Manual de Usuario

Lenguaje CSS

CAPÍTULO 1: Introducción al Lenguaje CSS

1.1 ¿Qué es CSS?

CSS (Cascading Style Sheets) es un lenguaje que define la **presentación visual** de los documentos HTML. Permite separar el **contenido (HTML)** del **diseño (CSS)**, controlando aspectos como colores, tipografía, márgenes, posiciones, animaciones, y mucho más.

Ejemplo básico:

```
<style>
  h1 { color: blue; text-align: center; }
  p { font-size: 18px; }
</style>
```

1.2 ¿Para qué sirve CSS?

- Dar **estilo visual** a los elementos HTML.
- Adaptar el diseño a **diferentes pantallas o dispositivos** (responsividad).
- Crear **animaciones y efectos visuales**.
- Mantener un **diseño coherente y reutilizable** en todo el sitio web.

1.3 Estructura de una Regla CSS

Cada regla está formada por un **selector** y un **bloque de declaraciones**.

```
selector {
  propiedad: valor;
}
```

Ejemplo:

```
p {
  color: red;
  font-size: 16px;
}
```

Selector: indica a qué elemento HTML se aplican los estilos.

Propiedad: especifica qué aspecto del elemento se modificará.

Valor: define cómo se aplicará la propiedad.

1.4 Tipos de Selectores

Tipo	Ejemplo	Descripción
Etiqueta	`p {}`	Aplica a todos los elementos `<p>`
Clase	`.boton {}`	Aplica a elementos con clase “boton”
ID	`#principal {}`	Aplica al elemento con ID “principal”
Atributo	`input[type=text] {}`	Aplica a elementos que cumplen el atributo
Universal	`* {}`	Aplica a todos los elementos

1.5 Cómo se aplica el CSS

Método	Descripción	Ejemplo
En línea	En el mismo elemento HTML.	`<p style="color:red;">Texto</p>`
Interno	Dentro de una etiqueta `<style>` en el HTML.	`<style>p{color:red;}</style>`
Externo	En un archivo `.css` enlazado.	`<link rel="stylesheet" href="estilos.css">`

Recomendación: siempre que sea posible, usa **hojas de estilo externas**, para mantener el código limpio y reutilizable.

CAPÍTULO 2: Conceptos Fundamentales de CSS

2.1 El Modelo de Caja (Box Model)

Cada elemento HTML se representa como una **caja rectangular**, compuesta por:

1. **Contenido (content):** texto o imagen dentro del elemento.
2. **Relleno (padding):** espacio entre el contenido y el borde.
3. **Borde (border):** línea que rodea el relleno.
4. **Margen (margin):** espacio externo entre el borde y otros elementos.

```
div {
  margin: 10px;
  border: 2px solid black;
  padding: 5px;
}
```

Nota: el tamaño total del elemento se calcula así:

Ancho total = width + padding + border + margin

2.2 Unidades de Medida

Tipo	Ejemplo	Descripción
Absolutas	`px`, `cm`, `mm`	No cambian con el tamaño de pantalla.
Relativas	`%`, `em`, `rem`, `vw`, `vh`	Se adaptan según el contexto.

Consejo: usa unidades **relativas** para sitios adaptables (responsive).

2.3 Colores en CSS

Puedes definir colores de varias maneras:

```
color: red;
color: #ff0000;
color: rgb(255, 0, 0);
color: hsl(0, 100%, 50%);
```

Tip: los valores RGBA y HSLA incluyen transparencia (`a` de *alpha*).

Ejemplo: `rgba(255, 0, 0, 0.5)` → rojo semitransparente.

2.4 Tipografía

Controla la apariencia del texto:

```
p {
  font-family: 'Arial', sans-serif;
  font-size: 16px;
  font-weight: bold;
  font-style: italic;
  text-align: justify;
}
```

Propiedad	Descripción	Ejemplo
font-family	Define la fuente	`'Roboto', sans-serif`
font-size	Tamaño del texto	`16px`
font-weight	Grosor	`bold`
font-style	Cursiva o normal	`italic`
text-align	Alineación	`center`

2.5 Bordes, Márgenes y Rellenos

Bordes:

```
border: 2px solid #000;
border-radius: 10px;
```

Rellenos y márgenes:

```
padding: 20px;
margin: 10px 15px;
```

Nota: puedes definir cada lado individualmente:

```
margin-top: 10px;
margin-right: 20px;
```

2.6 Fondos y Degradados

```
background-color: #e3f2fd;
background-image: linear-gradient(to right, blue, lightblue);
background-repeat: no-repeat;
background-size: cover;
```

2.7 Sombras y Opacidad

```
box-shadow: 3px 3px 5px rgba(0,0,0,0.4);
opacity: 0.8;
```

Usa `text-shadow` para sombras en texto:

```
text-shadow: 2px 2px 4px gray;
```

CAPÍTULO 3: Posicionamiento y Visualización de Elementos

3.1 Concepto de Posicionamiento

En CSS, el **posicionamiento** permite definir **dónde y cómo se muestra un elemento dentro del flujo del documento**. Por defecto, todos los elementos están en un flujo normal (uno debajo del otro), pero podemos alterar este comportamiento usando la propiedad `position`.

Tipos de posicionamiento principales:

Tipo	Descripción	Ejemplo
static	Posición por defecto. El elemento sigue el flujo natural del documento.	`position: static;`
relative	Permite mover el elemento respecto a su posición original.	`position: relative; top: 10px;`
absolute	Se posiciona respecto al primer contenedor con posición distinta de `static`.	`position: absolute; top: 50px; left: 30px;`
fixed	Se fija en la ventana del navegador, aunque se haga scroll.	`position: fixed; bottom: 0;`
sticky	Combina comportamiento de `relative` y `fixed`. Se mantiene en su lugar hasta cierto punto del scroll.	`position: sticky; top: 0;`

Nota práctica:

- Usa `relative` para pequeños ajustes visuales.
- Usa `absolute` para colocar elementos dentro de contenedores específicos.
- Usa `fixed` para menús o botones flotantes.

