidden en la tabla para redondear las esquinas (los bordes de las celdas no respetan el radius del table sin el overflow).

Ejemplo 4: Formulario de registro completo

```
<form class="registro">
  <h2>Crear cuenta</h2>
  <p class="sub">Únete gratis a la academia</p>

  <label for="nombre">Nombre completo</label>
  <input type="text" id="nombre" class="campo" placeholder="Ej. María González"
required>

  <label for="correo">Correo electrónico</label>
  <input type="email" id="correo" class="campo" placeholder="tucorreo@ejemplo.com"
required>

  <label for="pass">Contraseña</label>
  <input type="password" id="pass" class="campo" placeholder="Mínimo 8 caracteres"
minlength="8" required>

  <button type="submit" class="btn-enviar">Registrarme</button>
  <p class="login">¿Ya tienes cuenta? <a href="#">Inicia sesión</a></p>
</form>
```

```
.registro {
  max-width: 420px;
  margin: 40px auto;
  padding: 40px;
  background: white;
  border-radius: 16px;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.12);
  font-family: Arial, sans-serif;
}

.registro h2 {
  margin: 0;
  color: #1a1a2e;
  text-align: center;
}

.registro .sub {
  text-align: center;
  color: #888;
  margin: 8px 0 25px 0;
}
```

```
.registro label {
  display: block;
  font-weight: bold;
  font-size: 14px;
  color: #333;
  margin: 18px 0 8px 0;
}

.campo {
  width: 100%;
  box-sizing: border-box;
  font-family: inherit;
  font-size: 16px;
  padding: 13px 15px;
  border: 1.5px solid #ddd;
  border-radius: 10px;
  transition: border-color 0.25s, box-shadow 0.25s;
}

.campo:focus {
  outline: none;
  border-color: #4a90d9;
  box-shadow: 0 0 0 3px rgba(74, 144, 217, 0.2);
}

.campo::placeholder {
  color: #aaa;
}

.btn-enviar {
  width: 100%;
  margin-top: 25px;
  padding: 15px;
  background: linear-gradient(to right, #e94560, #ff7e5f);
  color: white;
  border: none;
  border-radius: 10px;
  font-size: 17px;
  font-weight: bold;
  cursor: pointer;
  transition: transform 0.2s, box-shadow 0.2s;
}

.btn-enviar:hover {
```

```
transform: translateY(-2px);
box-shadow: 0 8px 20px rgba(233, 69, 96, 0.35);
}

.login {
  text-align: center;
  font-size: 14px;
  color: #888;
  margin-top: 20px;
}
```

Un formulario de registro de nivel producción: tarjeta con sombra, etiquetas claras, campos con foco bonito, botón con degradado y efecto hover. Cada decisión de esta unidad aplicada en un solo componente.

Ejemplo 5: Lista de características con palomitas

```
<ul class="features">
  <li>Acceso de por vida a todos los cursos</li>
  <li>Certificado al completar cada curso</li>
  <li>Comunidad privada de estudiantes</li>
  <li>Actualizaciones gratis para siempre</li>
</ul>
```

```

.features {
  list-style: none;
  padding: 0;
  font-family: Arial, sans-serif;
  font-size: 17px;
  max-width: 450px;
}

.features li {
  padding: 12px 0 12px 40px;
  position: relative;
  border-bottom: 1px solid #f0f0f0;
}

.features li::before {
  content: "[ok]";
  position: absolute;
  left: 0;
  top: 12px;
  width: 26px;
  height: 26px;
  background-color: #2a9d8f;
  color: white;
  border-radius: 50%;
  display: flex;
  justify-content: center;
  align-items: center;
  font-size: 14px;
  font-weight: bold;
}

```

Palomitas en círculos verdes con `::before` posicionado, sin imágenes y sin tocar el HTML. Es el patrón de las páginas de precios y landing pages.

Práctica

Ejercicio 1

Dale a una tarjeta la sombra elegante estándar y haz que al hover la sombra crezca suavemente.

SOLUCIÓN:

```
.tarjeta {  
  box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);  
  transition: box-shadow 0.3s ease;  
}  
  
.tarjeta:hover {  
  box-shadow: 0 12px 28px rgba(0, 0, 0, 0.18);  
}
```

La sombra base sutil y la de hover más grande y difusa crean la ilusión de que la tarjeta se acerca al usuario.

Ejercicio 2

Convierte esta lista en un menú horizontal sin viñetas, con fondo oscuro y enlaces blancos:

```
<ul class="nav">  
  <li><a href="#">A</a></li>  
  <li><a href="#">B</a></li>  
  <li><a href="#">C</a></li>  
</ul>
```

SOLUCIÓN:

```

.nav {
  list-style: none;
  padding: 0;
  margin: 0;
  display: flex;
  background-color: #222;
}

.nav a {
  display: block;
  color: white;
  text-decoration: none;
  padding: 14px 20px;
}

.nav a:hover {
  background-color: #444;
}

```

El trío de reseteo (list-style, padding, margin) más flex para la fila. Sin el reseteo del padding, el menú tendría un hueco a la izquierda.

Ejercicio 3

Explica qué está mal en esta tabla y corrígela:

```

<table class="t">
  <tr><td>A</td><td>B</td></tr>
  <tr><td>C</td><td>D</td></tr>
</table>

```

```

.t {
  width: 100%;
}

.t td {
  border: 1px solid black;
  padding: 10px;
}

```

SOLUCIÓN: Falta `border-collapse: collapse`, así que los bordes de celdas adyacentes se dibujan dobles con un espacio entre ellos. Corrección: agregar `border-collapse: collapse`; a. t. Con eso, los bordes se fusionan en líneas simples y limpias.

Ejercicio 4

Estiliza un campo de texto: ancho completo, 16px, padding 12px, borde gris redondeado, y al enfocarlo borde azul con anillo suave (sin usar el outline por defecto).

SOLUCIÓN:

```
.campo {
  width: 100%;
  box-sizing: border-box;
  font-size: 16px;
  font-family: inherit;
  padding: 12px;
  border: 1px solid #ccc;
  border-radius: 8px;
}

.campo:focus {
  outline: none;
  border-color: #4a90d9;
  box-shadow: 0 0 0 3px rgba(74, 144, 217, 0.25);
}
```

El anillo con `box-shadow` reemplaza al `outline` manteniendo la accesibilidad. El `box-sizing: border-box` evita el desbordamiento con `width 100%`.

Ejercicio 5

Haz que un título sobre una imagen de fondo sea legible usando `text-shadow`, suponiendo texto blanco.

SOLUCIÓN:

```
.hero h1 {  
  color: white;  
  text-shadow: 2px 2px 6px rgba(0, 0, 0, 0.7);  
}
```

La sombra negra al 70% crea un halo oscuro alrededor de cada letra que separa el texto del fondo, sea claro u oscuro. Para fotos muy claras, puedes agregar además un degradado oscuro semitransparente sobre la imagen.

Ejercicio 6

Crema una sombra interna (inset) para un campo de texto que simule estar "hundido", y una sombra múltiple para un botón con efecto 3D.

SOLUCIÓN:

```
.campo-hundido {  
  box-shadow: inset 0 2px 5px rgba(0, 0, 0, 0.15);  
  border: 1px solid #ddd;  
  padding: 12px;  
  border-radius: 8px;  
}  
  
.boton-3d {  
  background-color: #e94560;  
  color: white;  
  border: none;  
  padding: 14px 30px;  
  border-radius: 8px;  
  box-shadow: 0 4px 0 #b0304a, 0 6px 12px rgba(0, 0, 0, 0.2);  
}  
  
.boton-3d:active {  
  transform: translateY(4px);  
  box-shadow: 0 0 0 #b0304a, 0 2px 6px rgba(0, 0, 0, 0.2);  
}
```

El inset hunde el campo. El botón 3D usa dos sombras: una sólida de 4px abajo (el "grosor" del botón) y una difusa. Al presionarlo, el botón baja 4px y la sombra sólida desaparece: parece que se hunde de verdad.

Resumen de la unidad

En esta unidad aprendiste a dar profundidad con `box-shadow` (desplazamientos, blur, spread, color con transparencia, `inset`, sombras múltiples) y `text-shadow` (legibilidad sobre imágenes y efectos decorativos). Dominaste las listas: el reseteo obligatorio (`list-style: none + padding/margin 0`), los tipos de marcador, y los patrones de menú horizontal y lateral. Aprendiste las tablas profesionales: `border-collapse: collapse` como primer mandamiento, encabezados contrastantes, cebreado con `:nth-child(even)`, hover en filas y el truco de `radius + overflow`. Y construiste formularios completos: normalización de campos (16px, inherit, box-sizing), estados: focus con anillo visible, validación con `:valid / :invalid`, etiquetas asociadas, botones con presencia y layouts con `flex + gap`.

Puntos clave: sombras siempre con transparencia; resetea listas y colapsa tablas antes de vestir las; 16px mínimo en campos de formulario; nunca quites el foco sin reemplazo; y la consistencia convierte componentes sueltos en un sistema de diseño.

UNIDAD 9: DISEÑO RESPONSIVE, UNIDADES Y VARIABLES

Introducción a la unidad

Más de la mitad del tráfico web mundial viene de teléfonos celulares. Un sitio que se ve perfecto en la computadora pero se rompe en el celular está perdiendo a la mayoría de sus visitantes. El diseño responsive (adaptable) es la disciplina de hacer que una misma página se vea bien en cualquier pantalla: del celular pequeño al monitor gigante, pasando por la tablet. No se trata de hacer "una versión para celular", sino de construir con flexibilidad desde el inicio.

En esta unidad aprenderás las tres herramientas del responsive: las media queries (reglas que se activan según las características del dispositivo, principalmente el ancho de pantalla), las unidades de medida relativas (que permiten que los tamaños se adapten en vez de ser rígidos) y las variables CSS (que te permiten centralizar valores como colores y espaciados para mantener un diseño consistente y fácil de ajustar). Al terminar podrás tomar cualquier diseño de escritorio y adaptarlo elegantemente al celular.

Explicación detallada

Media queries: estilos condicionales

Una media query es un bloque de CSS que solo se aplica cuando se cumple una condición sobre el dispositivo, típicamente su ancho:

```
.contenedor {  
  width: 1000px;  
  margin: 0 auto;  
}  
  
@media (max-width: 768px) {  
  .contenedor {  
    width: 95%;  
    padding: 0 15px;  
  }  
}
```

Léelo así: "el contenedor mide 1000px normalmente; PERO si la pantalla mide 768px o menos, entonces mide el 95% con padding lateral". El navegador evalúa la condición en tiempo real: si giras el celular o redimensionas la ventana, los estilos se activan y desactivan solos.

La anatomía: @media es la regla, entre paréntesis va la condición (max-width: 768px significa "ancho máximo de 768px", es decir, pantallas de 768px o menos), y dentro de las llaves van las reglas normales que solo aplican cuando la condición se cumple. También existe min-width ("ancho mínimo", pantallas de ese tamaño o más), y se pueden combinar: @media (min-width: 768px) and (max-width: 1024px) para un rango específico, como tablets.

Hay dos filosofías para escribir responsive: desktop-first y mobile-first. En desktop-first (la tradicional), escribes primero los estilos para pantalla grande y luego agregas media queries con max-width para ir ajustando hacia abajo. En mobile-first (la recomendada hoy), escribes primero para celular -el caso más restringido- y luego agregas media queries con min-width para mejorar progresivamente en pantallas más grandes:

```

/* Mobile-first: base para celular */
.tarjetas {
  display: grid;
  grid-template-columns: 1fr;
  gap: 15px;
}

/* Tablet: 2 columnas */
@media (min-width: 600px) {
  .tarjetas {
    grid-template-columns: repeat(2, 1fr);
  }
}

/* Escritorio: 4 columnas */
@media (min-width: 1024px) {
  .tarjetas {
    grid-template-columns: repeat(4, 1fr);
  }
}

```

Mobile-first tiene ventajas: el CSS base es más simple, los celulares (la mayoría del tráfico) cargan solo lo necesario, y te obliga a priorizar el contenido esencial. Los puntos de corte típicos (breakpoints) son: 600px (celulares grandes/tablets pequeñas), 768px (tablets), 1024px (laptops) y 1200px+ (escritorio amplio). No son leyes: el mejor breakpoint es el que tu diseño necesita, el punto donde "se rompe" visualmente.

Además del ancho, las media queries pueden detectar orientación (@media (orientation: landscape)), resolución, y preferencias del usuario como prefers-reduced-motion (que viste en la Unidad 7) o prefers-color-scheme (modo oscuro del sistema):

```

@media (prefers-color-scheme: dark) {
  body {
    background-color: #121212;
    color: #e0e0e0;
  }
}

```

Con esto tu sitio puede tener modo oscuro automático que sigue la configuración del teléfono del usuario. Es un detalle que impresiona y se logra con pocas líneas.

Unidades de medida: absolutas vs relativas

Elegir la unidad correcta es una de las decisiones más importantes del CSS responsive. Las unidades se dividen en absolutas (siempre miden lo mismo) y relativas (dependen de un contexto).

La unidad absoluta por excelencia es el píxel (px): `font-size: 16px` siempre son 16 píxeles, sin importar nada. Es predecible y perfecto para cosas que deben ser exactas: bordes de 1px, sombras, detalles finos. Su debilidad: no se adapta a nada.

Las unidades relativas sí se adaptan, cada una a su manera:

- % (porcentaje): relativo al elemento padre. `width: 50%` = la mitad del ancho del contenedor. Ideal para layouts fluidos.
- `em`: relativo al tamaño de fuente del propio elemento (o de su padre si el elemento no define uno). Si un elemento tiene `font-size: 20px`, entonces `padding: 1em` = 20px de padding. El problema del `em` es que se compone: si anidas elementos con `em`, los tamaños se multiplican y se vuelven impredecibles.
- `rem` (root `em`): relativo al tamaño de fuente del elemento raíz (`<html>`), que por defecto es 16px. `1.5rem = 24px`, siempre, en cualquier parte del documento. No se compone como el `em`, así que es predecible. Es la unidad favorita moderna para tamaños de fuente y espaciados.
- `vw` y `vh` (viewport width/height): relativos a la ventana del navegador. `100vw` = todo el ancho visible; `100vh` = todo el alto visible. `50vh` = la mitad de la altura de la pantalla. Perfectos para secciones hero de pantalla completa: `.hero {height: 100vh;}`.
- `vmin` y `vmax`: relativos a la dimensión menor o mayor del viewport. Útiles para elementos que deben verse bien en cualquier orientación.

La estrategia profesional: `px` para detalles exactos (bordes, sombras), `rem` para tipografía y espaciados (escalan si el usuario cambia el tamaño base de fuente en su navegador, lo cual es accesibilidad), % para anchos fluidos dentro de contenedores, y `vw` / `vh` para secciones de pantalla completa.

Un truco popular: definir `font-size: 62.5%` en el `html` para que `1rem` equivalga a 10px (el 62.5% de 16px), haciendo los cálculos mentales fáciles (`1.6rem = 16px`, `2.4rem = 24px`). Es controvertido porque interfiere con las preferencias de accesibilidad del usuario, así que úsalo sabiendo que hay debate al respecto; muchos equipos modernos prefieren trabajar directamente con el `rem` base de 16px.

Otro detalle: para tipografía que escala fluidamente entre el celular y el escritorio sin media queries, existe `clamp()`:

```
h1 {  
  font-size: clamp(28px, 5vw, 56px);  
}
```

Esto significa: "el título mide 5vw (5% del ancho de la pantalla), pero nunca menos de 28px ni más de 56px". En celular se ve a 28px, en escritorio a 56px, y en medio escala suavemente. Es tipografía responsive en una línea.

Variables CSS: un solo lugar para gobernarlos a todos

Las variables CSS (oficialmente "custom properties") te permiten guardar valores reutilizables con nombre y usarlos en todo tu CSS. Se definen con dos guiones al inicio y se usan con la función `var()`:

```
:root {  
  --color-primario: #1a1a2e;  
  --color-acento: #e94560;  
  --espaciado: 20px;  
  --radio-borde: 10px;  
}  
  
.boton {  
  background-color: var(--color-acento);  
  padding: 12px var(--espaciado);  
  border-radius: var(--radio-borde);  
}  
  
.tarjeta {  
  border: 1px solid var(--color-primario);  
  border-radius: var(--radio-borde);  
  padding: var(--espaciado);  
}
```

`:root` es un selector que apunta al elemento raíz del documento (`<html>`); las variables definidas ahí están disponibles en toda la página. El beneficio es enorme: si el cliente pide cambiar el color de acento de todo el sitio, cambias UNA línea en `:root` y los 50 lugares donde se usa se actualizan solos. Sin variables, tendrías que buscar y reemplazar 50 valores regados por el archivo, con el riesgo de olvidar alguno.

Las variables también aceptan un valor de respaldo por si la variable no existe: `var(--color, blue)` usa azul si `--color` no está definida. Esto evita que una variable mal escrita deje un estilo vacío.

Pero la característica más poderosa es que las variables respetan la cascada: puedes redefinirlas en contextos específicos y todos los usos dentro de ese contexto toman el nuevo valor:

```
:root {
  --fondo-tarjeta: white;
  --texto-tarjeta: #333;
}

.tarjeta {
  background-color: var(--fondo-tarjeta);
  color: var(--texto-tarjeta);
}

.modos-oscuro .tarjeta {
  --fondo-tarjeta: #1a1a2e;
  --texto-tarjeta: #e0e0e0;
}
```

Al agregar la clase `modos-oscuro` a un contenedor, TODAS las tarjetas dentro de él cambian a tema oscuro automáticamente, sin reescribir ni una regla de componente. Esta es la base de los sistemas de temas (claro/oscuro) modernos: no cambias los componentes, cambias las variables.

Las variables también se pueden modificar con JavaScript en tiempo real (`element.style.setProperty('--color-acento', 'green')`), lo que permite cosas como selectores de tema, personalización por el usuario y animaciones controladas por código. Es el puente entre CSS y JavaScript.

Errores comunes de esta unidad

Error 1: escribir media queries que contradicen el orden de la cascada. En mobile-first, las queries de `min-width` deben ir de menor a mayor; en desktop-first, las de `max-width` de mayor a menor. Si las desordenas, una query puede pisar a otra incorrectamente.

Error 2: olvidar el meta viewport en el HTML. Sin `<meta name="viewport" content="width=device-width, initial-scale=1">` en el `<head>`, los celulares simulan

una pantalla de ~980px y tu responsive no funcionará. Es la causa número uno de "mis media queries no hacen nada en el celular".

Error 3: usar solo px para todo y luego sufrir para hacerlo responsive. Reserva px para detalles; usa%, rem y vw / vh para lo estructural.

Error 4: anidar em sin control y terminar con tamaños impredecibles. Prefiere rem para tipografía a menos que quieras específicamente el comportamiento relativo al padre.

Error 5: definir variables pero seguir usando valores hardcodeados en algunas reglas, rompiendo la consistencia. Si defines --color-acento, úsala en TODOS lados donde va ese color.

Buenas prácticas

Mobile-first como filosofía por defecto. Breakpoints según tu diseño, no por copiar listas de internet. El meta viewport siempre. Unidades relativas para lo estructural, píxeles para detalles. Define tus variables de diseño (colores, espaciados, radios, fuentes) en :root al inicio del proyecto y úsalas religiosamente: tu CSS se vuelve un sistema de diseño en vez de una colección de valores sueltos.

Ejemplos paso a paso

Ejemplo 1: Landing responsive mobile-first

```
<header class="hero">
  <h1>Aprende diseño web</h1>
  <p>De cero a profesional, a tu ritmo.</p>
  <a href="#" class="cta">Empezar gratis</a>
</header>
<section class="beneficios">
  <div class="beneficio">...</div>
  <div class="beneficio">...</div>
  <div class="beneficio">...</div>
</section>
```

```
:root {
  --primario: #1a1a2e;
  --acento: #e94560;
  --texto: #333;
```

```

    --espaciado: 1.25rem;
}

/* Base: celular */
.hero {
  background: linear-gradient(135deg, var(--primario), #16213e);
  color: white;
  text-align: center;
  padding: 4rem 1.5rem;
}

.hero h1 {
  font-size: clamp(1.75rem, 6vw, 3.5rem);
  margin: 0 0 1rem 0;
  font-family: Arial, sans-serif;
}

.hero p {
  font-size: 1.1rem;
  margin-bottom: 2rem;
  opacity: 0.9;
}

.cta {
  display: inline-block;
  background-color: var(--acento);
  color: white;
  padding: 1rem 2.5rem;
  border-radius: 50px;
  text-decoration: none;
  font-weight: bold;
}

.beneficios {
  display: grid;
  grid-template-columns: 1fr;
  gap: var(--espaciado);
  padding: 2rem 1.5rem;
  max-width: 1100px;
  margin: 0 auto;
}

.beneficio {
  background: white;

```

```

border: 1px solid #eee;
border-radius: 12px;
padding: 1.5rem;
box-shadow: 0 2px 8px rgba(0, 0, 0, 0.08);
}

/* Tablet */
@media (min-width: 600px) {
  .beneficios {
    grid-template-columns: repeat(2, 1fr);
  }
}

/* Escritorio */
@media (min-width: 1024px) {
  .beneficios {
    grid-template-columns: repeat(3, 1fr);
  }
  .hero {
    padding: 6rem 2rem;
  }
}

```

Todo mobile-first: la base es celular (1 columna), y las media queries mejoran progresivamente. El título usa `clamp()` para escalar fluidamente. Las variables centralizan los colores.