3.2 Propiedades de Coordenadas

Cuando un elemento tiene una posición distinta de `static`, puedes moverlo usando:

```
top: 20px;
left: 50px;
right: 0;
bottom: 10px;
```

Estas propiedades indican **distancia desde los bordes del contenedor** (o del viewport si está en posición fija).

Advertencia:

Un valor `position: absolute` **ignora márgenes** y puede superponerse con otros elementos.

3.3 Control de Superposición: `z-index`

Define qué elemento se muestra **encima o debajo** de otros.

```
.caja1 {
  position: absolute;
  z-index: 2;
}
.caja2 {
  position: absolute;
  z-index: 1;
}
```

Regla:

Cuanto mayor el valor de `z-index`, **más arriba** se muestra el elemento.

3.4 Propiedad `display`

Controla **cómo se renderiza** un elemento en el flujo del documento.

Valor	Descripción	Ejemplo
`block`	Ocupa todo el ancho disponible. Inicia en nueva línea.	`<div>, <p>, <section>`
`inline`	Ocupa solo el espacio del contenido. No inicia nueva línea.	`<span>, <a>`
`inline-block`	Similar a `inline`, pero permite ajustar ancho y alto.	`display: inline-block;`
`none`	Oculta completamente el elemento (no ocupa espacio).	`display: none;`
`flex`	Activa el modelo de caja flexible (ver Capítulo 4).	`display: flex;`
`grid`	Activa el modelo de cuadrícula (ver Capítulo 4).	`display: grid;`

Tip: Usa `inline-block` cuando necesites alinear elementos en fila sin usar Flexbox.

3.5 Propiedad `visibility`

Permite **ocultar un elemento sin eliminar su espacio** en el documento.

```
visibility: hidden;
```

Diferencia entre `display: none` y `visibility: hidden`:

Propiedad	Efecto visual	Ocupa espacio
`display: none`	No visible	No
`visibility: hidden`	Invisible	Sí

3.6 Desbordamiento de Contenido (`overflow`)

Controla qué sucede cuando el contenido excede el tamaño de su contenedor.

```
overflow: hidden;   /* Oculta el exceso */
```

```
overflow: scroll;   /* Muestra barras de desplazamiento */
overflow: auto;     /* Añade scroll solo si es necesario */
```

Consejo:

Para manejar scroll en un contenedor específico, usa:

```
overflow-y: scroll;
overflow-x: hidden;
```

3.7 Propiedad `float` y `clear`

float: permite alinear un elemento a la izquierda o derecha, haciendo que el texto fluya alrededor. **clear:** evita que otros elementos floten a su lado.

```
img {
  float: right;
  margin: 10px;
}

p {
  clear: both;
}
```

Atención:

El uso excesivo de `float` está en desuso; se recomienda emplear **Flexbox o Grid**.

3.8 Propiedad `overflow-wrap` y `word-break`

Estas propiedades controlan el comportamiento del texto largo dentro de un contenedor.

```
overflow-wrap: break-word;
word-break: break-all;
```

Evitan que textos o URLs largas se salgan del contenedor.

3.9 Ocultamiento y Despliegue Condicional

Puedes combinar **transiciones** o **pseudo-clases** para mostrar u ocultar elementos.

```
.menu {
  display: none;
}
button:hover + .menu {
  display: block;
}
```

Esto se usa comúnmente en menús desplegables o tooltips.

3.10 Ejemplo Práctico: Caja Fija y Elemento Flotante

```
<div class="header">Encabezado fijo</div>
<div class="contenido">
  <img src="foto.jpg" class="imagen" alt="Foto ejemplo">
  <p>Texto con imagen flotante a la derecha...</p>
</div>
.header {
  position: fixed;
  top: 0;
  width: 100%;
  background-color: #333;
  color: white;
  text-align: center;
  padding: 10px;
}

.imagen {
  float: right;
  margin: 10px;
}

.contenido {
  margin-top: 60px;
}
```

Resultado:

El encabezado permanece fijo mientras el usuario se desplaza, y la imagen flota a la derecha del texto.

3.11 Posicionamiento Avanzado: `clip-path` y `object-fit`

clip-path: recorta un elemento según una forma.

```
.clip {
  clip-path: circle(50% at 50% 50%);
}
```

object-fit: controla cómo se ajustan imágenes o videos en su contenedor.

```
img {
  width: 100%;
```

```
  height: 300px;
  object-fit: cover;
}
```

Muy útil para **galerías responsivas** y **interfaces modernas**.

CAPÍTULO 4: Diseño Moderno con Flexbox y Grid Layout

4.1 Introducción

A medida que los sitios web se vuelven más complejos, el diseño y la disposición de los elementos (llamado **layout**) requieren herramientas potentes y flexibles.

CSS ofrece dos sistemas modernos para este propósito:

1. **Flexbox (Flexible Box Layout):** ideal para diseños **unidimensionales** (una fila o columna).
2. **Grid Layout:** ideal para diseños **bidimensionales** (filas y columnas simultáneamente).

Consejo:

Usa **Flexbox** para estructuras internas (por ejemplo, botones, barras de navegación) y **Grid** para estructuras generales de página.

PARTE I: FLEXBOX

4.2 ¿Qué es Flexbox?

Flexbox (Flexible Box Layout) es un modelo de diseño que permite organizar, alinear y distribuir los elementos dentro de un contenedor de forma dinámica, adaptándose automáticamente al tamaño de la pantalla.

Para activarlo, se debe aplicar:

```
display: flex;
```

4.3 Estructura de un contenedor Flexbox

Contenedor flexible (Flex container): el elemento padre.

Ítems flexibles (Flex items): los elementos hijos directos que se distribuyen dentro del contenedor.

```
<div class="contenedor">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>
.contenedor {
  display: flex;
}
.item {
  background: #3498db;
  color: white;
  padding: 20px;
  margin: 10px;
}
```

4.4 Propiedades del contenedor Flexbox

Propiedad	Descripción	Ejemplo
flex-direction	Define la dirección del eje principal (fila o columna).	`flex-direction: row;`
justify-content	Alinea los elementos a lo largo del eje principal.	`justify-content: center;`
align-items	Alinea los elementos en el eje transversal.	`align-items: center;`
align-content	Controla el espacio entre filas múltiples.	`align-content: space-around;`
flex-wrap	Permite que los elementos se ajusten a varias líneas.	`flex-wrap: wrap;`
gap	Define el espacio entre elementos.	`gap: 10px;`

Consejo práctico:

`flex-direction: row-reverse;` o `column-reverse;` invierten el orden de los elementos.

4.5 Propiedades de los ítems flexibles

Propiedad	Descripción	Ejemplo
order	Cambia el orden visual del elemento.	`order: 2;`
flex-grow	Permite que un elemento crezca proporcionalmente.	`flex-grow: 1;`
flex-shrink	Controla cuánto puede reducirse.	`flex-shrink: 0;`
flex-basis	Define el tamaño inicial antes del ajuste.	`flex-basis: 200px;`
align-self	Alinea un ítem individualmente.	`align-self: flex-end;`

Ejemplo completo:

```
.item1 { flex-grow: 2; }   /* Crece el doble */
.item2 { flex-shrink: 0; } /* No se encoge */
.item3 { flex-basis: 150px; }
```

4.6 Ejemplo práctico: barra de navegación

```
<nav class="menu">
  <a href="#">Inicio</a>
  <a href="#">Servicios</a>
  <a href="#">Contacto</a>
</nav>
.menu {
  display: flex;
  justify-content: space-between;
```

```
  align-items: center;
  background-color: #34495e;
  padding: 10px 20px;
}
.menu a {
  color: white;
  text-decoration: none;
  padding: 10px;
}
.menu a:hover {
  background-color: #2ecc71;
}
```

Resultado:

Los enlaces se distribuyen horizontalmente con espacios equitativos, y se adaptan al ancho del contenedor.

4.7 Alineación avanzada

```
.contenedor {
  display: flex;
  justify-content: space-evenly; /* Distribuye igual espacio entre elementos */
  align-items: flex-start;       /* Alinea al inicio verticalmente */
}
```

Usa `justify-content` y `align-items` en conjunto para lograr alineaciones precisas.

PARTE II: GRID LAYOUT

4.8 ¿Qué es CSS Grid?

CSS Grid Layout permite crear **estructuras bidimensionales** (filas y columnas) de manera sencilla, ideal para diseñar maquetas completas de sitios web.

Para activarlo:

```
display: grid;
```

4.9 Estructura de un Grid

```
<div class="grid">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
</div>
```

```
.grid {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: auto auto;
  gap: 10px;
}
```

Cada **item** se ubica en una celda de la cuadrícula, pudiendo expandirse a varias filas o columnas.

4.10 Propiedades principales del contenedor Grid

Propiedad	Descripción	Ejemplo
grid-template-columns	Define el número y tamaño de las columnas.	`grid-template-columns: 1fr 2fr 1fr;`
grid-template-rows	Define el número y tamaño de las filas.	`grid-template-rows: auto 200px;`
gap	Espacio entre filas y columnas.	`gap: 15px;`
justify-items	Alinea los ítems horizontalmente.	`justify-items: center;`
align-items	Alinea los ítems verticalmente.	`align-items: stretch;`
grid-auto-rows / grid-auto-columns	Tamaño automático para elementos adicionales.	`grid-auto-rows: 100px;`

Tip:

Puedes usar `repeat()` para crear columnas dinámicamente:

```
grid-template-columns: repeat(3, 1fr);
```

4.11 Propiedades de los ítems Grid

Propiedad	Descripción	Ejemplo
grid-column	Define cuántas columnas ocupa.	`grid-column: 1 / 3;`
grid-row	Define cuántas filas ocupa.	`grid-row: 1 / 2;`
grid-area	Asigna nombre o posición específica.	`grid-area: header;`
justify-self / align-self	Alineación individual de un ítem.	`justify-self: end;`

4.12 Ejemplo práctico: estructura de sitio web

```
<div class="layout">
  <header>Encabezado</header>
  <nav>Menú</nav>
  <main>Contenido principal</main>
  <aside>Barra lateral</aside>
```

```
  <footer>Pie de página</footer>
</div>
.layout {
  display: grid;
  grid-template-areas:
    "header header"
    "nav main"
    "nav aside"
    "footer footer";
  grid-template-columns: 200px 1fr;
  gap: 10px;
}

header { grid-area: header; background: #3498db; }
nav { grid-area: nav; background: #2ecc71; }
main { grid-area: main; background: #f1c40f; }
aside { grid-area: aside; background: #e67e22; }
footer { grid-area: footer; background: #34495e; color: white; }
```

Resultado:

Un diseño de sitio completo con encabezado, menú, contenido, barra lateral y pie de página perfectamente organizados.

4.13 Unidades fr, auto y minmax()

Unidad	Descripción	Ejemplo
`fr`	Fracción del espacio disponible.	`grid-template-columns: 1fr 2fr;`
`auto`	Tamaño según el contenido.	`grid-template-rows: auto;`
`minmax()`	Define un rango mínimo y máximo.	`grid-template-columns: repeat(3, minmax(150px, 1fr));`

Ideal para diseños **responsivos**.

4.14 Grid responsivo con `auto-fit` y `auto-fill`

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

```
}
```

Resultado:

Los elementos se reorganizan automáticamente según el ancho de la pantalla.

4.15 Combinar Flexbox y Grid

En proyectos reales, se pueden combinar ambos sistemas:

- **Grid:** para la estructura global del sitio.
- **Flexbox:** para la organización interna de secciones o componentes.

CAPÍTULO 5: Animaciones y Transiciones

5.1 Introducción

CSS no solo controla el aspecto visual de los elementos, sino también **cómo cambian esos estilos en el tiempo**. Las **animaciones y transiciones** permiten crear efectos suaves, interactivos y profesionales que mejoran la experiencia del usuario.