Ejemplo 2: Menú hamburguesa responsive

```

<nav class="nav">
  <span class="logo">MiSitio</span>
  <button class="hamburguesa">[=]</button>
  <ul class="links">
    <li><a href="#">Inicio</a></li>
    <li><a href="#">Cursos</a></li>
    <li><a href="#">Blog</a></li>
    <li><a href="#">Contacto</a></li>
  </ul>
</nav>

```

```

/* Base móvil: menú colapsado */
.nav {
  display: flex;

```

```

    justify-content: space-between;
    align-items: center;
    background-color: #1a1a2e;
    padding: 1rem 1.5rem;
    font-family: Arial, sans-serif;
}

.logo {
    color: white;
    font-weight: bold;
    font-size: 1.3rem;
}

.hamburguesa {
    display: block;
    background: none;
    border: none;
    color: white;
    font-size: 1.8rem;
    cursor: pointer;
}

.links {
    display: none;
    list-style: none;
    padding: 0;
    margin: 0;
}

/* Escritorio: menú expandido, hamburguesa oculta */
@media (min-width: 768px) {
    .hamburguesa {
        display: none;
    }
    .links {
        display: flex;
        gap: 2rem;
    }
    .links a {
        color: white;
        text-decoration: none;
    }
    .links a:hover {
        color: #e94560;
    }
}

```

```
}  
}
```

El patrón responsive más famoso: en el celular se ve el botón hamburguesa (que con JavaScript mostraría los links), en escritorio los links se ven directos y la hamburguesa desaparece. Nota cómo `display: none / flex` se alternan con la media query.

Ejemplo 3: Sistema de variables con modo oscuro

```
:root {  
  --fondo: #ffffff;  
  --fondo-tarjeta: #f8f9fa;  
  --texto: #222222;  
  --texto-suave: #666666;  
  --acento: #e94560;  
  --borde: #e0e0e0;  
}  
  
body {  
  background-color: var(--fondo);  
  color: var(--texto);  
  font-family: Arial, sans-serif;  
  transition: background-color 0.3s, color 0.3s;  
}  
  
.tarjeta {  
  background-color: var(--fondo-tarjeta);  
  border: 1px solid var(--borde);  
  border-radius: 12px;  
  padding: 1.5rem;  
}  
  
.tarjeta p {  
  color: var(--texto-suave);  
}  
  
.tarjeta .btn {  
  background-color: var(--acento);  
  color: white;  
  border: none;  
  padding: 0.8rem 1.5rem;  
  border-radius: 8px;  
  cursor: pointer;  
}
```

```

/* Tema oscuro: solo se redefinen las variables */
body.moda-oscuro {
  --fondo: #121212;
  --fondo-tarjeta: #1e1e1e;
  --texto: #e0e0e0;
  --texto-suave: #a0a0a0;
  --borde: #333333;
}

```

Al agregar la clase `moda-oscuro` al `body` (típicamente con un botón y una línea de JavaScript), todo el sitio cambia de tema sin tocar ningún componente. El acento se mantiene igual en ambos temas para conservar la identidad de marca. La transición en el `body` suaviza el cambio.

Ejemplo 4: Tipografía fluida con clamp

```

:root {
  font-size: 16px;
}

h1 { font-size: clamp(2rem, 5vw, 3.5rem); }
h2 { font-size: clamp(1.5rem, 4vw, 2.5rem); }
h3 { font-size: clamp(1.25rem, 3vw, 1.75rem); }
p { font-size: 1rem; line-height: 1.7; }
.texto-pequeno { font-size: 0.875rem; }

```

Toda la escala tipográfica en 6 líneas, responsive sin media queries. Los títulos crecen con la pantalla entre un mínimo legible en el celular y un máximo elegante en escritorio. El cuerpo se queda en 1rem (16px) siempre: el texto de lectura no debe escalar agresivamente.

Ejemplo 5: Hero de pantalla completa

```

<section class="hero-full">
  <div class="hero-contenido">
    <h1>Tu próximo nivel empieza aquí</h1>
    <p>Únete a miles de estudiantes.</p>
    <a href="#" class="cta">Comenzar</a>
  </div>
</section>

```

```

.hero-full {
  min-height: 100vh;
  min-height: 100svh; /* versión moderna que considera barras del móvil */
  display: flex;
  justify-content: center;
  align-items: center;
  text-align: center;
  background: linear-gradient(rgba(26, 26, 46, 0.7), rgba(26, 26, 46, 0.7)),
    url("fondo.jpg") center/cover no-repeat;
  color: white;
  font-family: Arial, sans-serif;
  padding: 2rem;
}

.hero-full h1 {
  font-size: clamp(2rem, 6vw, 4rem);
  margin-bottom: 1rem;
}

```

100vh hace que la sección mida exactamente lo alto de la pantalla. El 100svh es la versión moderna que descuenta las barras del navegador del celular (detalle fino pero profesional). El degradado semitransparente sobre la imagen garantiza legibilidad del texto.

Práctica

Ejercicio 1

Escribe una media query que haga que un contenedor de 900px pase a ocupar el 95% del ancho en pantallas de 768px o menos.

SOLUCIÓN:

```
.contenedor {
  width: 900px;
  margin: 0 auto;
}

@media (max-width: 768px) {
  .contenedor {
    width: 95%;
  }
}
```

Enfoque desktop-first: base para escritorio y ajuste hacia abajo con `max-width`.

Ejercicio 2

Convierte este CSS desktop-first a mobile-first:

```
.cajas { display: grid; grid-template-columns: repeat(3, 1fr); gap: 20px; }

@media (max-width: 768px) {
  .cajas { grid-template-columns: 1fr; }
}
```

SOLUCIÓN:

```
.cajas {
  display: grid;
  grid-template-columns: 1fr;
  gap: 20px;
}

@media (min-width: 768px) {
  .cajas {
    grid-template-columns: repeat(3, 1fr);
  }
}
```

La base ahora es para celular (1 columna) y la mejora es progresiva con `min-width`. Mismo resultado visual, filosofía moderna.

Ejercicio 3

Define variables para los colores de una marca (primario #1a1a2e, acento #e94560, fondo #f8f9fa) y úsalas en un botón y una tarjeta.

SOLUCIÓN:

```
:root {
  --primario: #1a1a2e;
  --acento: #e94560;
  --fondo: #f8f9fa;
}

.boton {
  background-color: var(--acento);
  color: white;
  padding: 12px 28px;
  border: none;
  border-radius: 8px;
}

.tarjeta {
  background-color: var(--fondo);
  border-top: 4px solid var(--primario);
  padding: 20px;
  border-radius: 8px;
}
```

Si la marca cambia su acento mañana, es un cambio de una línea.

Ejercicio 4

Explica por qué este responsive "no funciona" en un celular real aunque sí funciona al redimensionar la ventana en la computadora:

```
@media (max-width: 600px) {
  .menu { display: none; }
}
```

SOLUCIÓN: Casi seguro falta el meta viewport en el <head> del HTML. Sin <meta name="viewport" content="width=device-width, initial-scale=1">, el celular renderiza la página como si tuviera ~980px de ancho y luego la encoge, así que la

condición de 600px nunca se cumple. Al redimensionar la ventana en la computadora sí funciona porque ahí el ancho es real. La solución es agregar el meta viewport.

Ejercicio 5

Haz que un título mida 32px en el celular, escale con la pantalla en tablets y no pase de 64px en escritorio, sin media queries.

SOLUCIÓN:

```
h1 {  
  font-size: clamp(32px, 6vw, 64px);  
}
```

El `clamp()` establece mínimo, valor fluido y máximo en una sola línea. En un celular de 375px, $6vw = 22.5px$, pero el mínimo de 32px prevalece; en escritorio de 1440px, $6vw = 86px$, pero el máximo de 64px lo limita.

Ejercicio 6

Crea un modo oscuro que se active automáticamente según la preferencia del sistema, cambiando fondo, texto y color de tarjetas.

SOLUCIÓN:

```

:root {
  --fondo: #ffffff;
  --texto: #222222;
  --tarjeta: #f5f5f5;
}

body {
  background-color: var(--fondo);
  color: var(--texto);
  font-family: Arial, sans-serif;
}

.tarjeta {
  background-color: var(--tarjeta);
  padding: 20px;
  border-radius: 10px;
}

@media (prefers-color-scheme: dark) {
  :root {
    --fondo: #121212;
    --texto: #e0e0e0;
    --tarjeta: #1e1e1e;
  }
}

```

Como todo usa variables, la media query solo redefine los valores y el sitio completo cambia de tema automáticamente según el modo del sistema del usuario.

Resumen de la unidad

En esta unidad aprendiste las media queries (`@media (max-width/min-width)`) con las dos filosofías: `desktop-first` (ajustes hacia abajo) y `mobile-first` (base para celular + mejoras progresivas, la recomendada). Viste los breakpoints típicos, el meta viewport obligatorio, y condiciones avanzadas como `orientation` y `prefers-color-scheme` para modo oscuro automático. Dominaste las unidades: `px` para detalles exactos, `%` para fluidez respecto al padre, `em` vs `rem` (prefiere `rem` para no componer), `vw` / `vh` para pantalla completa, y `clamp()` para tipografía fluida sin media queries. Y aprendiste las variables CSS (`--nombre` en `:root`, `var()` para usarlas, valores de respaldo, redefinición por contexto para temas claro/oscuro y modificación desde JavaScript).

Puntos clave: mobile-first por defecto; el meta viewport siempre; unidades relativas para estructura; y centraliza tu diseño en variables desde el día uno.

UNIDAD 10: PROYECTOS INTEGRADORES Y GRADUACIÓN

Introducción a la unidad

Llegaste a la última unidad, y es la más importante: aquí dejas de practicar piezas sueltas y armas proyectos completos con todo lo que aprendiste. La diferencia entre "saber CSS" y "ser diseñador web" está en la capacidad de combinar decenas de propiedades en componentes reales que se ven profesionales, funcionan en celular y escritorio, y tienen esos detalles de pulido que distinguen el trabajo amateur del profesional.

En esta unidad construirás dos proyectos paso a paso: una tarjeta de perfil (el componente más común de la web: lo verás en redes sociales, equipos de trabajo, testimonios y catálogos) y una landing page responsive completa (hero, beneficios, testimonios, llamada a la acción y pie). Al final celebraremos tu graduación con un repaso de todo el camino y tu mapa de ruta hacia JavaScript, el siguiente paso natural.

Explicación detallada

Proyecto 1: Tarjeta de perfil profesional

La tarjeta de perfil es el "hola mundo" de los componentes reales. Parece simple, pero combina casi todo lo del curso: modelo de caja, bordes redondeados, sombras, tipografía, flexbox, posicionamiento, transiciones y variables. Vamos a diseccionarla antes de construirla.

Anatomía de una tarjeta de perfil: una foto circular (border-radius 50%), el nombre en tipografía destacada, el puesto o rol en texto suave, una breve descripción, estadísticas o datos (seguidores, proyectos, ubicación) y un botón de acción (seguir, contactar). Opcionalmente, una portada decorativa arriba con la foto traslapada (el patrón de las redes sociales).

Las decisiones de diseño: la tarjeta necesita aire (padding generoso), jerarquía visual clara (el nombre más grande que el puesto, el puesto más grande que la descripción), un punto focal (la foto), y un estado hover que invite a la interacción (elevación suave). Los colores deben venir de variables para mantener consistencia con el resto del sitio.

El patrón de la foto traslapada sobre la portada usa posicionamiento: la portada es un div con altura fija y fondo degradado, y la foto usa `margin-top` negativo o `position: relative` con desplazamiento para montarse sobre la portada. El margin negativo es la técnica más simple: `.foto {margin-top: -60px;}` la sube sobre la portada, y como la tarjeta tiene `overflow: visible` (el valor por defecto), la foto se muestra traslapada sin problemas. Para que se vea bien, la foto necesita un borde blanco grueso que la separe visualmente de la portada.

Las estadísticas se hacen con flexbox: tres bloques iguales (`flex: 1`) con número grande y etiqueta pequeña. El botón de acción lleva los estados `hover/active` que ya dominas. Y toda la tarjeta lleva la transición de elevación al `hover`.

Proyecto 2: Landing page responsive

Una landing page (página de aterrizaje) tiene un objetivo: convertir visitantes en clientes, suscriptores o usuarios. Su estructura clásica es: hero (propuesta de valor + llamada a la acción), beneficios o características, prueba social (testimonios), llamada a la acción final y pie. Cada sección usa técnicas distintas del curso, y el conjunto debe ser responsive.

El hero: pantalla completa o generosa (`min-height` con `vh`), fondo con imagen + overlay oscuro para legibilidad, título con `clamp()` para escala fluida, subtítulo y botón CTA prominente. Centrado con flexbox en ambos ejes.

Los beneficios: rejilla responsive con `repeat(auto-fit, minmax(250px, 1fr))` -sin media queries-, cada beneficio con icono (puede ser un emoji o un círculo con inicial), título y descripción. Hover de elevación para interactividad.

Los testimonios: tarjetas con cita estilizada (el patrón de la Unidad 2), foto circular pequeña, nombre y rol. En el celular una columna, en escritorio tres.

La llamada a la acción final: sección con fondo de color de acento o degradado, título directo ("¿Listo para empezar?") y botón contrastante (si el fondo es de color, el botón es blanco con texto de color).

El pie: fondo oscuro, columnas con enlaces (grid o flex), y línea de copyright. En el celular las columnas se apilan.

El responsive de la landing: con `mobile-first`, la base es una columna y las rejillas crecen con `auto-fit` o media queries puntuales. La navegación (si la hay) usa el patrón hamburguesa de la Unidad 9. Las imágenes usan `max-width: 100%`. La tipografía usa `clamp()`.

Detalles de pulido profesional

Lo que separa una landing "de curso" de una "de cliente" son los detalles: espaciado vertical consistente entre secciones (por ejemplo, `padding: 5rem 1.5rem` en cada sección), una paleta limitada a 3-4 colores usada religiosamente, máximo dos familias tipográficas, bordes redondeados consistentes (8px para componentes pequeños, 16px para tarjetas), sombras por niveles (sutil en reposo, mayor en hover), y transiciones en todo lo interactivo.

Otro detalle: los estados vacíos y los focos. Cada botón debe tener hover, active y focus visibles. Cada campo debe tener su placeholder, su foco y sus estados de validación. Estos detalles no se ven en las capturas de pantalla, pero se sienten al usar el sitio.

Tu graduación: el mapa completo

Haz una pausa y mira todo lo que dominas ahora. Empezaste entendiendo qué es CSS y cómo seleccionar elementos (Unidad 1). Aprendiste a vestir con color, tipografía y fondos (Unidad 2). Entendiste que todo es una caja con cuatro capas (Unidad 3). Controlaste tamaños, displays y posicionamiento (Unidad 4). Conquistaste flexbox y grid (Unidades 5 y 6). Le diste vida con movimiento (Unidad 7). Puliste con sombras, listas, tablas y formularios (Unidad 8). Lo hiciste responsive con media queries, unidades relativas y variables (Unidad 9). Y ahora armaste proyectos completos (Unidad 10).

Eres capaz de tomar un diseño (una imagen, un boceto, una idea) y convertirlo en una página web real, responsive y pulida. Eso es exactamente lo que hacen los desarrolladores frontend todos los días, y es una habilidad que se paga bien en el mercado laboral.

El siguiente paso: JavaScript

CSS controla la apariencia; JavaScript controla el comportamiento. Con JavaScript tus páginas dejarán de ser documentos bonitos para convertirse en aplicaciones: menús que se abren y cierran, formularios que validan en tiempo real, carruseles, contadores, datos que llegan de internet, juegos, y todo lo interactivo que ves en la web moderna.

El camino natural es: HTML (estructura) -> CSS (presentación, este libro) -> JavaScript (comportamiento). Después de JavaScript vienen los frameworks (React, Vue), y con esos tres pilares eres desarrollador frontend completo. La academia tiene el curso de JavaScript esperándote en lucasegaacademy.com, gratis, como todo lo demás.

Mi consejo final: no dejes de construir. El conocimiento de este libro se consolida construyendo proyectos reales: clona las páginas que admiras, haz tu portafolio, ofrece hacer el sitio de un

negocio local. Cada proyecto te enseñará algo que ningún libro puede. Y cuando te atores -porque te vas a atorar, todos nos atoramos-, recuerda que la respuesta casi siempre está en las herramientas de desarrollo del navegador y en volver a los fundamentos de este libro.

¡Felicidades, eres Frontend!

Ejemplos paso a paso

Ejemplo 1: Tarjeta de perfil completa

```
<article class="perfil">
  <div class="portada"></div>
  
  <h2 class="nombre">Ana Martínez</h2>
  <p class="puesto">Diseñadora UX Senior</p>
  <p class="bio">Creo experiencias digitales que la gente ama usar. 8 años creando productos.</p>
  <div class="stats">
    <div class="stat"><strong>248</strong><span>Proyectos</span></div>
    <div class="stat"><strong>12k</strong><span>Seguidores</span></div>
    <div class="stat"><strong>8</strong><span>Años</span></div>
  </div>
  <button class="seguir">Seguir</button>
</article>
```

```
:root {
  --primario: #1a1a2e;
  --acento: #e94560;
  --texto: #333;
  --suave: #888;
}

.perfil {
  width: 320px;
  background: white;
  border-radius: 18px;
  overflow: hidden;
  box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
  text-align: center;
  font-family: Arial, sans-serif;
  transition: transform 0.3s ease, box-shadow 0.3s ease;
}
```

```
.perfil:hover {
  transform: translateY(-8px);
  box-shadow: 0 15px 35px rgba(0, 0, 0, 0.16);
}

.portada {
  height: 110px;
  background: linear-gradient(135deg, #667eea, #764ba2);
}

.foto {
  width: 110px;
  height: 110px;
  border-radius: 50%;
  border: 5px solid white;
  margin-top: -55px;
  object-fit: cover;
}

.nombre {
  margin: 12px 0 4px 0;
  color: var(--primario);
  font-size: 1.4rem;
}

.puesto {
  margin: 0;
  color: var(--acento);
  font-weight: bold;
  font-size: 0.9rem;
  text-transform: uppercase;
  letter-spacing: 1px;
}

.bio {
  color: var(--suave);
  font-size: 0.95rem;
  line-height: 1.6;
  padding: 0 25px;
}

.stats {
  display: flex;
```

```
margin: 20px 0;
border-top: 1px solid #f0f0f0;
border-bottom: 1px solid #f0f0f0;
}

.stat {
  flex: 1;
  padding: 15px 0;
  display: flex;
  flex-direction: column;
  gap: 4px;
}

.stat strong {
  font-size: 1.3rem;
  color: var(--primario);
}

.stat span {
  font-size: 0.8rem;
  color: var(--suave);
  text-transform: uppercase;
  letter-spacing: 0.5px;
}

.seguir {
  background-color: var(--acento);
  color: white;
  border: none;
  padding: 12px 50px;
  border-radius: 25px;
  font-size: 1rem;
  font-weight: bold;
  cursor: pointer;
  margin-bottom: 25px;
  transition: background-color 0.3s, transform 0.2s;
}

.seguir:hover {
  background-color: #d13a54;
  transform: scale(1.05);
}
```

Cada técnica del curso en un solo componente: degradado en la portada, círculo con borde blanco y margin negativo para traslapar, variables para colores, flexbox para stats, hover de elevación en la tarjeta y de escala en el botón, tipografía con jerarquía. Si entiendes cada línea de este ejemplo, entiendes CSS.