- Las **transiciones** cambian una propiedad de un estado a otro.
- Las **animaciones** permiten definir **múltiples etapas de movimiento o cambio** con control total.

Ejemplo visual:

Un botón que cambia de color suavemente al pasar el mouse o una caja que se desplaza de izquierda a derecha.

PARTE I: TRANSICIONES

5.2 ¿Qué es una transición?

Una **transición** define **cómo y durante cuánto tiempo** cambia una propiedad CSS cuando ocurre un evento (por ejemplo, `hover`, `focus` o `active`).

Estructura básica:

```
selector {
  transition: propiedad duración tipo_retraso;
}
```

Ejemplo:

```
button {
  background-color: #3498db;
  transition: background-color 0.3s ease;
}
button:hover {
  background-color: #2ecc71;
}
```

Resultado:

El color de fondo cambia gradualmente en 0.3 segundos cuando el usuario pasa el cursor sobre el botón.

5.3 Propiedades de transición

Propiedad	Descripción	Ejemplo
transition-property	Especifica qué propiedad CSS se animará.	`transition-property: width;`
transition-duration	Define la duración del efecto.	`transition-duration: 0.5s;`
transition-timing-function	Controla la curva de aceleración (cómo cambia la velocidad).	`transition-timing-function: ease-in-out;`

transition-delay	Tiempo de espera antes de comenzar la transición.	`transition-delay: 1s;`

5.4 Funciones de tiempo (Timing functions)

Controlan cómo varía la velocidad del cambio en el tiempo.

Valor	Descripción
`linear`	Velocidad constante
`ease`	Aceleración y desaceleración suaves
`ease-in`	Comienza lento y acelera
`ease-out`	Comienza rápido y desacelera
`ease-in-out`	Lento al inicio y al final
`cubic-bezier(x1, y1, x2, y2)`	Curva personalizada

Ejemplo:

```
div {
  transition: transform 1s cubic-bezier(0.68, -0.55, 0.27, 1.55);
}
```

5.5 Transiciones múltiples

Puedes aplicar transiciones a varias propiedades al mismo tiempo:

```
.box {
  transition: background-color 0.5s ease, transform 0.5s ease;
}
.box:hover {
  background-color: #e74c3c;
  transform: scale(1.2);
}
```

Resultado:

La caja aumenta su tamaño y cambia de color de manera simultánea y fluida.

5.6 Ejemplo práctico: Tarjeta interactiva

```
<div class="tarjeta">
  <h3>Información</h3>
  <p>Haz hover para ver el efecto</p>
</div>
.tarjeta {
  background: #ecf0f1;
  border-radius: 10px;
```

```
  padding: 20px;
  text-align: center;
  transition: transform 0.5s ease, box-shadow 0.5s ease;
}
.tarjeta:hover {
  transform: translateY(-10px);
  box-shadow: 0 10px 20px rgba(0,0,0,0.3);
}
```

Resultado:

Al pasar el mouse, la tarjeta se eleva suavemente generando sensación de profundidad.

PARTE II: ANIMACIONES

5.7 ¿Qué es una animación en CSS?

Una **animación** permite definir **cambios graduales** en las propiedades CSS mediante una regla especial llamada `@keyframes`.

Estructura básica:

```
@keyframes nombre_animacion {
  from { propiedad: valor_inicial; }
  to { propiedad: valor_final; }
}
```

Luego se aplica al elemento:

```
selector {
  animation: nombre_animacion duración tipo_repetición;
}
```

5.8 Ejemplo básico de animación

```
<div class="caja"></div>
@keyframes mover {
  from { transform: translateX(0); }
  to { transform: translateX(200px); }
}

.caja {
  width: 100px;
  height: 100px;
  background: #3498db;
  animation: mover 2s infinite alternate;
```

```
}
```

Resultado:

La caja se mueve de izquierda a derecha de forma continua.

5.9 Propiedades de animación

Propiedad	Descripción	Ejemplo
animation-name	Nombre definido en `@keyframes`.	`animation-name: mover;`
animation-duration	Duración del ciclo de animación.	`animation-duration: 2s;`
animation-timing-function	Curva de velocidad.	`animation-timing-function: ease-in-out;`
animation-delay	Retraso antes de comenzar.	`animation-delay: 1s;`
animation-iteration-count	Número de repeticiones (`infinite` = sin fin).	`animation-iteration-count: infinite;`
animation-direction	Dirección del movimiento (`normal`, `reverse`, `alternate`).	`animation-direction: alternate;`
animation-fill-mode	Mantiene estilos al terminar la animación (`forwards`, `backwards`, `both`).	`animation-fill-mode: forwards;`

5.10 Animaciones con múltiples etapas

Puedes definir más de dos pasos usando porcentajes:

```
@keyframes colores {
  0% { background: red; }
  50% { background: yellow; }
  100% { background: green; }
}
div {
  animation: colores 3s infinite;
}
```

Resultado:

El color del fondo cambia gradualmente de rojo → amarillo → verde.

5.11 Animaciones combinadas

Puedes aplicar más de una animación al mismo elemento:

```
div {
  animation: mover 2s infinite alternate, colores 4s infinite linear;
}
```

Tip: Combinar animaciones permite crear efectos complejos con poco código.

5.12 Ejemplo práctico: Rebote y rotación

```
<div class="bola"></div>
@keyframes rebotar {
  0%, 100% { transform: translateY(0); }
  50% { transform: translateY(-80px); }
}

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

.bola {
  width: 80px;
  height: 80px;
  background: #e74c3c;
  border-radius: 50%;
  animation: rebotar 1s ease-in-out infinite, girar 2s linear infinite;
}
```

Resultado:

Una bola que rebota y gira al mismo tiempo, generando movimiento continuo y natural.