Ejemplo 2: Landing page responsive completa

```
<header class="hero">
  <nav class="topnav">
    <span class="logo">AcademiaPro</span>
    <a href="#beneficios" class="nav-cta">Empezar</a>
  </nav>
  <div class="hero-texto">
    <h1>Domina el diseño web</h1>
    <p>Aprende CSS desde cero hasta nivel profesional, con proyectos reales.</p>
    <a href="#" class="cta-grande">Comenzar gratis</a>
  </div>
</header>

<section class="seccion" id="beneficios">
  <h2 class="titulo-seccion">¿Por qué nosotros?</h2>
  <div class="rejilla-beneficios">
    <div class="beneficio">
      <div class="icono"></div>
      <h3>100% gratis</h3>
      <p>Todos los cursos sin costo, para siempre.</p>
    </div>
    <div class="beneficio">
      <div class="icono"></div>
      <h3>Proyectos reales</h3>
      <p>Aprende construyendo, no solo mirando.</p>
    </div>
    <div class="beneficio">
      <div class="icono"></div>
      <h3>A tu ritmo</h3>
      <p>Sin horarios, sin presiones, sin límites.</p>
    </div>
  </div>
</section>

<section class="seccion testimonios">
  <h2 class="titulo-seccion">Lo que dicen los estudiantes</h2>
  <div class="rejilla-testimonios">
```

```

    <blockquote class="testimonio">
      <p>"En dos meses pasé de no saber nada a tener mi portafolio."</p>
      <footer>- Carlos R., Desarrollador Jr.</footer>
    </blockquote>
    <blockquote class="testimonio">
      <p>"Los proyectos me consiguieron mi primer cliente freelance."</p>
      <footer>- Lucía F., Freelancer</footer>
    </blockquote>
    <blockquote class="testimonio">
      <p>"La mejor forma de aprender CSS que he encontrado."</p>
      <footer>- Diego M., Estudiante</footer>
    </blockquote>
  </div>
</section>

<section class="cta-final">
  <h2>¿Listo para empezar?</h2>
  <p>Únete hoy y publica tu primera página esta semana.</p>
  <a href="#" class="cta-blanco">Crear mi cuenta gratis</a>
</section>

<footer class="pie">
  <p>© 2026 AcademiaPro - Hecho con CSS y café</p>
</footer>

```

```

:root {
  --primario: #1a1a2e;
  --acento: #e94560;
  --fondo-suave: #f8f9fa;
}

* { box-sizing: border-box; }

body {
  margin: 0;
  font-family: Arial, sans-serif;
  color: #333;
}

/* HERO */
.hero {
  min-height: 90vh;
  background: linear-gradient(rgba(26,26,46,0.75), rgba(26,26,46,0.75)),
    url("hero.jpg") center/cover no-repeat;
  color: white;
}

```

```

    display: flex;
    flex-direction: column;
}

.topnav {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 1.2rem 1.5rem;
}

.logo { font-weight: bold; font-size: 1.3rem; }

.nav-cta {
    color: white;
    text-decoration: none;
    border: 2px solid white;
    padding: 0.6rem 1.5rem;
    border-radius: 25px;
    font-weight: bold;
    transition: background-color 0.3s, color 0.3s;
}

.nav-cta:hover { background-color: white; color: var(--primario); }

.hero-texto {
    flex: 1;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    text-align: center;
    padding: 2rem;
}

.hero-texto h1 {
    font-size: clamp(2.2rem, 6vw, 4rem);
    margin: 0 0 1rem 0;
}

.hero-texto p {
    font-size: 1.2rem;
    opacity: 0.9;
    margin-bottom: 2rem;
}

```

```

    max-width: 600px;
}

.cta-grande {
    background-color: var(--acento);
    color: white;
    text-decoration: none;
    padding: 1.1rem 3rem;
    border-radius: 50px;
    font-size: 1.1rem;
    font-weight: bold;
    transition: transform 0.25s, box-shadow 0.25s;
}

.cta-grande:hover {
    transform: translateY(-3px);
    box-shadow: 0 12px 25px rgba(233, 69, 96, 0.4);
}

/* SECCIONES */
.seccion { padding: 4.5rem 1.5rem; }

.titulo-seccion {
    text-align: center;
    font-size: clamp(1.6rem, 4vw, 2.4rem);
    color: var(--primario);
    margin: 0 0 3rem 0;
}

.titulo-seccion::after {
    content: "";
    display: block;
    width: 60px;
    height: 4px;
    background-color: var(--acento);
    margin: 1rem auto 0;
    border-radius: 2px;
}

.rejilla-beneficios {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
    gap: 1.5rem;
    max-width: 1100px;
}

```

```

    margin: 0 auto;
}

.beneficio {
    text-align: center;
    padding: 2rem 1.5rem;
    border-radius: 14px;
    border: 1px solid #eee;
    transition: transform 0.3s, box-shadow 0.3s;
}

.beneficio:hover {
    transform: translateY(-8px);
    box-shadow: 0 14px 30px rgba(0, 0, 0, 0.12);
}

.icono {
    font-size: 2.8rem;
    margin-bottom: 1rem;
}

.beneficio h3 { color: var(--primario); margin: 0 0 0.6rem 0; }
.beneficio p { color: #666; line-height: 1.6; margin: 0; }

/* TESTIMONIOS */
.testimonios { background-color: var(--fondo-suave); }

.rejilla-testimonios {
    display: grid;
    grid-template-columns: 1fr;
    gap: 1.5rem;
    max-width: 1100px;
    margin: 0 auto;
}

@media (min-width: 768px) {
    .rejilla-testimonios { grid-template-columns: repeat(3, 1fr); }
}

.testimonio {
    background: white;
    border-radius: 14px;
    padding: 1.8rem;
    margin: 0;
}

```

```

    box-shadow: 0 3px 12px rgba(0, 0, 0, 0.08);
}

.testimonio p {
    font-style: italic;
    line-height: 1.7;
    color: #444;
}

.testimonio footer {
    margin-top: 1rem;
    font-weight: bold;
    color: var(--acento);
    font-size: 0.9rem;
}

/* CTA FINAL */
.cta-final {
    background: linear-gradient(135deg, var(--acento), #ff7e5f);
    color: white;
    text-align: center;
    padding: 4.5rem 1.5rem;
}

.cta-final h2 {
    font-size: clamp(1.8rem, 5vw, 2.8rem);
    margin: 0 0 1rem 0;
}

.cta-final p { font-size: 1.15rem; margin-bottom: 2rem; opacity: 0.95; }

.cta-blanco {
    display: inline-block;
    background-color: white;
    color: var(--acento);
    text-decoration: none;
    padding: 1.1rem 3rem;
    border-radius: 50px;
    font-weight: bold;
    font-size: 1.1rem;
    transition: transform 0.25s, box-shadow 0.25s;
}

.cta-blanco:hover {

```

```
    transform: scale(1.05);
    box-shadow: 0 10px 25px rgba(0, 0, 0, 0.25);
}

/* PIE */
.pie {
    background-color: var(--primario);
    color: #aaa;
    text-align: center;
    padding: 2rem 1.5rem;
    font-size: 0.9rem;
}
```

Esta landing usa literalmente todo el curso: variables, clamp, flexbox, grid con auto-fit, media queries, pseudo-elemento decorativo, degradados, sombras por niveles, transiciones en todo lo interactivo, overlay sobre imagen de fondo y mobile-first. Es tu proyecto de graduación: si puedes leer y entender cada línea, estás listo.

Práctica

Ejercicio 1

Agrega a la tarjeta de perfil una insignia "PRO" en la esquina superior derecha de la portada usando posicionamiento.

SOLUCIÓN: La portada necesita `position: relative` y la insignia `position: absolute` :

```
.portada { position: relative; }

.insignia-pro {
  position: absolute;
  top: 12px;
  right: 12px;
  background-color: #ffd93d;
  color: #1a1a2e;
  font-size: 0.75rem;
  font-weight: bold;
  padding: 4px 12px;
  border-radius: 12px;
  letter-spacing: 1px;
}
```

```
<div class="portada"><span class="insignia-pro">PRO</span></div>
```

El patrón padre-relative/hijo-absolute de la Unidad 4, aplicado a un caso real.

Ejercicio 2

Haz que las tarjetas de beneficio de la landing muestren 1 columna en el celular, 2 en tablet y 3 en escritorio usando media queries (sin auto-fit).

SOLUCIÓN:

```
.rejilla-beneficios {
  display: grid;
  grid-template-columns: 1fr;
  gap: 1.5rem;
}

@media (min-width: 600px) {
  .rejilla-beneficios { grid-template-columns: repeat(2, 1fr); }
}

@media (min-width: 1024px) {
  .rejilla-beneficios { grid-template-columns: repeat(3, 1fr); }
}
```

Mobile-first: la base es 1 columna y las mejoras son progresivas. (El auto-fit del ejemplo principal logra lo mismo con menos código; esta es la versión explícita.)

Ejercicio 3

Agrega un modo oscuro a la landing usando variables y una clase en el body.

SOLUCIÓN: Primero convierte los colores fijos a variables (ya están en :root), luego:

```
body.moda-oscuro {
  --fondo-suave: #1a1a2e;
}

body.moda-oscuro .beneficio {
  background-color: #16213e;
  border-color: #2a2a4a;
}

body.moda-oscuro .beneficio h3 { color: white; }
body.moda-oscuro .beneficio p { color: #b8b8c8; }
body.moda-oscuro .testimonio { background-color: #16213e; }
body.moda-oscuro .testimonio p { color: #d0d0d0; }
```

Con un botón que alterne la clase `moda-oscuro` en el body (una línea de JavaScript), toda la landing cambia de tema.

Ejercicio 4

Crea una sección de "precios" con 3 planes para agregar a la landing, donde el plan del medio destaque visualmente.

SOLUCIÓN:

```

<section class="seccion precios-seccion">
  <h2 class="titulo-seccion">Planes</h2>
  <div class="planes">
    <div class="plan">
      <h3>Gratis</h3><p class="monto">$0</p>
      <ul><li>Cursos básicos</li><li>Comunidad</li></ul>
      <a href="#" class="btn-plan">Elegir</a>
    </div>
    <div class="plan popular">
      <span class="etiqueta-pop">Más popular</span>
      <h3>Pro</h3><p class="monto">$199<span>/mes</span></p>
      <ul><li>Todos los cursos</li><li>Certificados</li><li>Soporte
priority</li></ul>
      <a href="#" class="btn-plan">Elegir</a>
    </div>
    <div class="plan">
      <h3>Equipos</h3><p class="monto">$499</p>
      <ul><li>Todo lo Pro</li><li>5 cuentas</li></ul>
      <a href="#" class="btn-plan">Elegir</a>
    </div>
  </div>
</section>

```

```

.planes {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(260px, 1fr));
  gap: 1.5rem;
  max-width: 1000px;
  margin: 0 auto;
  align-items: center;
}

.plan {
  border: 1px solid #e0e0e0;
  border-radius: 16px;
  padding: 2rem;
  text-align: center;
  position: relative;
  background: white;
}

.plan.popular {
  border: 2px solid var(--acento);
  box-shadow: 0 15px 40px rgba(233, 69, 96, 0.2);
}

```

```

    transform: scale(1.05);
}

.etiqueta-pop {
  position: absolute;
  top: -14px;
  left: 50%;
  transform: translateX(-50%);
  background: var(--acento);
  color: white;
  font-size: 0.75rem;
  font-weight: bold;
  padding: 5px 18px;
  border-radius: 15px;
  text-transform: uppercase;
  letter-spacing: 1px;
  white-space: nowrap;
}

.plan ul { list-style: none; padding: 0; margin: 1.5rem 0; }
.plan li { padding: 0.5rem 0; color: #666; border-bottom: 1px solid #f0f0f0; }
.plan li:last-child { border-bottom: none; }

.monto { font-size: 2.5rem; font-weight: bold; color: var(--primario); margin: 1rem 0; }
.monto span { font-size: 1rem; color: #888; font-weight: normal; }

.btn-plan {
  display: inline-block;
  background: var(--primario);
  color: white;
  text-decoration: none;
  padding: 0.9rem 2.5rem;
  border-radius: 25px;
  font-weight: bold;
  transition: background-color 0.3s;
}

.plan.popular .btn-plan { background: var(--acento); }
.btn-plan:hover { filter: brightness(1.15); }

```

El plan popular destaca con borde de acento, sombra de color, escala y etiqueta flotante posicionada con el truco de centrado absoluto (left: 50% + translateX(-50%)).

Ejercicio 5

Explica qué hace cada línea de este CSS y qué efecto produce en conjunto:

```
.hero-texto h1 {  
  font-size: clamp(2.2rem, 6vw, 4rem);  
}
```

SOLUCIÓN: Define el tamaño del título del hero con tres límites: mínimo 2.2rem (35px, legible en el celular más pequeño), valor fluido 6vw (6% del ancho de la pantalla: crece suavemente al agrandar la ventana) y máximo 4rem (64px, para que no se vea gigante en monitores enormes). En conjunto produce un título que escala fluidamente entre el celular y el escritorio sin necesidad de media queries: tipografía responsive en una línea.

Ejercicio 6

Proyecto final: combina todo para crear una tarjeta de producto para tienda en línea con imagen, insignia de descuento, nombre, precio con descuento, botón "Agregar" y hover de elevación. Debe ser responsive.

SOLUCIÓN:

```
<article class="producto-card">  
  <div class="img-wrap">  
      
    <span class="dto">-25%</span>  
  </div>  
  <div class="info">  
    <h3>Audífonos Bluetooth Pro</h3>  
    <p class="precios"><span class="ahora">$1,499</span> <span  
class="antes">$1,999</span></p>  
    <button class="agregar">Agregar al carrito</button>  
  </div>  
</article>
```

```
.producto-card {  
  max-width: 300px;  
  background: white;  
  border-radius: 16px;  
  overflow: hidden;  
  box-shadow: 0 3px 12px rgba(0, 0, 0, 0.1);
```

```

    font-family: Arial, sans-serif;
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.producto-card:hover {
    transform: translateY(-8px);
    box-shadow: 0 15px 35px rgba(0, 0, 0, 0.16);
}

.img-wrap { position: relative; }

.img-wrap img {
    width: 100%;
    height: 200px;
    object-fit: cover;
    display: block;
}

.dto {
    position: absolute;
    top: 12px;
    right: 12px;
    background: var(--acento, #e94560);
    color: white;
    font-weight: bold;
    font-size: 0.85rem;
    padding: 6px 14px;
    border-radius: 20px;
}

.info { padding: 1.2rem 1.4rem 1.5rem; }

.info h3 { margin: 0 0 0.6rem 0; color: #1a1a2e; font-size: 1.1rem; }

.precios { margin: 0 0 1rem 0; }

.ahora { font-size: 1.5rem; font-weight: bold; color: #e94560; }
.antes { text-decoration: line-through; color: #999; margin-left: 8px; }

.agregar {
    width: 100%;
    padding: 0.9rem;
    background: #1a1a2e;
    color: white;

```

```
border: none;
border-radius: 10px;
font-size: 1rem;
font-weight: bold;
cursor: pointer;
transition: background-color 0.3s;
}

.agregar:hover { background: #e94560; }
.agregar:active { transform: scale(0.98); }
```

El `object-fit: cover` hace que la imagen llene su contenedor sin deformarse (recortando lo necesario). El `overflow: hidden` en la tarjeta con `border-radius` redondea también la imagen. Es responsive por naturaleza: `max-width + width: 100%` de la imagen.

Resumen de la unidad

En esta unidad integradora construiste dos proyectos completos: una tarjeta de perfil (portada con degradado, foto traslapada con margin negativo, stats con flexbox, botón con estados, hover de elevación) y una landing page responsive (hero de pantalla completa con overlay, navegación, beneficios con auto-fit, testimonios, CTA final y pie, todo mobile-first con variables y clamp). Viste los detalles de pulido que separan lo amateur de lo profesional: espaciado consistente, paleta limitada, transiciones en todo lo interactivo, focos visibles y estados completos. Y celebraste tu graduación con el repaso del camino completo y el siguiente paso: JavaScript.

Puntos clave: un componente real combina 10+ propiedades trabajando juntas; mobile-first + variables + clamp es la base del responsive moderno; y la mejor forma de consolidar lo aprendido es construir proyectos reales, uno tras otro.

¡Felicidades! Completaste el Curso Completo de CSS3 de Lucas EGA Academy. Eres Frontend. El siguiente paso te espera en lucasegaacademy.com: JavaScript, para darle vida a todo lo que ahora sabes vestir.

Ampliación Unidad 1: casos de estudio y más práctica

Caso de estudio 1: rescatando un sitio con estilos en línea

Te contratan para darle mantenimiento a un sitio viejo donde todo el estilo está en atributos `style` regados por el HTML. Hay 40 páginas y el cliente quiere cambiar el color principal de rojo a azul. Con estilos en línea, tendrías que editar cientos de atributos a mano.

La solución profesional: crear una hoja externa, definir clases semánticas y migrar página por página. Primero defines en `estilos.css` las clases que representan los patrones repetidos:

```
.btn-primario {
  background-color: blue;
  color: white;
  padding: 12px 28px;
  border: none;
  border-radius: 8px;
}

.titulo-seccion {
  color: #1a1a2e;
  font-size: 28px;
}
```

Luego reemplazas cada `style="background-color: red; ..."` por `class="btn-primario"`. El cambio de color futuro será de una línea. Moraleja: los estilos en línea no escalan; las clases sí.

Caso de estudio 2: depurando especificidad con las herramientas del navegador

Un botón debería ser verde pero se ve gris. Abres F12, seleccionas el botón y en el panel de estilos ves:

```
.btn { background-color: green; }           /* tachado */
#formulario .btn { background-color: gray; } /* aplicado */
```

El navegador te muestra la regla ganadora y tacha las perdedoras. Aquí `#formulario.btn` (100 + 10 = 110 puntos) le gana a `.btn` (10 puntos). Soluciones: hacer tu selector más específico (`#formulario.btn-accion`), o mejor, revisar por qué hay un estilo gris tan específico y simplificar la arquitectura. La lección: nunca adivines la especificidad, inspeccionala.

Caso de estudio 3: nomenclatura que sobrevive al tiempo

Comparamos dos hojas de estilo del mismo proyecto, una de hace dos años:

```
/* Mal: describe apariencia */
.rojo-grande { color: red; font-size: 30px; }
.caja-azul { background: blue; }

/* Bien: describe proposito */
.titulo-alerta { color: red; font-size: 30px; }
.tarjeta-info { background: blue; }
```

Cuando el diseño cambio y la alerta paso a ser naranja, la clase `.rojo-grande` se volvio una mentira que confundio a todo el equipo, mientras `.titulo-alerta` siguio teniendo sentido. Nombra por proposito, no por apariencia.

Más práctica

Ejercicio 7

Un párrafo tiene `class="nota"` e `id="aviso"`. Estas reglas aplican: `p {color: black;}`, `.nota {color: blue;}`, `#aviso {color: red;}` y además el HTML trae `style="color: green"`. ¿De que color se ve y por que?

SOLUCIÓN: Verde. El estilo en línea vale 1000 puntos de especificidad, más que el id (100), la clase (10) y la etiqueta (1). El orden de prioridad completo: en línea > id > clase > etiqueta.

Ejercicio 8

Escribe un selector que aplique a todos los enlaces dentro de un elemento con id "menu", y otro que aplique solo a los párrafos con clase "intro" que esten dentro de un article.

SOLUCIÓN:

```
#menu a {
  color: white;
}

article p.intro {
  font-size: 18px;
}
```

El primero es un selector descendiente: cualquier `a` dentro de `#menu`. El segundo combina etiqueta y clase con descendencia: `p` con clase `intro` dentro de `article`. Nota que `p.intro` (sin espacio) significa "párrafos que ADEMÁS tienen la clase", mientras `article p.intro` (con espacio) significa "dentro de `article`".

Ejercicio 9

Convierte estos estilos en línea a una hoja externa con clases reutilizables:

```
<h2 style="color: navy; font-size: 26px;">Titulo 1</h2>
<h2 style="color: navy; font-size: 26px;">Titulo 2</h2>
<p style="color: gray; font-size: 15px;">Texto 1</p>
<p style="color: gray; font-size: 15px;">Texto 2</p>
```

SOLUCIÓN: Detectas dos patrones repetidos y creas dos clases:

```
.titulo-bloque {
  color: navy;
  font-size: 26px;
}

.texto-bloque {
  color: gray;
  font-size: 15px;
}
```

```
<h2 class="titulo-bloque">Titulo 1</h2>
<h2 class="titulo-bloque">Titulo 2</h2>
<p class="texto-bloque">Texto 1</p>
<p class="texto-bloque">Texto 2</p>
```

De 4 declaraciones duplicadas a 2 clases. Si hay que cambiar el navy, es un cambio en un lugar.

Ejercicio 10

¿Que está mal en este CSS y como lo arreglas sin usar !important?

```
#contenido p { color: black; }  
.destacado { color: orange; }
```

```
<div id="contenido"><p class="destacado">Oferta</p></div>
```

El párrafo se ve negro pero debería verse naranja.

SOLUCIÓN: #contenido p tiene 101 puntos (id + etiqueta) y le gana a .destacado (10 puntos). Sin !important, igualas o superas la especificidad de forma limpia:

```
#contenido p.destacado { color: orange; }
```

Este selector tiene 111 puntos (id + etiqueta + clase) y gana. Alternativa arquitectonica: evitar ids tan específicos en el CSS base para no tener que pelear contra ellos después.

Ampliación Unidad 2: casos de estudio y más práctica

Caso de estudio 1: paleta de marca completa

Un cliente te pide "los colores de mi marca en toda la web". En vez de regar hexadecimales, defines la paleta una vez y la documentas:

```
:root {  
  --marca-900: #0f1b2d; /* azul noche: encabezados, footer */  
  --marca-600: #1a3a5c; /* azul medio: botones primarios */  
  --marca-100: #e8f0f8; /* azul niebla: fondos suaves */  
  --acento-500: #e94560; /* coral: llamadas a la accion */  
  --acento-100: #fdeef1; /* coral palido: fondos de alerta */  
  --texto: #2d2d2d;  
  --texto-suave: #6b7280;  
}
```

Cada color tiene un rol documentado en el comentario. Cuando el diseño crece a 30 componentes, nadie inventa un azul nuevo: usa la escala. Así se construyen los sistemas de diseño.

Caso de estudio 2: texto sobre foto, el problema clásico

Tienes un hero con foto de fondo y texto blanco. En algunas fotos el texto se pierde en las zonas claras. Tres soluciones profesionales, de simple a robusta:

```
/* Opcion 1: sombra de texto */
.hero h1 {
  color: white;
  text-shadow: 0 2px 8px rgba(0, 0, 0, 0.7);
}

/* Opcion 2: overlay degradado sobre la imagen */
.hero {
  background:
    linear-gradient(rgba(0, 0, 0, 0.55), rgba(0, 0, 0, 0.55)),
    url("foto.jpg") center/cover no-repeat;
}

/* Opcion 3: las dos combinadas para fotos muy claras */
```

La opción 2 es la más usada: oscurece toda la imagen de forma pareja y el texto blanco siempre se lee. La opción 1 sirve para títulos grandes sueltos.

Caso de estudio 3: escala tipografica con ritmo vertical

Un blog se ve "desordenado" aunque cada elemento por separado está bien. El problema: espaciados y tamaños sin sistema. La solución es una escala:

```
:root { font-size: 16px; }

h1 { font-size: 2.5rem; line-height: 1.2; margin: 0 0 1.5rem 0; }
h2 { font-size: 1.75rem; line-height: 1.3; margin: 2.5rem 0 1rem 0; }
h3 { font-size: 1.25rem; line-height: 1.4; margin: 2rem 0 0.75rem 0; }
p { font-size: 1rem; line-height: 1.8; margin: 0 0 1.25rem 0; }
small { font-size: 0.875rem; }
```

Los márgenes inferiores crean un ritmo vertical constante: el ojo recorre la página con cadencia. Los títulos con line-height menor (1.2-1.4) se ven compactos y con autoridad; los párrafos con 1.8 respiran. Este sistema de 6 líneas arregla el 80% de los blogs "feos".

Más práctica

Ejercicio 7

Crea un "badge" circular con número de notificaciones: fondo rojo #e94560, texto blanco, 24px de diametro, texto centrado de 12px en negritas.

SOLUCIÓN:

```
.badge {  
  display: inline-flex;  
  justify-content: center;  
  align-items: center;  
  width: 24px;  
  height: 24px;  
  border-radius: 50%;  
  background-color: #e94560;  
  color: white;  
  font-size: 12px;  
  font-weight: bold;  
  font-family: Arial, sans-serif;  
}
```

El inline-flex con centrado en ambos ejes es la forma moderna de centrar texto en círculos pequeños. El border-radius: 50% con width y height iguales garantiza el círculo.