5.13 Control de animaciones con clases y eventos

Puedes activar o detener animaciones con clases CSS:

```
.pulso {
  animation: latir 1s infinite;
}

@keyframes latir {
  0%, 100% { transform: scale(1); }
  50% { transform: scale(1.1); }
}
<button onclick="this.classList.toggle('pulso')">Latir</button>
```

Aunque el ejemplo usa un `onclick`, también puedes hacerlo con pseudo-clases como `:hover`.

5.14 Animaciones con degradado y opacidad

```
@keyframes brillar {
  0% { background: linear-gradient(90deg, #ff0080, #ff8c00); opacity: 0.7; }
  50% { background: linear-gradient(90deg, #ff8c00, #40e0d0); opacity: 1; }
  100% { background: linear-gradient(90deg, #40e0d0, #ff0080); opacity: 0.7; }
}
.barra {
  width: 200px;
  height: 30px;
  animation: brillar 3s infinite;
}
```

Resultado:

Una barra que cambia gradualmente de color y brillo, simulando un efecto de energía o carga.

5.15 Buenas prácticas con animaciones

- **Usa animaciones para guiar la atención**, no para distraer.
- **Evita abusar de `infinite`** en animaciones pesadas.
- **Usa `will-change`** en propiedades animadas para mejorar el rendimiento:

```
.elemento { will-change: transform, opacity; }
```

- **Combina transiciones y animaciones**: Las transiciones son ideales para interacciones simples; las animaciones, para secuencias complejas.

CAPÍTULO 6: Diseño Responsivo y Media Queries

6.1 ¿Qué es el Diseño Responsivo?

El **Diseño Responsivo (Responsive Design)** es una técnica que permite que las páginas web **se adapten automáticamente** al tamaño y orientación de la pantalla del dispositivo donde se visualizan.

El objetivo es que el usuario tenga **una experiencia óptima** sin importar si usa un monitor grande o un teléfono pequeño.

6.2 Principios del Diseño Responsivo

1. **Diseño fluido:** los elementos usan tamaños relativos (`%`, `em`, `rem`, `vh`, `vw`) en lugar de valores fijos (`px`).
2. **Imágenes adaptables:** las imágenes deben escalar correctamente según el ancho del contenedor.
3. **Media Queries:** reglas condicionales que aplican estilos diferentes según el ancho de la pantalla.
4. **Diseños modulares:** estructuras que pueden reorganizarse fácilmente (usando **Flexbox** o **Grid**).

6.3 Unidades relativas para diseño adaptable

Unidad	Descripción	Ejemplo
`%`	Porcentaje del contenedor padre	`width: 80%;`
`em`	Tamaño relativo a la fuente del elemento	`margin: 2em;`
`rem`	Relativo a la fuente raíz (`html`)	`font-size: 1.2rem;`
`vh` / `vw`	Porcentaje del alto o ancho de la ventana	`height: 50vh;`

Consejo: usa unidades relativas para textos, márgenes y anchos, así el diseño se ajusta automáticamente.

6.4 Imágenes Responsivas

Las imágenes deben escalar sin deformarse ni desbordar su contenedor.

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

Resultado:

La imagen se adapta al ancho del contenedor manteniendo sus proporciones.

6.5 ¿Qué son las Media Queries?

Las **Media Queries** son condiciones que permiten aplicar estilos específicos según el tamaño de la pantalla, la orientación o incluso el tipo de dispositivo.

Estructura básica:

```
@media (condición) {
  /* Estilos aplicados cuando se cumple la condición */
}
```

Ejemplo:

```
@media (max-width: 768px) {
  body {
    background-color: lightgray;
  }
}
```

Resultado:

Cuando el ancho de la pantalla sea menor o igual a 768px, el fondo se volverá gris.

6.6 Tipos de Media Queries más usados

Condición	Descripción	Ejemplo
`max-width`	Aplica estilos cuando el ancho es menor o igual al valor indicado	`@media (max-width: 600px)`
`min-width`	Aplica estilos cuando el ancho es mayor o igual al valor indicado	`@media (min-width: 1024px)`
`orientation`	Detecta si el dispositivo está en modo **portrait** (vertical) o **landscape** (horizontal)	`@media (orientation: landscape)`
`prefers-color-scheme`	Detecta modo oscuro o claro del sistema	`@media (prefers-color-scheme: dark)`

6.7 Ejemplo práctico: Tres niveles de diseño

```
/* Computadoras */
@media (min-width: 1025px) {
  .contenedor {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
  }
}

/* Tablets */
@media (max-width: 1024px) and (min-width: 601px) {
  .contenedor {
    display: grid;
    grid-template-columns: repeat(2, 1fr);
```

```
  }
}

/* Teléfonos */
@media (max-width: 600px) {
  .contenedor {
    display: block;
  }
}
```

Resultado:

- En pantallas grandes → tres columnas.
- En tablets → dos columnas.
- En móviles → una sola columna.

6.8 Breakpoints comunes

Los **breakpoints** son los puntos de cambio de diseño más utilizados en desarrollo web:

Dispositivo	Ancho máximo recomendado
Teléfono pequeño	480px
Teléfono grande	600px
Tablet vertical	768px
Tablet horizontal	1024px
Laptop / Escritorio	1280px
Pantalla grande	1440px o más

Consejo:

Diseña **primero para móviles** (“mobile first”), luego amplía hacia pantallas mayores.

6.9 Enfoque "Mobile First"

Este enfoque consiste en **diseñar primero para pantallas pequeñas** y agregar estilos adicionales conforme el ancho aumenta.

Ejemplo:

```
/* Estilos base: móviles */
.card {
  display: block;
}

/* Estilos para pantallas más grandes */
```

```
@media (min-width: 768px) {
  .card {
    display: flex;
  }
}
```

Ventaja:

El código es más ligero y escalable; se prioriza el rendimiento en móviles.

6.10 Uso combinado con Flexbox y Grid

Ejemplo: Ajuste automático con Flexbox

```
.contenedor {
  display: flex;
  flex-wrap: wrap;
}
.item {
  flex: 1 1 300px;
}
```

Los elementos se reorganizan automáticamente según el ancho disponible.

Ejemplo: Rejilla adaptable con Grid

```
.grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
  gap: 10px;
}
```

La cuadrícula se ajusta automáticamente sin necesidad de muchas Media Queries.

6.11 Orientación del dispositivo

Puedes aplicar estilos según si el dispositivo está en **modo vertical (portrait)** o **horizontal (landscape):**

```
@media (orientation: portrait) {
  .fondo {
    background-color: lightblue;
  }
}
@media (orientation: landscape) {
  .fondo {
    background-color: lightcoral;
  }
}
```

6.12 Ejemplo completo: Página responsiva

```
<header>Encabezado</header>
<main>Contenido principal</main>
<aside>Barra lateral</aside>
<footer>Pie de página</footer>
body {
  display: grid;
  grid-template-areas:
    "header"
    "main"
    "aside"
    "footer";
  gap: 10px;
}

/* Tablet y escritorio */
@media (min-width: 768px) {
  body {
    grid-template-areas:
      "header header"
      "main aside"
      "footer footer";
    grid-template-columns: 2fr 1fr;
  }
}
```

Resultado:

En móvil, los bloques aparecen uno debajo del otro; en pantallas grandes, el contenido y la barra lateral se muestran en columnas.

6.13 Detección de modo oscuro

CSS también puede adaptarse al **modo oscuro del sistema operativo**:

```
@media (prefers-color-scheme: dark) {
  body {
    background-color: #121212;
    color: #ffffff;
  }
}
```

Consejo:

Permite ofrecer una experiencia más agradable para los usuarios que prefieren modo oscuro.

6.14 Buenas prácticas en diseño responsivo

- **Diseña pensando primero en móviles.**
- **Usa unidades relativas** (`%`, `em`, `rem`, `vh`, `vw`).
- **Evita anchos fijos** en píxeles.
- **Usa imágenes optimizadas y escalables.**
- **Prueba tu sitio en varios dispositivos.**
- **Combina Flexbox y Grid para diseños fluidos.**

Herramienta recomendada:

Usa el **modo de desarrollador del navegador (F12)** → opción **"Toggle Device Toolbar"** para probar diferentes resoluciones.

CAPÍTULO 7: Variables, Pseudo-clases y Pseudo-elementos

7.1 Introducción

En CSS moderno, existen mecanismos que permiten **personalizar el comportamiento y apariencia** de los elementos sin modificar el HTML.

Entre los más importantes están:

- **Variables CSS (Custom Properties)** → permiten reutilizar valores.
- **Pseudo-clases** → aplican estilos según el **estado o interacción** del usuario.
- **Pseudo-elementos** → permiten **crear o modificar partes específicas** de un elemento, sin alterar el contenido original.

PARTE I: VARIABLES CSS

7.2 ¿Qué son las Variables CSS?

Las **variables CSS**, también llamadas **Custom Properties**, permiten definir valores reutilizables (como colores, tamaños o fuentes) que pueden cambiarse fácilmente desde un único lugar.

Se declaran con el prefijo `--` y se usan con la función `var()`.

Ejemplo:

```
:root {
  --color-principal: #3498db;
  --color-secundario: #2ecc71;
  --fuente-base: 'Roboto', sans-serif;
}
body {
  background-color: var(--color-principal);
  font-family: var(--fuente-base);
}
button {
  background: var(--color-secundario);
}
```

Ventaja:

Cambiar el valor de una variable afecta a todos los elementos que la usan, sin modificar cada regla.

7.3 Ámbito de las variables

- Si se declaran en `:root`, son **globales** (disponibles en todo el documento).
- Si se declaran dentro de un selector, son **locales** a ese elemento.

```
.caja {
  --color-fondo: pink;
  background: var(--color-fondo);
}
```

Tip: Puedes redefinir variables dentro de una Media Query para cambiar temas o colores por tamaño de pantalla.

7.4 Valores por defecto

Si una variable no está definida, puedes usar un valor alternativo:

```
div {
  color: var(--color-texto, black);
}
```

En este caso, si `--color-texto` no existe, se usará **black**.

7.5 Ejemplo práctico: modo claro/oscuro con variables

```
:root {
  --fondo: #ffffff;
  --texto: #000000;
}
[data-tema="oscuro"] {
  --fondo: #121212;
  --texto: #f2f2f2;
}
body {
  background-color: var(--fondo);
  color: var(--texto);
}
```

Resultado:

Al cambiar el atributo `data-tema` del `<body>` a `"oscuro"`, se actualizan todos los colores automáticamente.

PARTE II: PSEUDO-CLASES

7.6 ¿Qué son las Pseudo-clases?

Las **pseudo-clases** permiten aplicar estilos a un elemento **en un estado particular**, como cuando el usuario pasa el mouse, hace clic, enfoca un campo o visita un enlace.

Sintaxis:

```
selector:pseudo-clase {
  propiedad: valor;
}
```

7.7 Pseudo-clases de interacción del usuario

Pseudo-clase	Descripción	Ejemplo
`:hover`	Cuando el cursor pasa sobre el elemento	`button:hover { background: blue; }`
`:active`	Mientras se hace clic en el elemento	`button:active { transform: scale(0.95); }`
`:focus`	Cuando un elemento (input, botón) recibe foco	`input:focus { border-color: green; }`
`:visited`	Enlaces ya visitados	`a:visited { color: purple; }`
`:checked`	Checkbox o radio seleccionados	`input:checked { accent-color: red; }`
`:disabled`	Elementos deshabilitados	`button:disabled { opacity: 0.5; }`
`:focus-visible`	Foco visible solo en interacción por teclado	`button:focus-visible { outline: 2px solid blue; }`

Ejemplo práctico:

```
input:focus {
  border: 2px solid #2ecc71;
  outline: none;
}
```

7.8 Pseudo-clases estructurales

Permiten aplicar estilos según la **posición o relación** de los elementos dentro del DOM.