Ejercicio 8

Escribe un degradado que simule un atardecer (de #ff9a56 a #ff5e7e a #6a5acd, diagonal) y aplícalo como fondo de un banner con texto blanco.

SOLUCIÓN:

```
.atardecer {
  background: linear-gradient(135deg, #ff9a56 0%, #ff5e7e 50%, #6a5acd 100%);
  color: white;
  padding: 70px 30px;
  text-align: center;
  font-family: Arial, sans-serif;
}

.atardecer h2 {
  font-size: 36px;
  text-shadow: 0 2px 6px rgba(0, 0, 0, 0.35);
  margin: 0 0 10px 0;
}
```

Las paradas de color (0%, 50%, 100%) controlan donde ocurre cada transición. El text-shadow protege la legibilidad sobre el degradado.

Ejercicio 9

¿Que está mal aquí y por que la imagen no se ve?

```
.banner {
  background-image: url("img/banner.jpg");
  background-color: #222;
  height: 300px;
}
```

El archivo CSS está en `css/estilos.css` y la imagen en `img/banner.jpg`, ambos dentro de la misma carpeta del proyecto.

SOLUCIÓN: La ruta se resuelve relativa al CSS, no al HTML. Desde `css/estilos.css`, para llegar a `img/banner.jpg` hay que subir un nivel: `url("../img/banner.jpg")`. Sin el `../`, el navegador busca `css/img/banner.jpg`, que no existe. El `background-color: #222` actúa como respaldo y por eso se ve el fondo oscuro pero no la imagen.

Ejercicio 10

Disena la tipografía de una tarjeta de testimonio: cita en serif itálica de 20px con interlineado 1.6, autor en sans-serif de 13px en mayúsculas con 2px de espaciado entre letras.

SOLUCIÓN:

```
.testimonio blockquote {
  font-family: Georgia, serif;
  font-style: italic;
  font-size: 20px;
  line-height: 1.6;
  color: #2c3e50;
  margin: 0 0 15px 0;
}

.testimonio cite {
  font-family: Arial, sans-serif;
  font-style: normal;
  font-size: 13px;
  text-transform: uppercase;
  letter-spacing: 2px;
  color: #888;
}
```

El contraste entre la cita expresiva (serif itálica grande) y el autor discreto (mayúsculas pequeñas espaciadas) crea jerarquía editorial. El `font-style: normal` en `cite` quita la itálica que el navegador le pone por defecto.

Ampliación Unidad 3: casos de estudio y más práctica

Caso de estudio 1: el misterio de las dos columnas que no caben

Un desarrollador tiene un contenedor de 960px con dos columnas de 480px cada una. Agrega `padding: 20px` a cada columna "para que el texto respire" y la segunda columna se cae abajo. Diagnóstico: sin `border-box`, cada columna mide $480 + 40 = 520\text{px}$; juntas son $1040\text{px} > 960\text{px}$. Soluciones posibles: 1) `box-sizing: border-box` global (la correcta y permanente), 2) reducir el `width` a 440px (frágil: cualquier cambio futuro lo rompe), 3) quitar el `padding` (empeora el diseño). La lección: los síntomas de layout casi siempre son el modelo de caja; la cura es `border-box` desde el inicio del proyecto.

Caso de estudio 2: ritmo vertical con una sola unidad

Un diseño se ve "apretado" en unas secciones y "aireado" en otras sin patrón. La solución profesional es una escala de espaciado basada en múltiplos de 8:

```

:root {
  --s1: 8px;
  --s2: 16px;
  --s3: 24px;
  --s4: 32px;
  --s5: 48px;
  --s6: 64px;
}

.seccion { padding: var(--s6) var(--s2); }
.tarjeta { padding: var(--s3); margin-bottom: var(--s3); }
.titulo { margin-bottom: var(--s2); }

```

Todos los espacios del sitio son multiples de 8. El resultado se siente ordenado aunque el visitante no sepa por que. Los sistemas de diseño de Google, Apple y todos los frameworks usan este principio.

Caso de estudio 3: bordes que comunican estado

En una app de tareas, cada tarjeta usa su borde izquierdo para comunicar prioridad, sin texto extra:

```

.tarea { border-left: 5px solid #ccc; padding: 15px; background: white; }
.tarea.alta { border-left-color: #e94560; }
.tarea.media { border-left-color: #f4a261; }
.tarea.baja { border-left-color: #2a9d8f; }
.tarea.completada { border-left-color: #ccc; opacity: 0.6; }

```

De un vistazo sabes que es urgente y que está hecho. Los bordes no son decoración: son un canal de comunicacion visual. Este patron aparece en paneles de administracion, gestores de tareas y bandejas de entrada.

Más práctica

Ejercicio 7

Calcula el espacio total horizontal que ocupa esta caja en la página (incluyendo margin), con y sin border-box:

```
.caja {  
  width: 500px;  
  padding: 30px;  
  border: 10px solid black;  
  margin: 20px;  
}
```

SOLUCIÓN: Sin border-box: contenido 500 + padding 60 + borde 20 = 580px de caja visible; más margin 40 = 620px de espacio total ocupado. Con border-box: la caja visible mide exactamente 500px (el contenido se encoge a $500 - 60 - 20 = 420$ px); más margin 40 = 540px de espacio total. El margin nunca entra en el width, con ningún box-sizing.

Ejercicio 8

Crea un "chip" o etiqueta pequena estilo red social: padding 6px 14px, fondo #eef2ff, texto #4a6cf7 de 13px, bordes totalmente redondeados (píldora).

SOLUCIÓN:

```
.chip {  
  display: inline-block;  
  padding: 6px 14px;  
  background-color: #eef2ff;  
  color: #4a6cf7;  
  font-size: 13px;  
  font-weight: 600;  
  font-family: Arial, sans-serif;  
  border-radius: 999px;  
}
```

El border-radius: 999px (un valor exageradamente grande) garantiza la forma de píldora sin importar la altura del elemento. Es más robusto que calcular la mitad exacta.

Ejercicio 9

Explica por que este código centra el div en escritorio pero lo rompe en celular, y corrígelo:

```
.panel {  
  width: 900px;  
  margin: 0 auto;  
  padding: 30px;  
}
```

SOLUCIÓN: En celular (viewport de 360px), el width fijo de 900px desborda y aparece scroll horizontal. Correccion con max-width:

```
.panel {  
  width: 100%;  
  max-width: 900px;  
  margin: 0 auto;  
  padding: 30px;  
  box-sizing: border-box;  
}
```

Ahora en el celular ocupa el 100% y en escritorio se queda en 900px centrado. El border-box evita que el padding de 30px sume otros 60px al ancho.

Ejercicio 10

Disena un divisor elegante entre secciones usando solo un div vacío con clase "divisor": línea de 1px gris claro con un diamante centrado encima.

SOLUCIÓN: El diamante se hace con un cuadrado rotado 45 grados usando un pseudo-elemento:

```

.divisor {
  position: relative;
  height: 1px;
  background-color: #ddd;
  margin: 50px auto;
  max-width: 800px;
}

.divisor::after {
  content: "";
  position: absolute;
  top: -5px;
  left: 50%;
  width: 10px;
  height: 10px;
  background-color: white;
  border: 1px solid #ddd;
  transform: translateX(-50%) rotate(45deg);
}

```

El `::after` crea el diamante: un cuadrado de 10px rotado 45 grados, centrado con `left 50% + translateX(-50%)`, con fondo blanco para "cortar" la línea. El `top` negativo lo centra verticalmente sobre la línea de 1px.

Ampliación Unidad 4: casos de estudio y más práctica

Caso de estudio 1: el menú fijo que se come el contenido

Un sitio con menú `position: fixed` de 64px: al cargar, el título principal queda escondido detrás del menú. El desarrollador "arregla" poniendo `margin-top: 64px` al título, pero en el celular el menú crece a 110px (dos líneas) y el problema regresa. Solución robusta: usar una variable para la altura y compensar con ella:

```
:root { --altura-menu: 64px; }

.menu { position: fixed; top: 0; height: var(--altura-menu); }

.contenido { padding-top: var(--altura-menu); }

@media (max-width: 600px) {
  :root { --altura-menu: 110px; }
}
```

Al cambiar la variable en la media query, la compensacion se ajusta sola. Moraleja: los valores magicos duplicados se desincronizan; las variables no.

Caso de estudio 2: modal accesible paso a paso

Un modal mal hecho: se abre pero el usuario de teclado no puede cerrarlo y el fondo sigue siendo clicable. Versión profesional:

```

.overlay {
  position: fixed;
  inset: 0; /* atajo para top/right/bottom/left: 0 */
  background: rgba(0, 0, 0, 0.6);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 1000;
}

.modal {
  background: white;
  border-radius: 14px;
  padding: 30px;
  width: 90%;
  max-width: 480px;
  max-height: 90vh;
  overflow-y: auto;
  position: relative;
}

.cerrar {
  position: absolute;
  top: 12px;
  right: 12px;
  background: none;
  border: none;
  font-size: 22px;
  cursor: pointer;
}

```

El `inset: 0` cubre toda la pantalla. El overlay usa flex para centrar el modal (matando dos pajaros: overlay + centrado). El `max-height: 90vh` con `overflow-y: auto` evita que un modal largo se salga de la pantalla en celulares. El botón cerrar va absolute en la esquina. (El manejo del foco con teclado es trabajo de JavaScript, pero el CSS ya deja la base lista.)

Caso de estudio 3: de floats a flexbox, la migración

Heredas un layout de 2012 hecho con floats y clearfixes:

```
/* Código viejo */  
.columna { float: left; width: 33.33%; }  
.fila::after { content: ""; display: table; clear: both; }
```

Lo migras a flexbox manteniendo el mismo HTML:

```
/* Código nuevo */  
.fila { display: flex; gap: 20px; }  
.columna { flex: 1; }
```

Menos código, sin clearfix, con gap real entre columnas y alturas igualadas automáticamente. La migración de floats a flex/grid es uno de los trabajos de modernización más comunes en proyectos reales.

Más práctica

Ejercicio 7

Crea un banner de cookies fijo abajo que ocupe todo el ancho, con fondo oscuro semitransparente y texto blanco, que quede sobre todo el contenido.

SOLUCIÓN:

```

.cookies {
  position: fixed;
  bottom: 0;
  left: 0;
  width: 100%;
  box-sizing: border-box;
  background-color: rgba(20, 20, 30, 0.95);
  color: white;
  padding: 18px 25px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  gap: 20px;
  z-index: 2000;
  font-family: Arial, sans-serif;
  font-size: 14px;
}

.cookies button {
  background-color: #e94560;
  color: white;
  border: none;
  padding: 10px 25px;
  border-radius: 6px;
  cursor: pointer;
  white-space: nowrap;
}

```

El z-index alto (2000) lo pone sobre modales normales. El `white-space: nowrap` evita que el botón se parta en dos líneas en el celular.

Ejercicio 8

Explica la diferencia práctica entre estos dos y cuando usarías cada uno:

```

.a { display: none; }
.b { visibility: hidden; }

```

SOLUCIÓN: `.a` desaparece por completo: no ocupa espacio y el layout se reorganiza como si el elemento no existiera. Se usa para mostrar/ocultar paneles, pestanas y menues (típicamente con JavaScript alternando la clase). `.b` se vuelve invisible pero conserva su espacio: queda un hueco vacío. Se usa cuando quieres ocultar algo sin que

el layout brinque, por ejemplo, ocultar un icono que luego apareciera, o en animaciones donde el espacio debe reservarse.

Ejercicio 9

Posiciona un tooltip encima de un botón, centrado horizontalmente, que aparezca al hover. El tooltip no debe mover nada del layout.

SOLUCIÓN:

```
.boton-wrap { position: relative; display: inline-block; }

.tooltip {
  position: absolute;
  bottom: 120%;
  left: 50%;
  transform: translateX(-50%);
  background-color: #1a1a2e;
  color: white;
  padding: 8px 14px;
  border-radius: 6px;
  font-size: 13px;
  white-space: nowrap;
  opacity: 0;
  pointer-events: none;
  transition: opacity 0.25s;
}

.boton-wrap:hover .tooltip { opacity: 1; }
```

El `absolute` lo saca del flujo (no mueve nada). `bottom: 120%` lo pone encima del botón. El `opacity` con transición permite aparecer suavemente (`display` no se puede animar). El `pointer-events: none` evita que el tooltip intercepte el mouse y cause parpadeo.

Ejercicio 10

¿Que hace `inset: 0` y a que equivale en propiedades individuales?

SOLUCIÓN: `inset` es la abreviatura de los cuatro desplazamientos de posicionamiento. `inset: 0` equivale a `top: 0; right: 0; bottom: 0; left: 0;`. En un elemento `fixed` o `absolute`, esto lo estira para cubrir por completo su contenedor de referencia. También acepta 2, 3 o 4 valores como `margin/padding` (vertical/horizontal, etc.).

Ampliación Unidad 5: casos de estudio y más práctica

Caso de estudio 1: la barra de app para celular con flexbox

Necesitas la típica barra inferior de app para celular: 4 iconos repartidos equitativamente, cada uno con etiqueta. Con flexbox son 10 líneas:

```
.tabbar {  
  display: flex;  
  position: fixed;  
  bottom: 0;  
  left: 0;  
  right: 0;  
  background: white;  
  border-top: 1px solid #eee;  
  padding: 8px 0 calc(8px + env(safe-area-inset-bottom));  
}  
  
.tab {  
  flex: 1;  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  gap: 4px;  
  color: #888;  
  font-size: 11px;  
  font-family: Arial, sans-serif;  
}  
  
.tab.activo { color: #e94560; }  
.tab .icono { font-size: 22px; }
```

Cada `.tab` es a su vez un flex en columna: icono arriba, etiqueta abajo, centrados. El `env(safe-area-inset-bottom)` respeta la zona segura de los iPhone con notch. Este componente existe en millones de apps y es puro flexbox anidado.

Caso de estudio 2: Holy Grail con grid moderno

El "Holy Grail layout" (encabezado, 3 columnas, pie) fue el problema famoso de los 2000 que requería trucos oscuros. Con grid son 15 líneas:

```
.layout {
  display: grid;
  grid-template:
    "header header header" auto
    "nav main aside" 1fr
    "footer footer footer" auto
    / 200px 1fr 220px;
  min-height: 100vh;
  gap: 15px;
}
```

```
.header { grid-area: header; }
.nav { grid-area: nav; }
.main { grid-area: main; }
.aside { grid-area: aside; }
.footer { grid-area: footer; }
```

La propiedad `grid-template` combina áreas, filas y columnas en una sola declaración compacta. Lo que tomaba 50 líneas de floats y hacks ahora es declarativo y legible.

Caso de estudio 3: cuando flexbox no es la respuesta

Un desarrollador intenta hacer una tabla de datos con flexbox y sufre: las columnas no se alinean entre filas porque cada fila calcula sus anchos por separado. La solución es reconocer que es un problema bidimensional y usar grid:

```
/* Mal: flexbox por fila, columnas desalineadas */
.fila { display: flex; }
.fila span { flex: 1; }

/* Bien: grid con columnas compartidas */
.tabla { display: grid; grid-template-columns: 2fr 1fr 1fr 1fr; }
```

En grid, todas las filas comparten las mismas líneas de columna, así que todo se alinea automáticamente. Regla de decisión: ¿los elementos deben alinearse en dos dimensiones? Grid. ¿Es una distribución en una línea? Flexbox.

Más práctica

Ejercicio 7

Crea un encabezado de artículo con autor a la izquierda y fecha a la derecha, verticalmente centrados, usando flexbox.

SOLUCIÓN:

```
.meta-articulo {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 12px 0;
  border-top: 1px solid #eee;
  border-bottom: 1px solid #eee;
  font-family: Arial, sans-serif;
  font-size: 14px;
  color: #666;
}

.autor { display: flex; align-items: center; gap: 10px; }

.autor img {
  width: 36px;
  height: 36px;
  border-radius: 50%;
}
```

El `space-between` empuja autor y fecha a los extremos; el flex anidado en `.autor` alinea la foto con el nombre. Patrón típico de blogs y periódicos.

Ejercicio 8

Haz una fila de estadísticas donde el número central destaque ocupando el doble de espacio que los laterales.

SOLUCIÓN:

```
.stats { display: flex; gap: 15px; text-align: center; }

.stat { flex: 1; padding: 20px; background: #f8f9fa; border-radius: 10px; }

.stat.central { flex: 2; background: #1a1a2e; color: white; }

.stat strong { display: block; font-size: 28px; }
.stat.central strong { font-size: 40px; }
```

El `flex: 2` le da el doble de espacio sobrante que a sus hermanos con `flex: 1`. El `flex-grow` es proporcional, no absoluto: lo que importa es la relación entre los valores.

Ejercicio 9

Explica que vería el usuario con este código y por qué:

```
.caja {
  display: flex;
  flex-direction: column;
  justify-content: center;
  height: 300px;
}
```

SOLUCIÓN: Vería los hijos apilados verticalmente (`column`) y centrados verticalmente dentro de los 300px. Como la dirección es columna, el eje principal es vertical: `justify-content: center` centra en vertical. Horizontalmente, los hijos se estiran al ancho completo por el `align-items: stretch` implícito (valor por defecto). Es el patrón para centrar contenido apilado verticalmente en un hero o una tarjeta alta.

Ejercicio 10

Crea un layout de dashboard: barra lateral fija de 240px y contenido flexible, que en el celular (< 768px) la barra se oculte y el contenido ocupe todo.

SOLUCIÓN:

```
.dashboard { display: flex; min-height: 100vh; }

.sidebar {
  flex: 0 0 240px;
  background: #1a1a2e;
  color: white;
  padding: 25px;
}

.main { flex: 1; padding: 30px; background: #f5f6fa; }

@media (max-width: 768px) {
  .sidebar { display: none; }
}
```

El `flex: 0 0 240px` fija la barra en 240px sin crecer ni encogerse. En el celular se oculta (en un proyecto real, un botón hamburguesa la mostraría con JavaScript). El contenido con `flex: 1` siempre ocupa el resto.

Ampliación Unidad 6: casos de estudio y más práctica

Caso de estudio 1: menú desplegable con CSS puro

Un menú con submenús sin JavaScript, usando `hover` y posicionamiento:

```

.menu-principal { display: flex; list-style: none; padding: 0; background: #1a1a2e;
}

.menu-principal > li { position: relative; }

.menu-principal a {
  display: block;
  color: white;
  text-decoration: none;
  padding: 15px 25px;
  font-family: Arial, sans-serif;
}

.submenu {
  display: none;
  position: absolute;
  top: 100%;
  left: 0;
  list-style: none;
  padding: 0;
  margin: 0;
  background: #16213e;
  min-width: 200px;
  box-shadow: 0 10px 25px rgba(0,0,0,0.25);
}

.menu-principal > li:hover .submenu { display: block; }

.submenu a:hover { background: #e94560; }

```

El `li` padre es `relative` para que el `submenu absolute` se ancle debajo de el (`top: 100%`). `El: hover` en el `li` muestra el submenu. Limitacion real: en pantallas tactiles no hay hover, asi que en producción esto se complementa con JavaScript; pero para escritorio funciona perfecto.

Caso de estudio 2: pasos de un proceso con contadores CSS

Una línea de pasos (1. Registro, 2. Pago, 3. Confirmacion) sin números escritos a mano, usando `counter`:

```

.pasos { list-style: none; padding: 0; display: flex; counter-reset: paso; }

.pasos li {
  flex: 1;
  counter-increment: paso;
  position: relative;
  padding-top: 45px;
  text-align: center;
  font-family: Arial, sans-serif;
}

.pasos li::before {
  content: counter(paso);
  position: absolute;
  top: 0;
  left: 50%;
  transform: translateX(-50%);
  width: 32px;
  height: 32px;
  border-radius: 50%;
  background: #ddd;
  display: flex;
  justify-content: center;
  align-items: center;
  font-weight: bold;
}

.pasos li.completado::before { background: #2a9d8f; color: white; }
.pasos li.actual::before { background: #e94560; color: white; }

```

counter-reset inicia el contador, counter-increment lo aumenta por cada li, y content: counter(paso) lo muestra. Si reordenas o agregas pasos en el HTML, los números se actualizan solos. Las clases completado / actual pintan el estado de cada paso.

Caso de estudio 3: tabla con columnas fijas y zebra

Una tabla ancha que necesita scroll horizontal pero con la primera columna siempre visible (como las hojas de calculo):

```

.tabla-scroll { overflow-x: auto; max-width: 100%; }

.tabla-datos { border-collapse: collapse; min-width: 800px; }

.tabla-datos th:first-child,
.tabla-datos td:first-child {
  position: sticky;
  left: 0;
  background: white;
  z-index: 2;
  box-shadow: 2px 0 5px rgba(0,0,0,0.08);
}

.tabla-datos thead th {
  position: sticky;
  top: 0;
  background: #1a1a2e;
  color: white;
  z-index: 3;
}

```

El sticky en dos ejes: la primera columna se pega a la izquierda y el encabezado arriba. La celda esquina (th primera) necesita z-index mayor y fondo sólido para que el contenido no se transparente. Es un componente de nivel avanzado que combina todo lo de la unidad.

Más práctica

Ejercicio 7

Sin tocar el HTML, numera automáticamente los items de una lista con clase "ranking" como "1.", "2.", "3." en negritas color #e94560.

SOLUCIÓN:

```

.ranking { list-style: none; padding: 0; counter-reset: lugar; }

.ranking li { counter-increment: lugar; padding: 8px 0; }

.ranking li::before {
  content: counter(lugar) ". ";
  font-weight: bold;
  color: #e94560;
  margin-right: 8px;
}

```

Los contadores CSS generan la numeración automáticamente. Si agregas un ítem en medio, todo se reenumera solo.

Ejercicio 8

Crea un enlace "Leer más" que muestre una flecha -> después del texto, y que la flecha se desplace 5px a la derecha al hover, con transición.

SOLUCIÓN:

```

.leer-mas {
  color: #e94560;
  text-decoration: none;
  font-weight: bold;
  font-family: Arial, sans-serif;
}

.leer-mas::after {
  content: " ->";
  display: inline-block;
  transition: transform 0.25s ease;
}

.leer-mas:hover::after {
  transform: translateX(5px);
}

```

El `::after` crea la flecha como elemento animable (`inline-block` para que acepte `transform`). Al `hover`, solo la flecha se mueve: un micro-detalle que invita al clic.

Ejercicio 9

Explica que selecciona cada uno y da un caso de uso:

- a) `li:not(:last-child)`
- b) `input:checked + label`
- c) `p:empty`

SOLUCIÓN: a) Todos los `li` excepto el último. Caso de uso: poner separadores entre items de una lista sin que el último tenga borde inferior: `li:not(:last-child) { border-bottom: 1px solid #eee; }`. b) El `label` que sigue inmediatamente a un `input` marcado. Caso de uso: estilizar la etiqueta del checkbox seleccionado, base de interruptores personalizados. c) Párrafos sin contenido (ni siquiera espacios). Caso de uso: ocultar párrafos vacíos que un CMS genera por error: `p:empty {display: none;}`.

Ejercicio 10

Haz una cuadrícula de 4 columnas donde cada celda par tenga fondo gris y la celda número 5 destaque ocupando 2 columnas, todo sin clases extra en las celdas.

SOLUCIÓN:

```
.rejilla {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  gap: 12px;
}

.rejilla > div:nth-child(even) { background: #f0f0f0; }

.rejilla > div:nth-child(5) {
  grid-column: span 2;
  background: #e94560;
  color: white;
}
```

El `nth-child(even)` cebra las celdas y el `nth-child(5)` selecciona la quinta para destacarla. Todo sin tocar el HTML: el CSS hace el trabajo pesado.

Ampliación Unidad 7: casos de estudio y más práctica

Caso de estudio 1: skeleton loading, el estandar moderno

Mientras carga el contenido, las apps muestran "esqueletos" grises que pulsan. Así se hacen con CSS puro:

```
.skeleton {  
  background: linear-gradient(90deg, #eee 25%, #ddd 50%, #eee 75%);  
  background-size: 200% 100%;  
  animation: brillo 1.5s linear infinite;  
  border-radius: 8px;  
}  
  
.skeleton.texto { height: 16px; margin: 10px 0; }  
.skeleton.titulo { height: 28px; width: 60%; }  
.skeleton.avatar { width: 60px; height: 60px; border-radius: 50%; }  
  
@keyframes brillo {  
  from { background-position: 200% 0; }  
  to { background-position: -200% 0; }  
}
```

El degradado se mueve horizontalmente creando la onda de brillo que ves en Facebook, YouTube y LinkedIn. El `background-size: 200%` permite que el degradado viaje. Es preferible a un spinner porque el usuario percibe la forma del contenido que viene.

Caso de estudio 2: acordeon animado sin JavaScript (truco checkbox)

Un acordeon de preguntas frecuentes usando `:checked`:

```

.acordeon input { display: none; }

.acordeon .respuesta {
  max-height: 0;
  overflow: hidden;
  transition: max-height 0.4s ease;
}

.acordeon input:checked ~ .respuesta { max-height: 300px; }

.acordeon label {
  display: block;
  padding: 15px;
  background: #f5f5f5;
  cursor: pointer;
  font-weight: bold;
}

```

```

<div class="acordeon">
  <input type="checkbox" id="p1">
  <label for="p1">¿Los cursos son gratis?</label>
  <div class="respuesta"><p>Si, todos...</p></div>
</div>

```

El checkbox oculto guarda el estado; el `label` lo alterna al clic; `:checked ~ .respuesta` expande la respuesta animando `max-height` (height: auto no se puede animar, por eso el truco del `max-height` generoso). Es el patron "checkbox hack" que sostiene muchos componentes CSS puros.

Caso de estudio 3: animación de entrada para toda la página

Efecto de "revelado" al cargar, con cada sección entrando en cascada:

```

.revelar {
  opacity: 0;
  transform: translateY(40px);
  animation: revelar 0.8s ease forwards;
}

@keyframes revelar {
  to { opacity: 1; transform: translateY(0); }
}

header.revelar { animation-delay: 0.1s; }
section.revelar:nth-of-type(1) { animation-delay: 0.25s; }
section.revelar:nth-of-type(2) { animation-delay: 0.4s; }
footer.revelar { animation-delay: 0.55s; }

@media (prefers-reduced-motion: reduce) {
  .revelar { animation: none; opacity: 1; transform: none; }
}

```

Cada bloque entra desde abajo con retardo progresivo. El `forwards` mantiene el estado final. Y el bloque de `prefers-reduced-motion` respeta a usuarios sensibles al movimiento: detalle profesional obligatorio.

Más práctica

Ejercicio 7

Crea un botón que al `hover` gire 360 grados en 0.6 segundos y al mismo tiempo cambie de color. Al quitar el mouse debe regresar suavemente.

SOLUCIÓN:

```

.btn-gira {
  padding: 14px 30px;
  background: #1a1a2e;
  color: white;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  transition: transform 0.6s ease, background-color 0.6s ease;
}

.btn-gira:hover {
  transform: rotate(360deg);
  background: #e94560;
}

```

Como la transición está en el estado base, el regreso también se anima (gira de vuelta). Girar 360 grados termina en la misma posición visual, así que no hay salto.

Ejercicio 8

Haz una barra de progreso que se llene de 0 a 100% en 3 segundos al cargar la página y se quede llena.

SOLUCIÓN:

```

.barra { width: 100%; height: 12px; background: #eee; border-radius: 6px; }

.relleno {
  height: 100%;
  width: 0;
  border-radius: 6px;
  background: linear-gradient(to right, #2a9d8f, #7bd389);
  animation: llenar 3s ease forwards;
}

@keyframes llenar {
  from { width: 0; }
  to { width: 100%; }
}

```

El forwards es esencial: sin el, la barra regresaría a 0 al terminar. (Animar width aquí es aceptable porque es un elemento aislado y pequeño; en listas largas preferiría transform: scaleX.)

Ejercicio 9

Explica por que esta transición "no funciona" al cargar la página:

```
.caja {  
  opacity: 0;  
  transition: opacity 1s;  
}  
.caja.visible { opacity: 1; }
```

La clase "visible" se agrega con JavaScript un segundo después de cargar.

SOLUCIÓN: En realidad SI funciona: al agregar la clase, la opacidad cambia de 0 a 1 y la transición lo suaviza durante 1s. La confusión común es esperar que la transición se ejecute "sola" al cargar sin que nada cambie. Las transiciones necesitan un cambio de valor disparado por algo (una clase, un hover). Si el JavaScript agrega la clase correctamente, el fundido aparece. Si no se ve, el problema suele ser que la clase se agrega antes de que el navegador pinte el estado inicial (solución: forzar un reflow o usar requestAnimationFrame).

Ejercicio 10

Crea el efecto de "latido" para un botón de emergencia: escala pulsante infinita de 1 a 1.08, con un anillo que se expande y desvanece.

SOLUCIÓN:

```

.btn-emergencia {
  position: relative;
  padding: 16px 45px;
  background: #e94560;
  color: white;
  border: none;
  border-radius: 50px;
  font-size: 18px;
  font-weight: bold;
  cursor: pointer;
  animation: latido 1.6s ease-in-out infinite;
}

@keyframes latido {
  0%, 100% { transform: scale(1); }
  15% { transform: scale(1.08); }
  30% { transform: scale(1); }
}

.btn-emergencia::after {
  content: "";
  position: absolute;
  inset: 0;
  border-radius: 50px;
  border: 3px solid #e94560;
  animation: anillo 1.6s ease-out infinite;
}

@keyframes anillo {
  from { transform: scale(1); opacity: 0.8; }
  to { transform: scale(1.5); opacity: 0; }
}

```

Dos animaciones sincronizadas (mismo 1.6s): el botón late y el pseudo-elemento proyecta un anillo que se expande y desvanece. El `inset: 0` hace que el anillo parta del borde del botón.

Ampliación Unidad 8: casos de estudio y más práctica

Caso de estudio 1: tarjeta de credito visual

Un formulario de pago con vista previa de la tarjeta que se actualiza (la actualizacion es JS, pero el CSS de la tarjeta es la estrella):

```
.tarjeta-credito {
  width: 340px;
  height: 200px;
  border-radius: 16px;
  background: linear-gradient(135deg, #1a1a2e 0%, #3a3a6e 50%, #16213e 100%);
  color: white;
  padding: 22px;
  font-family: "Courier New", monospace;
  box-shadow: 0 15px 35px rgba(0, 0, 0, 0.35);
  position: relative;
  overflow: hidden;
}

.tarjeta-credito::before {
  content: "";
  position: absolute;
  width: 300px;
  height: 300px;
  border-radius: 50%;
  background: radial-gradient(circle, rgba(255,255,255,0.12) 0%, transparent 70%);
  top: -100px;
  right: -80px;
}

.chip-tarjeta {
  width: 48px;
  height: 36px;
  border-radius: 6px;
  background: linear-gradient(135deg, #ffd93d, #f4a261);
  margin-bottom: 25px;
}

.numero-tarjeta { font-size: 20px; letter-spacing: 3px; margin-bottom: 20px; }

.datos-tarjeta { display: flex; justify-content: space-between; font-size: 12px; }
.datos-tarjeta span { display: block; opacity: 0.7; font-size: 10px; text-transform: uppercase; }
```

Degradado oscuro, brillo radial decorativo con::before, chip dorado con degradado, números monoespaciados con letter-spacing. Es el componente "wow" de los checkouts modernos, 100% CSS.

Caso de estudio 2: tabla de comparacion de planes

```
<table class="comparar">
  <thead>
    <tr><th></th><th>Basico</th><th
class="col-destacada">Pro</th><th>Empresas</th></tr>
  </thead>
  <tbody>
    <tr><td>Cursos</td><td>10</td><td
class="col-destacada">Ilimitados</td><td>Ilimitados</td></tr>
    <tr><td>Certificados</td><td
class="col-destacada">[ok]</td><td>[ok]</td></tr>
    <tr><td>Soporte</td><td
class="col-destacada">[ok]</td><td>Prioritario</td></tr>
  </tbody>
</table>
```

```
.comparar { border-collapse: collapse; width: 100%; max-width: 750px; font-family:
Arial, sans-serif; }

.comparar th, .comparar td { padding: 14px; text-align: center; border-bottom: 1px
solid #eee; }

.comparar td:first-child { text-align: left; font-weight: bold; }

.comparar .col-destacada { background: #fef2f4; }

.comparar thead .col-destacada {
  background: #e94560;
  color: white;
  border-radius: 10px 10px 0 0;
}

.comparar .no { color: #ccc; }
```

La columna destacada usa una clase aplicada a cada celda de la columna (no hay selector de columna nativo práctico). El encabezado de la columna popular va en color de acento: el ojo va directo al plan que quieres vender.

Caso de estudio 3: validación visual completa de formulario

Combinando pseudoclasas de validación con mensajes de ayuda:

```
.campo-wrap { margin-bottom: 20px; }

.campo-wrap input {
  width: 100%;
  box-sizing: border-box;
  padding: 13px 15px;
  border: 2px solid #ddd;
  border-radius: 10px;
  font-size: 16px;
  transition: border-color 0.25s;
}

.campo-wrap input:focus { outline: none; border-color: #4a90d9; }

.campo-wrap input:not(:placeholder-shown):valid { border-color: #2a9d8f; }

.campo-wrap input:not(:placeholder-shown):invalid { border-color: #e94560; }

.mensaje { font-size: 13px; margin-top: 6px; display: none; }

.campo-wrap input:not(:placeholder-shown):invalid ~ .mensaje.error { display: block; color: #e94560; }
.campo-wrap input:not(:placeholder-shown):valid ~ .mensaje.ok { display: block; color: #2a9d8f; }
```

El truco: `not(:placeholder-shown)` evita marcar los campos como invalidos antes de que el usuario escriba nada. Los mensajes aparecen solo después de la interacción. Validación visual completa sin JavaScript.

Más práctica

Ejercicio 7

Dale a un blockquote el estilo de "cita destacada": borde izquierdo grueso de acento, fondo suave, texto grande en serif itálica y comilla decorativa gigante con::before.

SOLUCIÓN:

```
.cita-destacada {  
  position: relative;  
  border-left: 5px solid #e94560;  
  background: #fdf2f4;  
  padding: 30px 30px 30px 70px;  
  font-family: Georgia, serif;  
  font-style: italic;  
  font-size: 22px;  
  line-height: 1.6;  
  color: #333;  
  border-radius: 0 12px 12px 0;  
  margin: 30px 0;  
}  
  
.cita-destacada::before {  
  content: "\201C";  
  position: absolute;  
  left: 18px;  
  top: 10px;  
  font-size: 60px;  
  color: #e94560;  
  opacity: 0.35;  
  font-style: normal;  
}
```

El "\201C" es el código de la comilla tipográfica de apertura (se escribe así en content para evitar problemas de encoding). La comilla gigante semitransparente es el sello visual de las citas editoriales.

Ejercicio 8

Crea una lista de pasos vertical con línea conectora a la izquierda y círculos numerados, usando solo CSS (sin imágenes).

SOLUCIÓN:

```

.pasos-v { list-style: none; padding: 0; margin: 0; counter-reset: paso; }

.pasos-v li {
  counter-increment: paso;
  position: relative;
  padding: 0 0 30px 50px;
  font-family: Arial, sans-serif;
}

.pasos-v li::before {
  content: counter(paso);
  position: absolute;
  left: 0;
  top: 0;
  width: 32px;
  height: 32px;
  border-radius: 50%;
  background: #1a1a2e;
  color: white;
  display: flex;
  justify-content: center;
  align-items: center;
  font-weight: bold;
  font-size: 14px;
}

.pasos-v li::after {
  content: "";
  position: absolute;
  left: 15px;
  top: 36px;
  bottom: 0;
  width: 2px;
  background: #ddd;
}

.pasos-v li:last-child::after { display: none; }

```

El `::before` dibuja el círculo numerado (con contador automático) y el `::after` la línea conectora. El último ítem no lleva línea (`:last-child::after`). Es la clásica *timeline vertical* de procesos.

Ejercicio 9

¿Que problema tiene este formulario en iPhone y como lo arreglas?

```
input { font-size: 13px; padding: 10px; }
```

SOLUCIÓN: iOS hace zoom automático al enfocar campos con font-size menor a 16px, lo que rompe el layout y confunde al usuario. Arreglo:

```
input, select, textarea {  
  font-size: 16px;  
  padding: 12px;  
}
```

16px es el mínimo seguro. Si el diseño pide texto más pequeño, se puede usar 16px igual y escalar visualmente otros elementos, o aceptar el zoom como comportamiento nativo.

Ejercicio 10

Crea un interruptor (toggle switch) estilo iOS usando un checkbox oculto y CSS puro.

SOLUCIÓN:

```

.switch input { display: none; }

.switch .riel {
  display: inline-block;
  width: 52px;
  height: 30px;
  border-radius: 15px;
  background: #ccc;
  position: relative;
  cursor: pointer;
  transition: background-color 0.3s;
}

.switch .riel::after {
  content: "";
  position: absolute;
  top: 3px;
  left: 3px;
  width: 24px;
  height: 24px;
  border-radius: 50%;
  background: white;
  box-shadow: 0 2px 5px rgba(0,0,0,0.25);
  transition: transform 0.3s;
}

.switch input:checked + .riel { background: #2a9d8f; }

.switch input:checked + .riel::after { transform: translateX(22px); }

```

```

<label class="switch">
  <input type="checkbox">
  <span class="riel"></span>
</label>

```

El checkbox guarda el estado, el .riel es la pista y su ::after la bolita. :checked + .riel pinta la pista en verde y mueve la bolita 22px. El + (hermano adyacente) conecta el estado con el estilo. Patron estandar de la industria.

Ampliación Unidad 9: casos de estudio y más práctica

Caso de estudio 1: imágenes responsive de verdad

max-width: 100% evita desbordes, pero para rendimiento real se usan imágenes distintas por tamaño con srcset (HTML) combinadas con CSS:

```

```

```
.img-responsive {
  max-width: 100%;
  height: auto;
  display: block;
  border-radius: 10px;
}
```

El navegador elige la imagen más adecuada según el ancho de pantalla y la densidad de píxeles: en el celular descarga la de 400px (rápida), en escritorio la de 1200px (nitida). El CSS garantiza que nunca desborde. Esta combinación HTML+CSS es el estándar de rendimiento en imágenes.

Caso de estudio 2: contenedores que se adaptan al contenido

Las Container Queries son la evolución de las media queries: en vez de responder al viewport, el componente responde al tamaño de SU contenedor:

```
.tarjeta-contenedor { container-type: inline-size; }

.info-producto { display: grid; grid-template-columns: 1fr; gap: 15px; }

@container (min-width: 500px) {
  .info-producto { grid-template-columns: 200px 1fr; }
}
```

La tarjeta muestra una columna cuando está en un espacio angosto (barra lateral) y dos columnas cuando tiene espacio (contenido principal), sin importar el tamaño de la pantalla. Es el futuro del responsive por componentes: el mismo componente se adapta a donde lo pongas.

Caso de estudio 3: sistema de espaciado y tipografía con variables

Un mini sistema de diseño completo en 20 líneas de variables:

```
:root {
  --f-u1: 0.75rem; --f-u2: 0.875rem; --f-base: 1rem;
  --f-u4: 1.25rem; --f-u5: 1.75rem; --f-u6: 2.5rem;
  --e-xs: 0.25rem; --e-s: 0.5rem; --e-m: 1rem;
  --e-l: 1.5rem; --e-xl: 2.5rem; --e-xxl: 4rem;
  --r-s: 6px; --r-m: 10px; --r-l: 16px; --r-full: 999px;
}

.titulo-pagina { font-size: var(--f-u6); margin-bottom: var(--e-l); }
.subtitulo { font-size: var(--f-u4); margin-bottom: var(--e-m); }
.texto { font-size: var(--f-base); line-height: 1.7; margin-bottom: var(--e-m); }
.btn { font-size: var(--f-u2); padding: var(--e-s) var(--e-l); border-radius:
var(--r-full); }
.card { padding: var(--e-l); border-radius: var(--r-l); margin-bottom: var(--e-l);
}
```

Con este sistema, un rediseño completo (tipografía más grande, espaciados más amplios) es cuestión de ajustar las variables, no de cazar valores por 2000 líneas de CSS. Así trabajan los equipos profesionales.

Más práctica

Ejercicio 7

Escribe los tres breakpoints típicos mobile-first para una rejilla que pasa de 1 a 2 a 3 a 4 columnas.

SOLUCIÓN:

```
.rejilla { display: grid; grid-template-columns: 1fr; gap: 15px; }

@media (min-width: 600px) { .rejilla { grid-template-columns: repeat(2, 1fr); } }
@media (min-width: 900px) { .rejilla { grid-template-columns: repeat(3, 1fr); } }
@media (min-width: 1200px) { .rejilla { grid-template-columns: repeat(4, 1fr); } }
```

Base para celular primero, mejoras progresivas ordenadas de menor a mayor. Cada breakpoint agrega una columna cuando hay espacio suficiente.

Ejercicio 8

Convierte estos tamaños fijos a un sistema relativo con rem (base 16px): título 32px, subtítulo 24px, texto 16px, pequeño 12px, padding 20px.

SOLUCIÓN:

```
.titulo { font-size: 2rem; }      /* 32px */
.subtitulo { font-size: 1.5rem; } /* 24px */
.texto { font-size: 1rem; }      /* 16px */
.pequeno { font-size: 0.75rem; } /* 12px */
.caja { padding: 1.25rem; }      /* 20px */
```

Divides entre 16. Si el usuario configura su navegador con fuente base más grande (accesibilidad), todo escala proporcionalmente: con px eso no pasaría.

Ejercicio 9

¿Que hace 100svh y en que se diferencia de 100vh?

SOLUCIÓN: 100vh es el 100% de la altura del viewport, pero en navegadores de celular el viewport incluye la zona detrás de las barras de navegación (que aparecen y desaparecen), causando saltos de layout. 100svh (small viewport height) usa la altura mínima estable, sin esos saltos. Para héroes de pantalla completa en el celular, 100svh (con 100vh como respaldo para navegadores viejos) es la práctica moderna.

Ejercicio 10

Crea un tema de alto contraste para accesibilidad usando variables, activado por una clase en el body.

SOLUCIÓN:

```

:root {
  --fondo: #ffffff;
  --texto: #333333;
  --enlace: #1a56db;
  --borde: #cccccc;
}

body { background: var(--fondo); color: var(--texto); }
a { color: var(--enlace); }

body.alto-contraste {
  --fondo: #000000;
  --texto: #ffffff;
  --enlace: #ffff00;
  --borde: #ffffff;
}

body.alto-contraste * {
  border-color: var(--borde);
}

```

Un interruptor de accesibilidad (un botón que alterna la clase con JavaScript) cambia todo el sitio a máximo contraste para usuarios con baja visión. Las variables hacen que sea trivial; sin ellas sería reescribir cientos de reglas.

Ampliación Unidad 10: casos de estudio y más práctica

Caso de estudio 1: checklist de lanzamiento

Antes de entregar un proyecto, los profesionales revisan esta lista. Cada punto viene de un error real que costó dinero:

1. [] Probado en Chrome, Firefox, Safari y Edge
2. [] Probado en celular real (no solo redimensionando)
3. [] Sin scroll horizontal en 360px de ancho
4. [] Imagenes con alt descriptivo y max-width: 100%
5. [] Contraste de texto verificado (herramienta de DevTools)
6. [] Foco visible en todos los elementos interactivos
7. [] Formularios validan y muestran errores claros
8. [] Tiempos de carga: imagenes optimizadas
9. [] prefers-reduced-motion implementado
10. [] Sin !important injustificados en el CSS
11. [] Sin estilos en linea (salvo generados por JS)
12. [] Variables usadas consistentemente, sin valores magicos

Imprimela y usala en cada proyecto. La diferencia entre junior y senior muchas veces es solo la disciplina del checklist.