Pseudo-clase	Descripción	Ejemplo
`:first-child`	Primer hijo de un contenedor	`li:first-child { color: red; }`
`:last-child`	Último hijo	`li:last-child { color: blue; }`
`:nth-child(n)`	Elementos según su posición (número o patrón)	`li:nth-child(odd) { background: #eee; }`
`:nth-of-type(n)`	Según el tipo de etiqueta	`p:nth-of-type(2) { font-weight: bold; }`
`:not(selector)`	Excluye ciertos elementos	`p:not(.importante) { color: gray; }`
`:empty`	Elemento sin contenido	`div:empty { display: none; }`

Ejemplo:

```
tr:nth-child(even) { background-color: #f8f8f8; }
tr:nth-child(odd) { background-color: #ffffff; }
```

7.9 Pseudo-clases avanzadas

Pseudo-clase	Función
`:root`	Selecciona el elemento raíz del documento (`html`)
`:target`	Aplica estilo al elemento con el id del enlace actual (`#id`)
`:is()`	Simplifica selecciones múltiples (`:is(h1, h2, h3)`)
`:has()`	Selecciona elementos que contienen otros específicos *(soporte moderno)*
`:where()`	Similar a `:is()`, pero sin aumentar la especificidad

Ejemplo con `:is()`

```
:is(h1, h2, h3) {
  color: #2c3e50;
}
```

PARTE III: PSEUDO-ELEMENTOS

7.10 ¿Qué son los Pseudo-elementos?

Los **pseudo-elementos** permiten aplicar estilos o agregar contenido **a una parte específica** de un elemento sin modificar el HTML.

Sintaxis:

```
selector::pseudo-elemento {
  propiedad: valor;
}
```

7.11 Pseudo-elementos más utilizados

Pseudo-elemento	Descripción	Ejemplo
`::before`	Inserta contenido antes del elemento	`p::before { content: "→ "; }`
`::after`	Inserta contenido después del elemento	`p::after { content: " ✓"; }`
`::first-letter`	Selecciona la primera letra del texto	`p::first-letter { font-size: 2em; }`
`::first-line`	Aplica estilo a la primera línea del texto	`p::first-line { font-weight: bold; }`
`::selection`	Cambia el estilo del texto seleccionado	`::selection { background: yellow; color: black; }`

`::marker`	Estilo de los indicadores de lista (`<ul>`)	`li::marker { color: red; }`

7.12 Ejemplo práctico: lista decorada

```
<ul>
  <li>Inicio</li>
  <li>Servicios</li>
  <li>Contacto</li>
</ul>
li::before {
  content: "• ";
  color: #3498db;
  font-weight: bold;
}
```

Resultado:

Cada ítem de la lista mostrará un punto azul personalizado antes del texto.

7.13 Ejemplo práctico: etiqueta con pseudo-elemento

```
<h2 class="titulo">Bienvenidos</h2>
.titulo {
  position: relative;
  display: inline-block;
}
.titulo::after {
  content: "";
  position: absolute;
  bottom: 0;
  left: 0;
  width: 100%;
  height: 4px;
  background: linear-gradient(to right, #2ecc71, #3498db);
}
```

Resultado:

Se crea una línea de color degradado bajo el título sin agregar ningún elemento adicional en HTML.

7.14 Ejemplo práctico: mensaje emergente con `::after`

```
<button class="info">Más información</button>
.info {
```

```
  position: relative;
}
.info:hover::after {
  content: "Haz clic para ver más detalles";
  position: absolute;
  bottom: -30px;
  left: 0;
  background: #333;
  color: #fff;
  padding: 5px 10px;
  border-radius: 5px;
  font-size: 0.8em;
  white-space: nowrap;
}
```

Resultado:

Al pasar el mouse sobre el botón, aparece un pequeño cuadro informativo (tooltip).

7.15 Combinando pseudo-clases y pseudo-elementos

Puedes combinarlos para efectos más complejos:

```
a:hover::after {
  content: "";
  color: #2ecc71;
}
```

Ejemplo práctico:

Cada enlace mostrará una flecha verde al pasar el cursor.

CAPÍTULO 8: Buenas Prácticas y Optimización de CSS

8.1 Introducción

A medida que los proyectos crecen, el archivo CSS puede volverse extenso, repetitivo y difícil de mantener. Aplicar **buenas prácticas de organización, optimización y estandarización** permite:

- Reducir errores y redundancia.
- Mejorar el rendimiento de carga.
- Facilitar el trabajo en equipo y la reutilización del código.
- Garantizar un diseño coherente en toda la aplicación o sitio web.

PARTE I: ORGANIZACIÓN Y ESTRUCTURA DEL CÓDIGO

8.2 Orden lógico del código CSS

Organizar el código facilita su lectura y mantenimiento. Un orden recomendado de propiedades dentro de cada bloque es:

1. **Propiedades de caja:** `display`, `position`, `top`, `left`, `right`, `bottom`, `z-index`.
2. **Propiedades de tamaño:** `width`, `height`, `margin`, `padding`.
3. **Propiedades visuales:** `background`, `border`, `box-shadow`.
4. **Tipografía y texto:** `font-family`, `font-size`, `color`, `text-align`.
5. **Otros efectos:** `transition`, `animation`, `transform`.

Ejemplo ordenado:

```
.card {
  display: flex;
  position: relative;
  width: 300px;
  margin: 20px;
  padding: 15px;
  background: #fff;
  border-radius: 10px;
  font-family: 'Roboto', sans-serif;
  color: #333;
  transition: transform 0.3s ease;
}
```

Consejo: Mantén la **coherencia** en el orden y formato a lo largo de todos tus archivos CSS.

8.3 Comentarios y secciones

Usa comentarios para separar secciones del archivo:

```
/* ===== ENCABEZADO ===== */
header { ... }

/* ===== CUERPO PRINCIPAL ===== */
main { ... }

/* ===== PIE DE PÁGINA ===== */
footer { ... }
```

Esto ayuda a navegar rápidamente por archivos extensos.

8.4 Convenciones de nombres (naming)

a) Estilo semántico

Los nombres deben describir **la función**, no la apariencia visual.