Caso de estudio 2: portafolio personal, estructura completa

El proyecto que todo estudiante debe construir al graduarse: su portafolio. Estructura recomendada con las tecnicas del curso:

- Hero (100svh, flex centrado, clamp en el titulo, CTA)
- Sobre mi (2 columnas en desktop con grid, 1 en el celular)
- Proyectos (rejilla auto-fit, tarjetas con hover de elevacion)
- Habilidades (barras de progreso animadas con keyframes)
- Testimonios (carrusel simple o rejilla)
- Contacto (formulario estilizado de la Unidad 8)
- Footer (oscuro, enlaces, redes)

Las barras de habilidades son un buen ejercicio final:

```
.barra { background: #eee; border-radius: 10px; height: 12px; overflow: hidden; }

.barra span {
  display: block;
  height: 100%;
  border-radius: 10px;
  background: linear-gradient(to right, #e94560, #ff7e5f);
  width: 0;
  animation: llenar 1.5s ease forwards;
}

@keyframes llenar { to { width: var(--nivel); } }
```

```
<div class="barra"><span style="--nivel: 90%;"></span></div>
```

Cada barra define su nivel con una variable inline (uno de los pocos usos legítimos de style en línea: valores dinámicos). La animación las llena al cargar.

Caso de estudio 3: cuanto cobrar por un sitio

Pregunta frecuente al graduarse. No hay tarifa única, pero este marco ayuda: calcula por valor y complejidad, no por horas a ciegas. Un sitio informativo de 5 secciones con diseño responsive (lo que ya sabes hacer) es un proyecto de nivel inicial; una tienda con carrito y pagos es nivel intermedio; una aplicación web con panel de administración es avanzado. Investiga los precios de tu ciudad, arma 3 paquetes (básico, profesional, premium) y muestra tu portafolio como prueba. Tu mejor herramienta de venta es un portafolio responsive y pulido: este libro te dio las técnicas, ahora construyelo.

Más práctica

Ejercicio 7

Agrega una sección de "habilidades" con 3 barras de progreso (HTML 95%, CSS 90%, JS 40%) usando el patrón de variables del caso de estudio.

SOLUCIÓN:

```

<div class="habilidades">
  <div class="hab">
    <div class="hab-top"><span>HTML</span><span>95%</span></div>
    <div class="barra"><span style="--nivel: 95%;"></span></div>
  </div>
  <div class="hab">
    <div class="hab-top"><span>CSS</span><span>90%</span></div>
    <div class="barra"><span style="--nivel: 90%;"></span></div>
  </div>
  <div class="hab">
    <div class="hab-top"><span>JavaScript</span><span>40%</span></div>
    <div class="barra"><span style="--nivel: 40%;"></span></div>
  </div>
</div>

```

(Corrige el typo del tercer div al copiar: debe ser class="barra".)

```

.habilidades { max-width: 600px; font-family: Arial, sans-serif; }

.hab { margin-bottom: 20px; }

.hab-top { display: flex; justify-content: space-between; margin-bottom: 8px;
font-weight: bold; }

.barra { background: #eee; border-radius: 10px; height: 12px; overflow: hidden; }

.barra span {
  display: block; height: 100%; border-radius: 10px;
  background: linear-gradient(to right, #e94560, #ff7e5f);
  width: 0; animation: llenar 1.5s ease forwards;
}

@keyframes llenar { to { width: var(--nivel); } }

```

El `overflow: hidden` en la barra recorta el relleno con los bordes redondeados. La variable `--nivel` personaliza cada barra.

Ejercicio 8

Crea el CSS para una página de "proximamente" (coming soon): pantalla completa, contenido centrado, título grande con clamp, fondo degradado oscuro y un campo de email con botón en la misma línea (flex).

SOLUCIÓN:

```
.proximamente {
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  text-align: center;
  background: linear-gradient(135deg, #0f1b2d, #1a3a5c);
  color: white;
  font-family: Arial, sans-serif;
  padding: 2rem;
}

.proximamente h1 { font-size: clamp(2.5rem, 8vw, 5rem); margin: 0 0 1rem 0; }

.proximamente p { opacity: 0.8; margin-bottom: 2rem; }

.aviso-form { display: flex; width: 100%; max-width: 450px; }

.aviso-form input {
  flex: 1;
  padding: 14px 18px;
  border: none;
  border-radius: 30px 0 0 30px;
  font-size: 16px;
}

.aviso-form button {
  padding: 14px 28px;
  border: none;
  border-radius: 0 30px 30px 0;
  background: #e94560;
  color: white;
  font-weight: bold;
  cursor: pointer;
  font-size: 16px;
}
```

El campo y el botón unidos forman una "píldora" compuesta: el input redondeado solo a la izquierda y el botón solo a la derecha. El flex los mantiene en una línea.

Ejercicio 9

Describe como harias responsive la tarjeta de perfil del Ejemplo 1 de esta unidad para que en el celular ocupe todo el ancho.

SOLUCIÓN: La tarjeta ya es bastante responsive por usar width fijo moderado, pero para el celular:

```
.perfil { width: 320px; max-width: 100%; }
```

Con `max-width: 100%`, en pantallas menores a 320px la tarjeta se encoge en vez de desbordar. Si está dentro de un contenedor con padding, mejor aún:

```
.contenedor-perfiles {  
  display: grid;  
  grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));  
  gap: 25px;  
  padding: 20px;  
}  
  
.perfil { width: 100%; }
```

Varias tarjetas en rejilla automática, cada una ocupando su celda. El patron auto-fit resuelve el responsive sin media queries.

Ejercicio 10

Proyecto de graduacion: describe la estructura HTML y el CSS clave de tu página portafolio personal (secciones, tecnicas de cada unidad que usarias y por que).

SOLUCIÓN: (Respuesta modelo, adaptala a tu caso.) Estructura: header fijo con navegación (Unidad 4: fixed + z-index, compensacion de espacio), hero de pantalla completa con título en clamp y CTA (Unidades 2 y 9), sección de proyectos en rejilla auto-fit con tarjetas que levitan al hover (Unidades 5, 6 y 7), sección sobre mi en dos columnas con grid areas (Unidad 6), habilidades con barras animadas (Unidad 7), formulario de contacto estilizado con validación visual (Unidad 8), footer oscuro (Unidad 2). Todo mobile-first con variables de diseño (Unidad 9), box-sizing border-box global (Unidad 3), y checklist de lanzamiento antes de publicar (esta ampliación). Si puedes planearlo así, el curso cumplió su objetivo.

APENDICE A: REFERENCIA RÁPIDA DE PROPIEDADES CSS

Guía de consulta rápida organizada por categoría. No es para memorizar de golpe, sino para tener a la mano mientras trabajas.

Selectores

```
/* Basicos */
p { }           /* etiqueta */
.titulo { }     /* clase */
#menu { }       /* id */
* { }           /* universal: todos */

/* Combinadores */
div p { }       /* descendiente: p dentro de div */
div > p { }     /* hijo directo */
h2 + p { }      /* hermano adyacente: p justo despues de h2 */
h2 ~ p { }      /* hermanos generales: p despues de h2 */

/* Agrupacion */
h1, h2, h3 { }  /* aplica a los tres */
```

Texto y tipografia

```

font-family: Arial, sans-serif;
font-size: 16px;           /* o rem, em */
font-weight: bold;         /* normal | bold | 100-900 */
font-style: italic;        /* normal | italic */
line-height: 1.7;          /* sin unidades = multiplo */
letter-spacing: 2px;
word-spacing: 4px;
text-align: center;        /* left | right | center | justify */
text-transform: uppercase; /* lowercase | capitalize */
text-decoration: underline; /* none | line-through */
text-shadow: 2px 2px 4px rgba(0,0,0,0.4);
white-space: nowrap;       /* no parte lineas */
overflow: hidden;
text-overflow: ellipsis;   /* "..." en texto cortado */

```

Color y fondo

```

color: #333;
background-color: #f5f5f5;
background-image: url("fondo.jpg");
background-repeat: no-repeat; /* repeat | repeat-x | repeat-y */
background-position: center;
background-size: cover;       /* contain | medidas */
background: #222 url("f.jpg") no-repeat center/cover;
opacity: 0.85;                /* 0 a 1 */

```

Modelo de caja

```

width: 300px; max-width: 100%; min-width: 200px;
height: 200px; max-height: 400px; min-height: 100px;
padding: 20px;           /* 1-4 valores */
margin: 0 auto;          /* centrado horizontal */
border: 2px solid #333;
border-radius: 10px;      /* 50% = circulo */
box-sizing: border-box;
box-shadow: 0 4px 12px rgba(0,0,0,0.15);

```

Display y posicionamiento

```
display: block | inline | inline-block | none | flex | grid;
position: static | relative | absolute | fixed | sticky;
top: 10px; right: 10px; bottom: 10px; left: 10px;
inset: 0;
z-index: 100;
float: left | right;           /* legado, evitar en layouts nuevos */
clear: both;
overflow: auto | hidden | scroll | visible;
visibility: visible | hidden;
cursor: pointer;
```

Flexbox

```
/* Contenedor */
display: flex;
flex-direction: row | column;
justify-content: flex-start | center | flex-end | space-between | space-around |
space-evenly;
align-items: stretch | flex-start | center | flex-end;
flex-wrap: nowrap | wrap;
gap: 15px;

/* Items */
flex: 1;           /* grow shrink basis */
flex-grow: 2;
flex-shrink: 0;
flex-basis: 200px;
align-self: center;
order: -1;
```

Grid

```
display: grid;
grid-template-columns: 1fr 1fr 1fr;
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
grid-template-rows: auto 1fr auto;
grid-template-areas: "a a" "b c";
grid-area: nombre;
grid-column: 1 / 3;          /* o span 2 */
grid-row: 2 / 4;
gap: 15px;
justify-items: center;
align-items: center;
```

Listas y tablas

```
list-style: none;
list-style-type: disc | circle | square | decimal;
table { border-collapse: collapse; width: 100%; }
```

Transiciones y animaciones

```
transition: all 0.3s ease;
transition: color 0.3s, transform 0.3s;
animation: nombre 2s ease-in-out infinite;
@keyframes nombre {
  0% { /* estado inicial */ }
  50% { /* estado medio */ }
  100% { /* estado final */ }
}
transform: translate(10px, 20px) | scale(1.2) | rotate(45deg) | skew(10deg);
transform-origin: center;
```

Responsive y variables

```
@media (max-width: 768px) { /* reglas */ }
@media (min-width: 1024px) { /* reglas */ }
@media (prefers-color-scheme: dark) { /* modo oscuro */ }
@media (prefers-reduced-motion: reduce) { /* sin animaciones */ }

:root { --color: #e94560; }
.elemento { color: var(--color); }
```

Unidades

```
px    /* absoluto: bordes, detalles */
%     /* relativo al padre: anchos fluidos */
em    /* relativo a la fuente del elemento */
rem   /* relativo a la fuente raiz: tipografia */
vw    /* % del ancho del viewport: heroes */
vh    /* % del alto del viewport */
```

APENDICE B: GLOSARIO

Accesibilidad: práctica de diseñar para que todas las personas, incluidas las que usan lector de pantalla o teclado, puedan usar el sitio. En CSS: focos visibles, contraste suficiente, respeto a prefers-reduced-motion.

Animación: secuencia de cambios de estilo definida con @keyframes que se ejecuta sola, con duración, repeticiones y dirección configurables.

Box-sizing: propiedad que define que incluye el width declarado. border-box (recomendado) incluye padding y borde; content-box (defecto histórico) solo el contenido.

Breakpoint: ancho de pantalla donde el diseño cambia mediante una media query. Típicos: 600px, 768px, 1024px, 1200px.

Cascada: el sistema por el que el navegador decide que regla gana cuando varias aplican al mismo elemento: origen, especificidad y orden.

Clase: atributo HTML reutilizable que agrupa elementos para estilizarlos. En CSS se selecciona con punto.nombre.

Contenedor flex: elemento con display: flex cuyos hijos directos se vuelven items flexibles.

Custom property: nombre oficial de las variables CSS. Se definen con --nombre y se usan con var(--nombre).

Declaración: par propiedad: valor dentro de una regla CSS. Ejemplo: color: red;

Degradado: transición suave entre dos o más colores generada por CSS (linear-gradient, radial-gradient), sin imágenes.

Display: propiedad que define como se comporta un elemento: block, inline, inline-block, none, flex, grid, etc.

Especificidad: sistema de puntos que decide que regla gana: etiqueta 1, clase 10, id 100, en línea 1000. A igualdad de puntos gana la última escrita.

Flexbox: sistema de layout unidimensional para distribuir espacio en una fila o columna.

Flujo normal: la forma por defecto en que los elementos aparecen en la página según el orden del HTML.

Fuente: familia tipográfica usada para el texto. Se declara con font-family y lista de respaldo.

Gap: espacio uniforme entre items flex o celdas grid, sin márgenes extra en los bordes.

Grid: sistema de layout bidimensional para cuadrículas de filas y columnas.

Herencia: mecanismo por el que algunas propiedades (color, font-family) pasan automáticamente de padres a hijos si no se definen.

Hover: estado de un elemento cuando el puntero está sobre él. Se estiliza con: hover.

Id: identificador único de un elemento en la página. En CSS se selecciona con numeral: #nombre.

Important: declaración que rompe la especificidad y gana siempre. Evitar salvo casos excepcionales.

Keyframe: cada paso definido dentro de @keyframes (0%, 50%, 100% o from/to).

Margin: espacio exterior transparente entre el borde del elemento y sus vecinos.

Media query: bloque de CSS condicional que aplica según características del dispositivo (@media).

Mobile-first: filosofía de diseñar primero para celular y mejorar progresivamente con min-width.

Modelo de caja: las cuatro capas de todo elemento: contenido, padding, border, margin.

Padding: espacio interior entre el contenido y el borde. Se pinta con el fondo del elemento.

Paleta: conjunto limitado de colores usado consistentemente en un proyecto.

Propiedad: cada característica controlable en CSS (color, width, display...).

Pseudo-clase: selector con un dos puntos que aplica según estado o posición (:hover, :nth-child...).

Pseudo-elemento: selector con doble dos puntos que estiliza partes o crea contenido virtual (::before, ::after...).

Regla CSS: unidad básica: selector + bloque de declaraciones entre llaves.

Responsive: diseño que se adapta a cualquier tamaño de pantalla.

Selector: la parte de la regla que elige a qué elementos aplica.

Transición: suavizado de un cambio de propiedad entre dos estados durante un tiempo definido.

Transform: modificación visual (mover, rotar, escalar) que no afecta el flujo del documento.

Unidad fr: fracción del espacio disponible en grid. 1fr 1fr = dos columnas iguales.

Valor: lo que toma una propiedad. Ejemplo: en color: red, "red" es el valor.

Viewport: el área visible del navegador. meta viewport es obligatorio para responsive en el celular.

z-index: orden de apilamiento de elementos superpuestos. Solo funciona con position (o flex/grid items).

APENDICE C: PROYECTO FINAL INTEGRADOR

Tu proyecto de graduacion: una página completa para un curso ficticio, usando todo el libro. Aquí esta la especificacion; el código es tu trabajo (usa los ejemplos de la Unidad 10 como base).

Especificacion

Secciones: 1) Navegación fija con logo y enlaces (se colapsa en el celular). 2) Hero de pantalla completa con título en clamp, subtítulo y dos CTAs. 3) Barra de estadísticas (estudiantes, cursos, horas). 4) Cursos destacados en rejilla auto-fit (6 tarjetas con imagen, título, precio y hover de elevacion). 5) Testimonios (3 citas estilizadas). 6) Planes de precio (3 planes, el del medio destacado). 7) Formulario de registro con validación visual. 8) Footer oscuro con columnas de enlaces.

Requisitos tecnicos

- Mobile-first: base de una columna, mejoras con min-width.
- Variables CSS para toda la paleta, espaciados y radios.
- box-sizing: border-box global.
- Tipografia con clamp en títulos, rem en cuerpo.
- Transiciones en todo lo interactivo (0.25s-0.35s).
- Animación de entrada en cascada al cargar (con prefers-reduced-motion).
- Imágenes con max-width: 100% y alt descriptivo.
- Foco visible en enlaces, botones y campos.
- Sin!important, sin estilos en línea (salvo variables dinamicas).
- Pasa el checklist de la Ampliación Unidad 10.

Criterios de evaluacion

Diseño (30%): paleta consistente, jerarquía tipografica clara, espaciado ritmico. Responsive (25%): sin scroll horizontal en 360px, rejillas que se adaptan, navegación usable en el celular. Interactividad (20%): hovers, focos, validación y animaciones funcionando. Código (15%): organizado, comentado, sin duplicacion, variables bien usadas. Detalle (10%): selection personalizado, scrollbar estilizado, micro-detalles que se sienten.

Cuando termines, publicalo y agregalo a tu portafolio. Es la prueba de que completaste este libro.

APENDICE D: GUIA DE DEPURACION RÁPIDA

"Mi CSS no se aplica para nada"

1. Verifica el `<link>`: ruta correcta, archivo existente (F12 > Red: busca 404).
2. Verifica que el selector coincida con el HTML (clase con punto, id con numeral).
3. Revisa llaves y puntos y coma: un error de sintaxis puede matar reglas siguientes.
4. Limpia la cache del navegador (Ctrl+F5) si editaste el CSS y no ves cambios.

"Se aplica pero no se ve como espero"

1. Inspecciona con F12: mira que reglas aplican y cuales estan tachadas (especificidad).
2. Revisa el modelo de caja en DevTools: padding/border/margin inesperados.
3. Confirma el display del elemento: ¿es inline y le pides width?
4. Confirma el position: ¿el absolute tiene padre relative? ¿el z-index tiene position?

"Se ve bien en PC pero mal en el celular"

1. ¿Tienes el meta viewport? (causa número 1)
2. Busca anchos fijos en px que desborden: usa max-width: 100%.
3. Imágenes sin max-width: agregalo global.
4. Revisa el orden de tus media queries (mobile-first: min-width de menor a mayor).

"La animación se ve mal"

1. ¿Falta animation-fill-mode: forwards? (salto al terminar)
2. ¿Animas width/height/top? Cambia a transform/opacity.
3. ¿La transición está en el estado base? (si no, solo funciona en una dirección)
4. ¿El elemento es inline? Las transformaciones necesitan inline-block o block.

"El layout se rompio de repente"

1. ¿Agregaste padding/borde a un width fijo sin border-box?
2. ¿Un margin colapso donde esperabas suma?
3. ¿Un float sin clearfix? (migra a flex/grid)
4. Deshaz el último cambio y reintrodúcelo por partes: el culpable aparece solo.

Herramientas que debes dominar

- F12 > Elementos: inspeccionar estilos aplicados y tachados.
- F12 > el diagrama del modelo de caja: medidas exactas por capa.
- F12 > botón de responsive (Ctrl+Shift+M): probar anchos de pantalla.
- F12 > Consola: errores de archivos no encontrados (404 del CSS o imágenes).
- Validador de contraste de DevTools: clic en el color para ver el ratio.

Regla de oro de la depuración: no adivines, inspecciona. El navegador siempre te dice la verdad.

APENDICE E: BANCO DE 30 MINI-RETOS

30 ejercicios cortos con solución. Hazlos sin mirar, uno al día, y tu CSS subira de nivel en un mes.

Retos 1-10: fundamentos y caja

RETO 1: Pinta todos los h2 de azul marino sin tocar el HTML.

SOLUCIÓN: `h2 {color: navy;}`

RETO 2: Crea la clase "resaltado" con fondo amarillo claro y padding de 8px.

SOLUCIÓN:

```
.resaltado {  
  background-color: #fff9c4;  
  padding: 8px;  
}
```

RETO 3: Haz que un div de 500px quede centrado horizontalmente.

SOLUCIÓN:

```
div {  
  width: 500px;  
  max-width: 100%;  
  margin: 0 auto;  
}
```

RETO 4: Crea un circulo rojo de 80px de diametro.

SOLUCIÓN:

```
.circulo {  
  width: 80px;  
  height: 80px;  
  border-radius: 50%;  
  background-color: red;  
}
```

RETO 5: Dale a una caja borde punteado azul de 3px con esquinas de 12px.

SOLUCIÓN:

```
.caja {  
  border: 3px dotted blue;  
  border-radius: 12px;  
}
```

RETO 6: Separa dos tarjetas con 24px entre ellas usando margin.

SOLUCIÓN:

```
.tarjeta { margin-bottom: 24px; }  
/* o gap si el padre es flex/grid */
```

RETO 7: Haz que el texto de un párrafo nunca mida menos de 14px ni más de 20px segun la pantalla, sin media queries.

SOLUCIÓN:

```
p { font-size: clamp(14px, 2.5vw, 20px); }
```

RETO 8: Convierte #FF8800 a RGB.

SOLUCIÓN: `rgb(255, 136, 0)`. FF=255, 88=136, 00=0.

RETO 9: Crea un degradado vertical de blanco a gris claro (#f5f5f5).

SOLUCIÓN:

```
.fondo { background: linear-gradient(to bottom, #ffffff, #f5f5f5); }
```

RETO 10: Calcula el ancho visible de .x {width: 200px; padding: 10px; border: 2px solid black;} con border-box.

SOLUCIÓN: 200px exactos. El contenido se ajusta a $200 - 20 - 4 = 176$ px.

Retos 11-20: layout y posicionamiento

RETO 11: Pon tres divs en fila con el mismo ancho y 15px de separación.

SOLUCIÓN:

```
.fila { display: flex; gap: 15px; }  
.fila > div { flex: 1; }
```

RETO 12: Centra un texto vertical y horizontalmente en un hero de 100vh.

SOLUCIÓN:

```
.hero {  
  min-height: 100vh;  
  display: flex;  
  justify-content: center;  
  align-items: center;  
  text-align: center;  
}
```

RETO 13: Fija un botón en la esquina inferior derecha de la pantalla.

SOLUCIÓN:

```
.flotante {  
  position: fixed;  
  bottom: 20px;  
  right: 20px;  
  z-index: 500;  
}
```

RETO 14: Crea una rejilla de 3 columnas iguales con 20px de separación.

SOLUCIÓN:

```
.rejilla {  
  display: grid;  
  grid-template-columns: repeat(3, 1fr);  
  gap: 20px;  
}
```

RETO 15: Haz que una imagen nunca desborde su contenedor.

SOLUCIÓN:

```
img { max-width: 100%; height: auto; display: block; }
```

RETO 16: Coloca una etiqueta "OFERTA" sobre la esquina superior izquierda de una imagen.

SOLUCIÓN:

```
.foto-wrap { position: relative; display: inline-block; }  
.etiqueta {  
  position: absolute;  
  top: 10px;  
  left: 10px;  
  background: #e94560;  
  color: white;  
  padding: 5px 12px;  
  border-radius: 4px;  
  font-size: 12px;  
  font-weight: bold;  
}
```

RETO 17: Crea dos columnas: 250px fija y resto flexible.

SOLUCIÓN:

```
.layout { display: flex; gap: 20px; }  
.lateral { flex: 0 0 250px; }  
.contenido { flex: 1; }
```

RETO 18: Haz que el último párrafo de un artículo no tenga margen inferior.

SOLUCIÓN:

```
article p:last-child { margin-bottom: 0; }
```

RETO 19: Cebra las filas de una tabla con gris claro en las pares.

SOLUCIÓN:

```
tbody tr:nth-child(even) { background-color: #f5f5f5; }
```

RETO 20: Oculta un elemento pero conservando su espacio.

SOLUCIÓN:

```
.oculto { visibility: hidden; }  
/* display: none lo eliminaria del flujo */
```

Retos 21-30: interacción, responsive y detalle

RETO 21: Un botón que cambia de azul a verde en 0.3s al hover.

SOLUCIÓN:

```
.btn { background: blue; transition: background-color 0.3s; }  
.btn:hover { background: green; }
```

RETO 22: Una caja que crece 10% al hover sin mover a sus vecinas.

SOLUCIÓN:

```
.caja { transition: transform 0.3s; }  
.caja:hover { transform: scale(1.1); }
```

RETO 23: Spinner de carga de 40px girando infinitamente.

SOLUCIÓN:

```
.spinner {  
  width: 40px; height: 40px;  
  border: 4px solid #eee;  
  border-top-color: #e94560;  
  border-radius: 50%;  
  animation: girar 0.8s linear infinite;  
}  
@keyframes girar { to { transform: rotate(360deg); } }
```

RETO 24: En el celular (< 600px) una rejilla de 4 columnas pasa a 1 columna.

SOLUCIÓN:

```
.rejilla { display: grid; grid-template-columns: 1fr; gap: 15px; }  
@media (min-width: 600px) { .rejilla { grid-template-columns: repeat(4, 1fr); } }
```

RETO 25: Define --acento y usalo en un botón y un título.

SOLUCIÓN:

```
:root { --acento: #e94560; }  
.boton { background-color: var(--acento); }  
h2 { color: var(--acento); }
```

RETO 26: Campo de texto con borde rojo cuando su valor es invalido (y ya se escribio algo).

SOLUCIÓN:

```
input:not(:placeholder-shown):invalid { border-color: red; }
```

RETO 27: Agrega "..." a un texto largo que se corta en una línea.

SOLUCIÓN:

```
.cortar {  
  white-space: nowrap;  
  overflow: hidden;  
  text-overflow: ellipsis;  
}
```

RETO 28: Título con línea decorativa roja de 60px debajo, centrada.

SOLUCIÓN:

```
.titulo { text-align: center; }  
.titulo::after {  
  content: "";  
  display: block;  
  width: 60px;  
  height: 4px;  
  background: #e94560;  
  margin: 10px auto 0;  
}
```

RETO 29: Sombra elegante estandar para tarjetas.

SOLUCIÓN:

```
.tarjeta { box-shadow: 0 4px 12px rgba(0, 0, 0, 0.12); }
```

RETO 30: Sitio con modo oscuro automático según el sistema.

SOLUCIÓN:

```
:root { --fondo: #fff; --texto: #222; }  
body { background: var(--fondo); color: var(--texto); }  
@media (prefers-color-scheme: dark) {  
  :root { --fondo: #121212; --texto: #e0e0e0; }  
}
```

APENDICE F: PLANTILLAS LISTAS PARA COPIAR

Cinco componentes completos, probados y comentados. Copialos a tus proyectos y adapталos.

Plantilla 1: barra de navegación responsive

```
/* Navegacion: hamburguesa en movil, links en escritorio */
:root {
  --nav-fondo: #1a1a2e;
  --nav-texto: #ffffff;
  --nav-acento: #e94560;
}

.navbar {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: var(--nav-fondo);
  padding: 1rem 1.5rem;
  position: sticky;
  top: 0;
  z-index: 100;
  font-family: Arial, sans-serif;
}

.navbar .logo {
  color: var(--nav-texto);
  font-weight: bold;
  font-size: 1.35rem;
  text-decoration: none;
}

.navbar .toggle {
  display: block;
  background: none;
  border: none;
  color: var(--nav-texto);
  font-size: 1.7rem;
  cursor: pointer;
}
```

```

.navbar .links {
  display: none;
  list-style: none;
  margin: 0;
  padding: 0;
}

.navbar .links a {
  color: var(--nav-texto);
  text-decoration: none;
  padding: 0.9rem 1.2rem;
  display: block;
  border-radius: 6px;
  transition: background-color 0.25s;
}

.navbar .links a:hover {
  background-color: rgba(255, 255, 255, 0.1);
}

.navbar .links a.cta {
  background-color: var(--nav-acento);
  font-weight: bold;
}

@media (min-width: 768px) {
  .navbar .toggle { display: none; }
  .navbar .links { display: flex; gap: 0.4rem; align-items: center; }
}

```

Plantilla 2: hero de pantalla completa

```

.hero {
  min-height: 92vh;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  text-align: center;
  padding: 3rem 1.5rem;
  color: #ffffff;
  background:

```

```

    linear-gradient(rgba(15, 27, 45, 0.72), rgba(15, 27, 45, 0.72)),
    url("hero.jpg") center/cover no-repeat;
    font-family: Arial, sans-serif;
}

.hero .kicker {
    text-transform: uppercase;
    letter-spacing: 3px;
    font-size: 0.8rem;
    color: #ffb3c1;
    margin-bottom: 1rem;
}

.hero h1 {
    font-size: clamp(2.4rem, 7vw, 4.5rem);
    margin: 0 0 1.2rem 0;
    line-height: 1.15;
    max-width: 900px;
}

.hero p {
    font-size: clamp(1rem, 2.5vw, 1.3rem);
    opacity: 0.9;
    max-width: 620px;
    margin: 0 0 2.2rem 0;
    line-height: 1.7;
}

.hero .acciones {
    display: flex;
    gap: 1rem;
    flex-wrap: wrap;
    justify-content: center;
}

.btn-primario {
    background-color: #e94560;
    color: #ffffff;
    text-decoration: none;
    padding: 1rem 2.6rem;
    border-radius: 50px;
    font-weight: bold;
    transition: transform 0.25s, box-shadow 0.25s;
}

```

```

.btn-primario:hover {
  transform: translateY(-3px);
  box-shadow: 0 12px 25px rgba(233, 69, 96, 0.45);
}

.btn-secundario {
  color: #ffffff;
  text-decoration: none;
  padding: 1rem 2.6rem;
  border-radius: 50px;
  font-weight: bold;
  border: 2px solid rgba(255, 255, 255, 0.6);
  transition: background-color 0.25s, border-color 0.25s;
}

.btn-secundario:hover {
  background-color: rgba(255, 255, 255, 0.12);
  border-color: #ffffff;
}

```

Plantilla 3: rejilla de tarjetas de curso

```

.cursos {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(270px, 1fr));
  gap: 1.6rem;
  max-width: 1200px;
  margin: 0 auto;
  padding: 3rem 1.5rem;
  font-family: Arial, sans-serif;
}

.curso {
  background: #ffffff;
  border-radius: 16px;
  overflow: hidden;
  box-shadow: 0 3px 12px rgba(0, 0, 0, 0.09);
  transition: transform 0.3s ease, box-shadow 0.3s ease;
  display: flex;
  flex-direction: column;
}

```

```
.curso:hover {
  transform: translateY(-8px);
  box-shadow: 0 16px 34px rgba(0, 0, 0, 0.15);
}

.curso img {
  width: 100%;
  height: 170px;
  object-fit: cover;
  display: block;
}

.curso .cuerpo {
  padding: 1.4rem;
  display: flex;
  flex-direction: column;
  flex: 1;
}

.curso .nivel {
  font-size: 0.72rem;
  text-transform: uppercase;
  letter-spacing: 1.5px;
  color: #e94560;
  font-weight: bold;
  margin-bottom: 0.5rem;
}

.curso h3 {
  margin: 0 0 0.6rem 0;
  color: #1a1a2e;
  font-size: 1.2rem;
}

.curso p {
  color: #666;
  font-size: 0.95rem;
  line-height: 1.6;
  margin: 0 0 1.2rem 0;
  flex: 1;
}

.curso .pie {
  display: flex;
```

```

    justify-content: space-between;
    align-items: center;
}

.curso .precio {
    font-size: 1.35rem;
    font-weight: bold;
    color: #1a1a2e;
}

.curso .ver {
    color: #e94560;
    text-decoration: none;
    font-weight: bold;
}

.curso .ver::after {
    content: " ->";
    display: inline-block;
    transition: transform 0.25s;
}

.curso .ver:hover::after {
    transform: translateX(4px);
}

```

Plantilla 4: pie de página con columnas

```

.footer {
    background-color: #0f1b2d;
    color: #a0aec0;
    font-family: Arial, sans-serif;
    padding: 3.5rem 1.5rem 1.5rem;
}

.footer .columnas {
    display: grid;
    grid-template-columns: 1fr;
    gap: 2rem;
    max-width: 1100px;
    margin: 0 auto 2.5rem;
}

```

```
@media (min-width: 768px) {  
  .footer .columnas { grid-template-columns: 2fr 1fr 1fr 1fr; }  
}  
  
.footer h4 {  
  color: #ffffff;  
  margin: 0 0 1rem 0;  
  font-size: 1rem;  
}  
  
.footer ul {  
  list-style: none;  
  padding: 0;  
  margin: 0;  
}  
  
.footer li { margin-bottom: 0.6rem; }  
  
.footer a {  
  color: #a0aec0;  
  text-decoration: none;  
  transition: color 0.25s;  
}  
  
.footer a:hover { color: #ffffff; }  
  
.footer .marca p {  
  line-height: 1.7;  
  font-size: 0.92rem;  
  max-width: 300px;  
}  
  
.footer .legal {  
  border-top: 1px solid rgba(255, 255, 255, 0.12);  
  padding-top: 1.4rem;  
  text-align: center;  
  font-size: 0.85rem;  
  max-width: 1100px;  
  margin: 0 auto;  
}
```

Plantilla 5: formulario de contacto

```
.form-card {
  max-width: 520px;
  margin: 3rem auto;
  padding: 2.5rem;
  background: #ffffff;
  border-radius: 18px;
  box-shadow: 0 12px 34px rgba(0, 0, 0, 0.12);
  font-family: Arial, sans-serif;
}

.form-card h2 {
  margin: 0 0 0.5rem 0;
  color: #1a1a2e;
  text-align: center;
}

.form-card .sub {
  text-align: center;
  color: #888;
  margin: 0 0 1.8rem 0;
  font-size: 0.95rem;
}

.form-card label {
  display: block;
  font-weight: bold;
  font-size: 0.88rem;
  margin: 1.1rem 0 0.45rem 0;
  color: #333;
}

.form-card input,
.form-card textarea,
.form-card select {
  width: 100%;
  box-sizing: border-box;
  padding: 0.85rem 1rem;
  border: 1.5px solid #ddd;
  border-radius: 10px;
  font-size: 1rem;
  font-family: inherit;
  transition: border-color 0.25s, box-shadow 0.25s;
```

```

}

.form-card textarea { min-height: 120px; resize: vertical; }

.form-card input:focus,
.form-card textarea:focus,
.form-card select:focus {
  outline: none;
  border-color: #4a90d9;
  box-shadow: 0 0 0 3px rgba(74, 144, 217, 0.2);
}

.form-card input:not(:placeholder-shown):valid { border-color: #2a9d8f; }
.form-card input:not(:placeholder-shown):invalid { border-color: #e94560; }

.form-card button {
  width: 100%;
  margin-top: 1.6rem;
  padding: 1rem;
  border: none;
  border-radius: 10px;
  background: linear-gradient(to right, #e94560, #ff7e5f);
  color: #ffffff;
  font-size: 1.05rem;
  font-weight: bold;
  cursor: pointer;
  transition: transform 0.2s, box-shadow 0.2s;
}

.form-card button:hover {
  transform: translateY(-2px);
  box-shadow: 0 8px 20px rgba(233, 69, 96, 0.35);
}

.form-card button:active { transform: translateY(0); }

```

Usa estas plantillas como punto de partida, no como jaula: cambiales colores, espaciados y tipografías con tus variables y hazlas tuyas.

APENDICE G: PREGUNTAS FRECUENTES

Las 20 preguntas que más hacen los estudiantes de CSS, respondidas a fondo.

1. ¿CSS es un lenguaje de programación?

No en el sentido tradicional: no tiene variables de lógica, ciclos ni condicionales como JavaScript. Es un lenguaje de hojas de estilo: declaras reglas y el navegador las aplica. Dicho eso, el CSS moderno tiene variables, calculos con `calc()`, condicionales con media queries y contadores, así que la línea se ha vuelto borrosa. Lo importante no es la etiqueta, sino dominarlo.

2. ¿En que orden aprendo HTML, CSS y JavaScript?

En ese orden exacto: HTML (estructura), CSS (presentación, este libro) y JavaScript (comportamiento). Cada uno se apoya en el anterior. Saltarte CSS para ir directo a JavaScript es como querer correr sin aprender a caminar: tarde o temprano tendrás que regresar.

3. ¿Debo memorizar todas las propiedades?

No. Memoriza las 30-40 más usadas (las de este libro) y consulta la referencia para el resto. Los profesionales buscan propiedades en la documentación todos los días; lo que los distingue es que saben que existe una propiedad para cada problema y donde buscarla.

4. ¿Por que mi CSS no funciona? (checklist de 60 segundos)

Primero: ¿el `<link>` apunta al archivo correcto? Segundo: ¿el selector coincide (punto para clase, numeral para id)? Tercero: ¿hay llaves y puntos y coma completos? Cuarto: ¿otra regla con más especificidad está ganando? (F12, reglas tachadas). El 95% de los casos cae en uno de estos cuatro.

5. ¿Flexbox o grid?

Una dimension (una fila o columna): flexbox. Dos dimensiones (filas y columnas alineadas): grid. En la práctica los combinas: grid para el esqueleto de la página, flexbox dentro de cada zona para alinear su contenido.

6. ¿Cuando uso px, rem o%?

px para detalles exactos (bordes de 1px, sombras). rem para tipografía y espaciados (respeta la accesibilidad del usuario).% para anchos fluidos dentro de contenedores. vw/vh para secciones de pantalla completa. clamp() para tipografía que escala sola.

7. ¿Como centro un div vertical y horizontalmente?

La receta moderna:

```
.centrar {  
  display: flex;  
  justify-content: center;  
  align-items: center;  
  min-height: 100vh;  
}
```

Alternativa clásica con posicionamiento: position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%); dentro de un padre relative.

8. ¿Por que se desborda mi layout en el celular?

Causas en orden de frecuencia: falta el meta viewport; un width fijo en px mayor que la pantalla; imágenes sin max-width: 100%; padding/borde sumados a width: 100% sin border-box. Arregla en ese orden.

9. ¿Que es el "fold" y debo diseñar para el?

El "fold" (el doblez) es la línea donde termina la pantalla sin hacer scroll. Mito viejo: "todo importante debe estar arriba del fold". Realidad: los usuarios si hacen scroll; lo importante es que el hero comunique el valor en 3 segundos y que haya una razon para seguir bajando (indicador visual, contenido parcial visible).

10. ¿Puedo usar CSS sin saber diseño gráfico?

Si. Sigue sistemas: paleta de 3-4 colores, 2 familias tipograficas, escala de espaciado en multiplos de 8, sombras por niveles. El 80% del "buen diseño" es consistencia sistematica, no talento artistico. Y puedes inspirarte en sitios que admires (sin copiarlos tal cual).

11. ¿Como hago un modo oscuro?

Con variables CSS: define la paleta en:root, usala en todo, y redefine las variables dentro de @media (prefers-color-scheme: dark) para automático, o dentro de body.moda-oscuro para manual con un botón. Nunca dupliques componentes enteros.

12. ¿Las animaciones afectan el rendimiento?

Solo si animas las propiedades equivocadas. Anima transform y opacity (aceleradas por GPU) y evita width, height, margin, top/left en animaciones. Una o dos animaciones suaves no afectan nada; cincuenta animaciones de layout si.

13. ¿Que es la especificidad y por que me persigue?

Es el sistema de puntos (etiqueta 1, clase 10, id 100) que decide que regla gana. Te persigue cuando escribes selectores muy específicos (como #pagina.contenido p.texto) que luego no puedes sobrescribir sin selectores aún más específicos. La cura: selectores cortos y planos, metodologia BEM u organización por componentes.

14. ¿Debo usar un framework como Bootstrap o Tailwind?

Primero aprende CSS puro (lo que hiciste en este libro). Después, los frameworks son herramientas validas que aceleran el trabajo. Pero quien solo sabe el framework y no el CSS que hay debajo se atora en cuanto necesita algo personalizado. Tu ya tienes la base: ahora los frameworks seran atajos, no muletas.

15. ¿Como organizo un archivo CSS grande?

De lo general a lo específico: 1) variables y reset, 2) estilos base (body, títulos, enlaces), 3) utilidades y layout (contenedores, rejillas), 4) componentes (botones, tarjetas, formularios), 5) secciones específicas, 6) media queries o temas. Con comentarios que marquen cada sección. En proyectos grandes, divide en varios archivos (base.css, componentes.css...) e importalos.

16. ¿Que hago si dos estilos se contradicen y no se cual gana?

No adivines: inspecciona. F12 sobre el elemento, panel de estilos: el navegador muestra todas las reglas que aplican en orden de prioridad y tacha las perdedoras. Ahi ves exactamente quien gana y por cuantos puntos. Es la habilidad de depuracion más valiosa en CSS.

17. ¿Como pruebo mi sitio en el celular si no tengo muchos dispositivos?

Primero: el modo responsive de DevTools (Ctrl+Shift+M) para anchos comunes. Segundo: prueba en tu propio celular en la red local. Tercero: pide a alguien más que lo abra en su telefono. El modo responsive simula tamaño, no tacto ni rendimiento real: siempre prueba en un dispositivo fisico antes de entregar.

18. ¿Cuanto CSS necesito para conseguir trabajo?

Lo de este libro te pone por encima del nivel junior tipico en CSS: selectores, caja, flexbox, grid, responsive, formularios y animaciones. Complementalo con JavaScript y un framework, más un portafolio con 3-4 proyectos pulidos, y tienes perfil contrataable como desarrollador frontend junior. El portafolio pesa más que el certificado.

19. ¿El CSS va a desaparecer con la IA?

No. Las herramientas de IA generan CSS cada vez mejor, pero alguien tiene que saber pedirlo, revisarlo, depurarlo e integrarlo. Quien domina CSS usa la IA como multiplicador de velocidad; quien no lo domina no puede distinguir una salida buena de una mala. Aprender CSS en la era de la IA es más valioso, no menos.

20. ¿Cual es el siguiente paso después de este libro?

JavaScript, en lucasegaacademy.com. Con HTML + CSS + JavaScript construyes cualquier interfaz web. Después: un framework (React es la apuesta más segura laboralmente), Git para control de versiones, y despliegue (publicar tus sitios). El camino completo está en la academia, gratis, como todo lo demás. Nos vemos en el siguiente curso.

EXAMEN FINAL

Instrucciones: responde las 50 preguntas sin consultar el libro. Las preguntas 1 a 20 son de opción múltiple (una sola respuesta correcta), las preguntas 21 a 35 son de verdadero o falso, y las preguntas 36 a 50 son abiertas: escribe el código CSS o la explicación que se pide. Cada pregunta vale 2 puntos, total 100. Necesitas 80 para aprobar.