- Malo: `.rojo`, `.grande`
- Bueno: `.alerta-error`, `.boton-principal`

b) Método BEM (Block Element Modifier)

BEM es una convención muy usada en proyectos grandes.

```
/* Bloque */
.menu {}

/* Elemento */
.menu__item {}

/* Modificador */
.menu__item--activo {}
```

Ventaja: evita conflictos de nombres y facilita la reutilización de componentes.

8.5 Separación del código

Regla general:

HTML estructura el contenido.

CSS controla el estilo.

JavaScript maneja la lógica.

Por tanto, **no mezcles estilos en línea** (`style="..."`) ni uses reglas internas salvo en casos de prueba o plantillas pequeñas.

Usa **hojas de estilo externas** (`.css`) y organízalas por componentes o secciones.

PARTE II: REUTILIZACIÓN Y MANTENIMIENTO

8.6 Uso de variables y clases reutilizables

Define **variables CSS** globales en `:root` para colores, fuentes o tamaños base.

```
:root {
  --color-primario: #3498db;
  --color-secundario: #2ecc71;
  --espaciado: 1rem;
}
```

Luego reutilízalas:

```
.boton {
  background: var(--color-primario);
  margin: var(--espaciado);
}
```

Así puedes cambiar todo el esquema de colores desde un solo punto.

8.7 Estilos base y reset

Los navegadores aplican estilos por defecto. Para mantener consistencia, usa un **reset CSS** o **normalize.css**:

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

Mejora la uniformidad entre navegadores.

8.8 Componentización del diseño

Divide tu CSS por **componentes reutilizables**:

- `header.css`
- `footer.css`
- `buttons.css`

- `cards.css`

Esto facilita la actualización y evita duplicación de código.

8.9 Evita la redundancia y la sobreespecificidad

Evita repetir estilos iguales:

```
h1 { color: blue; }
h2 { color: blue; }
```

Usa clases compartidas:

```
.titulo { color: blue; }
```

No abuses de selectores demasiado específicos como:

```
body div section article p span { ... }
```

Usa selectores simples y semánticos.

PARTE III: RENDIMIENTO Y OPTIMIZACIÓN

8.10 Minimización y compresión

Antes de publicar tu sitio, **minifica** los archivos CSS para reducir su tamaño:

- Quita espacios y comentarios.
- Herramientas recomendadas:
  - CSSNano
  - CleanCSS
  - Terser (en build tools)

```
npx cssnano estilos.css estilos.min.css
```

Reduce el tiempo de carga y el consumo de ancho de banda.

8.11 Carga eficiente de hojas de estilo

Coloca los archivos CSS en el `<head>` del documento para que se carguen antes de mostrar el contenido:

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

Consejo:

Usa el atributo `media` para cargar estilos condicionales:

```
<link rel="stylesheet" href="imprimir.css" media="print">
```

8.12 Combinar y diferir archivos

Para mejorar el rendimiento:

- **Combina** varios CSS en uno solo para reducir solicitudes HTTP.
- Usa `@import` con moderación, ya que puede ralentizar la carga.
- Si trabajas con frameworks o preprocesadores, usa **compiladores automáticos** (Webpack, Vite, Gulp).

8.13 Uso racional de animaciones

Las animaciones excesivas pueden degradar el rendimiento.

Sigue estas pautas:

- Usa `transform` y `opacity` en animaciones (son más eficientes).
- Evita animar propiedades como `width` o `top`.
- Usa `will-change` para anticipar cambios:

```
.elemento {
  will-change: transform, opacity;
}
```

8.14 Compatibilidad entre navegadores

Verifica siempre la compatibilidad de propiedades en Can I Use.

Ejemplo: algunas funciones avanzadas de `:has()` o `clip-path` no están soportadas en versiones antiguas.

Usa prefijos cuando sea necesario:

```
-webkit-border-radius: 10px; /* Safari */
-moz-border-radius: 10px;    /* Firefox */
border-radius: 10px;
```

8.15 Diseño accesible

CSS también contribuye a la **accesibilidad web (a11y)**.

- Mantén buen **contraste de colores** (usa contrast-checker).
- Evita textos muy pequeños.
- Usa `:focus-visible` para resaltar el foco al navegar con teclado.
- Evita animaciones con parpadeos intensos.

```
:focus-visible {
  outline: 2px solid #3498db;
}
```

Mejora la experiencia para todos los usuarios, incluyendo personas con discapacidad visual o motora.

8.16 Modo oscuro y temas dinámicos

Define esquemas de color variables y usa media queries o atributos para adaptarlos:

```
:root {
  --fondo: #ffffff;
  --texto: #000;
}

@media (prefers-color-scheme: dark) {
  :root {
    --fondo: #121212;
    --texto: #fff;
  }
}
```

Resultado:

El sitio se ajusta automáticamente al modo oscuro del sistema.

PARTE IV: HERRAMIENTAS Y BUENAS PRÁCTICAS PROFESIONALES

8.17 Usa preprocesadores CSS

Herramientas como **SASS**, **LESS** o **Stylus** permiten usar:

- Variables globales
- Anidación de selectores
- Mixins (plantillas de código)
- Importación modular

```
$color-principal: #3498db;

.boton {
  background: $color-principal;
  &:hover {
    background: darken($color-principal, 10%);
  }
}
```

Mejora la productividad y mantiene el código más organizado.

8.18 Usa herramientas de validación y análisis

- W3C CSS Validator para verificar errores de sintaxis.
- Stylelint para detectar inconsistencias.

- PurifyCSS o PurgeCSS para eliminar clases no utilizadas.

8.19 Versionado y documentación

- Usa **Git** para llevar control de versiones de los archivos CSS.
- Documenta las variables, clases y componentes clave.
- Mantén un archivo `README.md` o un "Guía de estilo" (Style Guide).

Ejemplo de comentario documental:

```
/* ========================================
   BOTONES PRINCIPALES
   Usados en formularios y menús
======================================== */
```