Preguntas de opción múltiple

PREGUNTA 1: ¿Qué significan las siglas CSS?

a) Computer Style Sheets b) Cascading Style Sheets c) Creative Style System d) Colorful Style Sheets

PREGUNTA 2: ¿Cuál es la forma profesional recomendada de incluir CSS en un proyecto?

a) Con el atributo style en cada etiqueta b) Con un bloque `<style>` en cada página c) Con un archivo externo enlazado con `<link>` d) Con JavaScript

PREGUNTA 3: En el CSS, ¿cómo se selecciona un elemento con `class="menu"`?

a) `#menú` b) `.menú` c) `menú` d) `*menú`

PREGUNTA 4: En el CSS, ¿cómo se selecciona un elemento con `id="principal"`?

a) `.principal` b) `#principal` c) `principal` d) `@principal`

PREGUNTA 5: ¿Qué color representa el hexadecimal `#00FF00`?

a) Rojo puro b) Azul puro c) Verde puro d) Blanco

PREGUNTA 6: ¿Qué hace la declaración `rgba(0, 0, 0, 0.5)`?

a) Pinta un gris sólido b) Pinta negro con 50% de opacidad c) Pinta blanco con 50% de opacidad d) Es una sintaxis inválida

PREGUNTA 7: ¿Cuál es la diferencia entre `margin` y `padding`?

a) Son lo mismo con distinto nombre b) `margin` es espacio hacia adentro y `padding` hacia afuera c) `margin` es espacio hacia afuera y `padding` hacia adentro d) `margin` solo funciona en vertical y `padding` en horizontal

PREGUNTA 8: ¿Qué capas forman el modelo de caja, de adentro hacia afuera?

a) margin, border, padding, contenido b) contenido, padding, border, margin c) contenido, margin, border, padding d) border, contenido, padding, margin

PREGUNTA 9: ¿Qué hace `box-sizing: border-box`?

a) Elimina los bordes del elemento b) Hace que width incluya contenido, padding y borde c) Centra la caja en la página d) Convierte la caja en un círculo

PREGUNTA 10: ¿Qué valor de `display` hace que un elemento desaparezca por completo sin ocupar espacio?

a) hidden b) invisible c) none d) block

PREGUNTA 11: Con `position: absolute`, ¿respecto a qué se posiciona el elemento?

a) Siempre respecto a la ventana del navegador b) Respecto a su ancestro posicionado más cercano c) Respecto a su hermano anterior d) Respecto al centro de la página

PREGUNTA 12: ¿Qué combinación centra un elemento horizontal y verticalmente con flexbox?

a) display: flex; margin: auto; b) display: flex; justify-content: center; align-items: center; c) display: block; text-align: center; d) position: center;

PREGUNTA 13: En flexbox con `flex-direction: row`, ¿qué controla `justify-content`?

a) La alineación vertical b) La alineación horizontal (eje principal) c) El tamaño de los ítems d) El orden de los ítems

PREGUNTA 14: ¿Qué significa `1fr` en grid?

a) Un frame de animación b) Una fracción del espacio disponible c) Un pixel flexible d) La primera fila

PREGUNTA 15: ¿Qué hace `grid-template-columns: repeat(auto-fit, minmax(250px, 1fr))`?

a) Crea exactamente 250 columnas b) Crea columnas responsive automáticas de mínimo 250px c) Repite la primera columna 250 veces d) Es sintaxis inválida

PREGUNTA 16: ¿Cuál es la diferencia entre `:nth-child(2)` y `:nth-of-type(2)`?

a) Son equivalentes b) nth-child cuenta todos los hermanos, nth-of-type solo los de ese tipo de etiqueta c) nth-child solo funciona en listas d) nth-of-type cuenta desde cero

PREGUNTA 17: ¿Qué propiedad se usa para animar un cambio suave entre dos estados?

a) animation b) transform c) transition d) keyframes

PREGUNTA 18: ¿Qué valores acepta animation-iteration-count para repetir infinitamente?

a) always b) infinite c) forever d) loop

PREGUNTA 19: ¿Cuáles son las dos propiedades más eficientes para animar por rendimiento?

a) width y height b) margin y padding c) transform y opacity d) top y left

PREGUNTA 20: ¿Qué hace box-shadow: inset 0 2px 5px rgba(0,0,0,0.2)?

a) Una sombra externa hacia abajo b) Una sombra interna (efecto hundido) c) Un borde degradado d) Nada, la sintaxis es inválida

Preguntas de verdadero o falso

PREGUNTA 21: El selector .titulo en CSS selecciona el elemento con id="titulo". (V/F)

PREGUNTA 22: Un id puede repetirse en varios elementos de la misma página sin problema. (V/F)

PREGUNTA 23: En especificidad, un selector de clase (10 puntos) le gana a un selector de etiqueta (1 punto). (V/F)

PREGUNTA 24: #FFF es equivalente a #FFFFFF. (V/F)

PREGUNTA 25: line-height: 1.7 significa 1.7 veces el tamaño de la fuente. (V/F)

PREGUNTA 26: La ruta en url() de background-image se resuelve relativa al archivo HTML, no al CSS. (V/F)

PREGUNTA 27: Con box-sizing: border-box, una caja con width: 300px, padding: 20px y border: 5px mide 300px de ancho visible. (V/F)

PREGUNTA 28: Los márgenes verticales de dos bloques apilados se suman. (V/F)

PREGUNTA 29: A un elemento con `display: inline` se le puede dar `width` y `height` efectivos. (V/F)

PREGUNTA 30: `position: fixed` posiciona el elemento respecto a la ventana del navegador y no se mueve con el scroll. (V/F)

PREGUNTA 31: `z-index` funciona en cualquier elemento, tenga o no `position`. (V/F)

PREGUNTA 32: En flexbox, si cambias `flex-direction` a `column`, `justify-content` pasa a controlar el eje vertical. (V/F)

PREGUNTA 33: Los pseudo-elementos `::before` y `::after` funcionan sin la propiedad `content`. (V/F)

PREGUNTA 34: `transition` puede animar un elemento en bucle infinito por sí sola. (V/F)

PREGUNTA 35: Sin el meta viewport, las media queries no funcionan correctamente en celulares reales. (V/F)

Preguntas abiertas

PREGUNTA 36: Escribe el CSS para un botón con clase "comprar": fondo #e94560, texto blanco, padding 14px 35px, bordes redondeados de 8px, y que al pasar el mouse el fondo cambie a #1a1a2e suavemente en 0.3 segundos.

PREGUNTA 37: Escribe el HTML y CSS mínimo para enlazar un archivo externo "estilos.css" y pintar todos los párrafos de color gris oscuro (#333) con fuente Arial.

PREGUNTA 38: Explica con tus palabras la diferencia entre `display: none` y `visibility: hidden`.

PREGUNTA 39: Escribe el CSS para centrar un div de clase "caja" (ancho 400px) horizontalmente en la página usando márgenes.

PREGUNTA 40: Dado este HTML, escribe el CSS para que la insignia quede en la esquina superior derecha de la tarjeta:

```
<div class="tarjeta">
  <span class="insignia">NUEVO</span>
  <p>Producto increíble</p>
</div>
```

PREGUNTA 41: Escribe los keyframes de una animación llamada "pulso" que escale un elemento de 1 a 1.2 y de regreso, y aplícala en bucle infinito de 2 segundos con timing ease-in-out.

PREGUNTA 42: Escribe una media query mobile-first que haga que una rejilla ".galeria" tenga 1 columna en el celular y 3 columnas en pantallas de 900px o más.

PREGUNTA 43: Define tres variables CSS en:root (color primario, color de acento y espaciado base) y úsalas en una clase ".tarjeta".

PREGUNTA 44: Escribe el CSS para una tabla con clase "datos" que incluya: border-collapse, encabezados con fondo oscuro y texto blanco, filas alternadas (cebreado) y resaltado de fila al pasar el mouse.

PREGUNTA 45: Explica qué está mal en este código y corrígelo:

```
.aviso
  color: red
  background: yellow;
```

PREGUNTA 46: Escribe el CSS para un menú de navegación fijo en la parte superior que no se mueva con el scroll, ocupe todo el ancho y quede sobre el contenido (z-index 100). Incluye la compensación necesaria en el contenido.

PREGUNTA 47: ¿Qué color final tendrá el texto y por qué?

```
<p id="texto" class="rojo">Hola</p>
```

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

PREGUNTA 48: Escribe el CSS para un campo de texto con clase "email" que al recibir el foco muestre borde azul (#4a90d9) con un anillo suave, sin usar el outline por defecto del navegador.

PREGUNTA 49: Describe los pasos para convertir un diseño de escritorio a responsive con enfoque mobile-first.

PREGUNTA 50: Proyecto mini: escribe el HTML y CSS completo de una tarjeta de producto con imagen, nombre, precio, botón "Comprar" y efecto hover de elevación. Debe usar variables CSS y ser responsive (imagen que no se desborde).

Respuestas y soluciones

Opción múltiple

RESPUESTA 1: b) Cascading Style Sheets. "Cascading" (en cascada) se refiere a cómo las reglas fluyen y se combinan por prioridad; "Style Sheets" (hojas de estilo) porque describen la presentación.

RESPUESTA 2: c) Con un archivo externo enlazado con `<link>`. Es la forma profesional porque centraliza el diseño: un solo archivo controla todo el sitio, se guarda en caché y es fácil de mantener.

RESPUESTA 3: b).menú. El punto antes del nombre indica selector de clase en CSS.

RESPUESTA 4: b) #principal. El numeral antes del nombre indica selector de id en CSS.

RESPUESTA 5: c) Verde puro. El formato es #RRGGBB: 00 de rojo, FF (máximo) de verde, 00 de azul.

RESPUESTA 6: b) Pinta negro con 50% de opacidad. Los tres primeros valores son RGB (0,0,0 = negro) y el cuarto es el canal alfa: 0.5 = 50% de opacidad.

RESPUESTA 7: c) margin es espacio hacia afuera y padding hacia adentro. El margin separa la caja de sus vecinas; el padding separa el contenido del borde dentro de la misma caja.

RESPUESTA 8: b) contenido, padding, border, margin. De adentro hacia afuera: lo que contiene el elemento, el aire interno, la línea del borde y la separación externa.

RESPUESTA 9: b) Hace que width incluya contenido, padding y borde. Con border-box, lo que declaras en width es el ancho visible total; el navegador ajusta el contenido para que cuadre.

RESPUESTA 10: c) none. `display: none` saca al elemento del flujo por completo: no se ve y no ocupa espacio.

RESPUESTA 11: b) Respecto a su ancestro posicionado más cercano. Busca hacia arriba el antepasado más próximo con `position` distinto de `static`; si no hay ninguno, usa el documento. Por eso el padre suele llevar `position: relative`.

RESPUESTA 12: b) `display: flex`; `justify-content: center`; `align-items: center`; La receta del centrado perfecto: `justify-content` en el eje principal (horizontal por defecto) y `align-items` en el transversal (vertical).

RESPUESTA 13: b) La alineación horizontal (eje principal). Con `direction row`, el eje principal es horizontal; `justify-content` siempre controla el eje principal.

RESPUESTA 14: b) Una fracción del espacio disponible. `1fr 1fr 1fr` reparte el espacio en tres partes iguales; `2fr 1fr` da el doble a la primera columna.

RESPUESTA 15: b) Crea columnas responsive automáticas de mínimo 250px. El navegador crea tantas columnas de mínimo 250px como quepan y reparte el sobrante con `1fr`. Es la línea mágica de las galerías sin media queries.

RESPUESTA 16: b) `nth-child` cuenta todos los hermanos, `nth-of-type` solo los de ese tipo de etiqueta. Con un `h2` antes de los párrafos, `p:nth-child(2)` selecciona el primer párrafo (que es el segundo hijo), mientras `p:nth-of-type(2)` selecciona el segundo párrafo.

RESPUESTA 17: c) `transition`. Suaviza el cambio de una propiedad entre dos estados definidos (por ejemplo, `normal` y `hover`) durante un tiempo determinado.

RESPUESTA 18: b) `infinite`. Hace que la animación se repita para siempre.

RESPUESTA 19: c) `transform` y `opacity`. El navegador las anima con la GPU sin recalcular el layout; animar `width`, `height` o `top/left` causa tirones en páginas complejas.

RESPUESTA 20: b) Una sombra interna (efecto hundido). La palabra `inset` invierte la sombra hacia adentro del elemento.

Verdadero o falso

RESPUESTA 21: Falso. `titulo` selecciona `class="titulo"`; para `id="titulo"` se usa `#titulo`.

RESPUESTA 22: Falso. Cada `id` debe ser único en toda la página. Repetirlo produce HTML inválido y comportamiento impredecible en JavaScript.

RESPUESTA 23: Verdadero. En el sistema de puntos (etiqueta 1, clase 10, id 100), la clase le gana a la etiqueta.

RESPUESTA 24: Verdadero. Cuando los tres pares tienen dígitos repetidos, se puede usar la forma abreviada de 3 dígitos.

RESPUESTA 25: Verdadero. Sin unidades, line-height se interpreta como múltiplo del tamaño de fuente, y es la forma recomendada.

RESPUESTA 26: Falso. Se resuelve relativa al archivo CSS, no al HTML. Si el CSS está en css/ y la imagen en imágenes/, la ruta es ../imagenes/foto.jpg.

RESPUESTA 27: Verdadero. Con border-box el width declarado (300px) es el ancho visible total; el contenido se reduce a $300 - 40 - 10 = 250\text{px}$ automáticamente.

RESPUESTA 28: Falso. Los márgenes verticales colapsan: se traslapan y prevalece el mayor, no se suman.

RESPUESTA 29: Falso. Los elementos inline ignoran width, height y márgenes verticales. Hay que cambiarlos a inline-block o block.

RESPUESTA 30: Verdadero. Fixed se ancla al viewport: no se mueve con el scroll. Hay que compensar el espacio que ocupa porque sale del flujo.

RESPUESTA 31: Falso. z-index solo funciona en elementos posicionados (relative, absolute, fixed, sticky) o ítems flex/grid.

RESPUESTA 32: Verdadero. justify-content y align-items se refieren a los ejes, no a direcciones absolutas; al cambiar a column, los ejes se intercambian.

RESPUESTA 33: Falso. Sin content (aunque sea content: ""), el pseudo-elemento no se genera y no aparece nada.

RESPUESTA 34: Falso. Las transiciones solo suavizan cambios entre dos estados disparados por algo (como:hover); no se ejecutan solas ni en bucle. Para eso están las animaciones.

RESPUESTA 35: Verdadero. Sin `<meta name="viewport" content="width=device-width, initial-scale=1">`, el celular simula ~980px de ancho y las condiciones de ancho nunca se cumplen como esperas.

Preguntas abiertas

RESPUESTA 36:

```
.comprar {
  background-color: #e94560;
  color: white;
  padding: 14px 35px;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  font-size: 16px;
  transition: background-color 0.3s ease;
}

.comprar:hover {
  background-color: #1a1a2e;
}
```

La transición va en el estado base para que suavice tanto al entrar como al salir el mouse.

RESPUESTA 37:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <link rel="stylesheet" href="estilos.css">
</head>
<body>
  <p>Hola mundo</p>
</body>
</html>
```

```
/* estilos.css */
p {
  color: #333333;
  font-family: Arial, sans-serif;
}
```

El <link> va en el <head> con rel="stylesheet" y el href apuntando al archivo.

RESPUESTA 38: `display: none` elimina al elemento del flujo por completo: no se ve, no ocupa espacio y los demás elementos se acomodan como si no existiera. `visibility: hidden` lo hace invisible pero conserva su espacio: queda un hueco vacío del tamaño del elemento. Con `none` el layout se reorganiza; con `hidden` no.

RESPUESTA 39:

```
.caja {  
  width: 400px;  
  margin: 0 auto;  
}
```

El `auto` en los márgenes laterales reparte el espacio sobrante en partes iguales a izquierda y derecha, centrando el bloque. Solo funciona en elementos de bloque con ancho definido menor al del contenedor.

RESPUESTA 40:

```
.tarjeta {  
  position: relative;  
  width: 280px;  
  padding: 20px;  
  border: 1px solid #ddd;  
  border-radius: 10px;  
}  
  
.insignia {  
  position: absolute;  
  top: 10px;  
  right: 10px;  
  background-color: #e94560;  
  color: white;  
  padding: 4px 12px;  
  border-radius: 12px;  
  font-size: 12px;  
  font-weight: bold;  
}
```

El `position: relative` en la tarjeta es obligatorio: sin él, la insignia se posicionaría respecto a la página completa.

RESPUESTA 41:

```

@keyframes pulso {
  0%, 100% { transform: scale(1); }
  50% { transform: scale(1.2); }
}

.elemento {
  animation: pulso 2s ease-in-out infinite;
}

```

Agrupar 0% y 100% con el mismo valor garantiza un ciclo sin saltos. Se anima transform (eficiente) en vez de width/height.

RESPUESTA 42:

```

.galeria {
  display: grid;
  grid-template-columns: 1fr;
  gap: 15px;
}

@media (min-width: 900px) {
  .galeria {
    grid-template-columns: repeat(3, 1fr);
  }
}

```

Mobile-first: la base es 1 columna (celular) y la media query con min-width mejora a 3 columnas en pantallas grandes.

RESPUESTA 43:

```
:root {
  --color-primario: #1a1a2e;
  --color-acento: #e94560;
  --espaciado: 20px;
}

.tarjeta {
  border-top: 4px solid var(--color-primario);
  padding: var(--espaciado);
  border-radius: 10px;
  background-color: white;
}

.tarjeta .btn {
  background-color: var(--color-acento);
  color: white;
  padding: 10px var(--espaciado);
  border: none;
  border-radius: 6px;
}
```

Cambiar `--color-acento` en `:root` actualiza todos los usos a la vez.

RESPUESTA 44:

```

.datos {
  border-collapse: collapse;
  width: 100%;
  font-family: Arial, sans-serif;
}

.datos th {
  background-color: #1a1a2e;
  color: white;
  text-align: left;
  padding: 12px 15px;
}

.datos td {
  padding: 10px 15px;
  border-bottom: 1px solid #e0e0e0;
}

.datos tbody tr:nth-child(even) {
  background-color: #f5f5f5;
}

.datos tbody tr:hover {
  background-color: #e3f2fd;
}

```

Los cinco ingredientes: collapse, encabezado contrastante, padding generoso, cebreado y hover.

RESPUESTA 45: Faltan las llaves del bloque y el punto y coma en `color: red`. Versión corregida:

```

.aviso {
  color: red;
  background: yellow;
}

```

Sin llaves el navegador no sabe dónde empieza y termina la regla; sin punto y coma, la segunda declaración se pega a la primera y se rompe. (Nota de diseño: texto rojo sobre fondo amarillo tiene contraste pobre; en un caso real convendría un rojo más oscuro o un fondo más suave.)

RESPUESTA 46:

```
.menu-superior {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  background-color: #1a1a2e;
  padding: 15px 20px;
  z-index: 100;
}

.contenido {
  padding-top: 70px; /* compensa la altura del menú fijo */
}
```

El menú sale del flujo con fixed, así que el contenido necesita padding-top (ajustado a la altura real del menú) para no quedar escondido debajo. El z-index 100 lo mantiene sobre el contenido al hacer scroll.

RESPUESTA 47: El texto será verde. Las tres reglas aplican al mismo párrafo: p tiene 1 punto de especificidad, .rojo tiene 10 y #texto tiene 100. Gana el id, así que aplica color: green.

RESPUESTA 48:

```
.email {
  padding: 12px 15px;
  font-size: 16px;
  font-family: inherit;
  border: 1px solid #ccc;
  border-radius: 8px;
  width: 100%;
  box-sizing: border-box;
}

.email:focus {
  outline: none;
  border-color: #4a90d9;
  box-shadow: 0 0 0 3px rgba(74, 144, 217, 0.25);
}
```

Se quita el outline por defecto pero se reemplaza con borde de color + anillo de box-shadow, manteniendo el foco visible para usuarios de teclado (accesibilidad).

RESPUESTA 49: Pasos del enfoque mobile-first: 1) Agregar el meta viewport al HTML. 2) Escribir primero los estilos base para celular: una columna, tipografía legible, botones de buen tamaño táctil. 3) Usar unidades relativas (% , rem, vw) y max-width: 100% en imágenes para que todo sea fluido desde la base. 4) Probar y encontrar los puntos donde el diseño "se rompe" al agrandar la pantalla. 5) Agregar media queries con min-width en esos puntos para mejorar progresivamente (2 columnas en tablet, más columnas y layouts amplios en escritorio). 6) Convertir la navegación a patrón hamburguesa en el celular. 7) Probar en un celular real, no solo redimensionando la ventana de la computadora.

RESPUESTA 50:

```
<article class="card-producto">
  
  <div class="card-body">
    <h3 class="card-nombre">Audífonos Pro</h3>
    <p class="card-precio">$1,299</p>
    <button class="card-btn">Comprar</button>
  </div>
</article>
```

```
:root {
  --acento: #e94560;
  --oscuro: #1a1a2e;
  --radio: 14px;
}

.card-producto {
  max-width: 280px;
  background: white;
  border-radius: var(--radio);
  overflow: hidden;
  box-shadow: 0 3px 10px rgba(0, 0, 0, 0.1);
  font-family: Arial, sans-serif;
  transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.card-producto:hover {
  transform: translateY(-8px);
```

```

    box-shadow: 0 14px 30px rgba(0, 0, 0, 0.16);
}

.card-img {
  width: 100%;
  height: 190px;
  object-fit: cover;
  display: block;
  max-width: 100%;
}

.card-body { padding: 1.2rem; }

.card-nombre { margin: 0 0 0.5rem 0; color: var(--oscuro); }

.card-precio {
  font-size: 1.4rem;
  font-weight: bold;
  color: var(--acento);
  margin: 0 0 1rem 0;
}

.card-btn {
  width: 100%;
  padding: 0.85rem;
  background: var(--oscuro);
  color: white;
  border: none;
  border-radius: 8px;
  font-weight: bold;
  cursor: pointer;
  transition: background-color 0.3s;
}

.card-btn:hover { background: var(--acento); }

```

Usa variables para colores y radio, `max-width: 100% + object-fit: cover` para imagen responsive sin deformarse, `hover de elevación con transición`, y `overflow: hidden + border-radius` para que la imagen respete las esquinas redondeadas.

Calificación

Cuenta 2 puntos por respuesta correcta. 90-100: excelente, dominas CSS3. 80-89: aprobado, repasa los temas donde fallaste. Menos de 80: vuelve a las unidades correspondientes y reintenta el examen en una semana. Recuerda: el objetivo no es la calificación, es que puedas construir sitios reales con confianza.

Colibrí Editorial EGA

Curso Completo de CSS3

Curso Completo de CSS3 es la versión escrita y extendida del curso de 30 clases de Lucas EGA Academy. En 10 unidades —cada una con ampliación de casos de estudio— explica qué hace cada propiedad y por qué existe, desde selectores y el modelo de caja hasta flexbox, grid, animaciones y diseño responsive. Incluye ejemplos, ejercicios con soluciones, apéndices de referencia y un examen final de 50 preguntas con respuestas explicadas.

lucasegaacademy.com