

Manual Completo de CSS: De Cero a Experto

Autor: Basdonax AI **Serie:** Fundamentos del Desarrollo Web

Índice General

(Este índice será generado con hipervínculos en el PDF final. Por ahora, sirve como estructura.)

- [Nivel 1: Básico \(Fundamentos Esenciales\)](#)
 - [Módulo 1: Introducción a CSS](#)
 - [Módulo 2: Selectores Fundamentales](#)
 - [Módulo 3: Estilos de Texto y Tipografía](#)
 - [Módulo 4: El Modelo de Caja \(Box Model\)](#)
- [Nivel 2: Intermedio \(Posicionamiento y Diseño Clásico\)](#)
 - [Módulo 5: Posicionamiento de Elementos](#)
 - [Módulo 6: Fondos y Visuales](#)
 - [Módulo 7: Pseudoclases y Pseudoelementos](#)
 - [Módulo 8: Diseño de Formularios y Tablas](#)
- [Nivel 3: Avanzado \(Layouts Modernos y Adaptativos\)](#)
 - [Módulo 9: Diseño Flexible con Flexbox](#)
 - [Módulo 10: Diseño de Cuadrícula con CSS Grid](#)
 - [Módulo 11: Diseño Responsivo \(Responsive Design\)](#)
- [Nivel 4: Experto \(Efectos, Rendimiento y Arquitectura\)](#)
 - [Módulo 12: Transiciones y Transformaciones](#)
 - [Módulo 13: Animaciones con Keyframes](#)
 - [Módulo 14: Selectores Avanzados y Funciones](#)

- [Módulo 15: Arquitectura y Buenas Prácticas](#)
 - [Proyecto Integrador](#)
 - [Apéndice](#)
-

Nivel 1: Básico (Fundamentos Esenciales)

Módulo 1: Introducción a CSS

1.1. ¿Qué es CSS y por qué lo necesitamos?

Teoría: CSS (*Cascading Style Sheets* u Hojas de Estilo en Cascada) es el lenguaje que utilizamos para describir la **presentación** de un documento escrito en un lenguaje de marcado, como **HTML**. Mientras que HTML se encarga de la **estructura** y el **contenido** (los sustantivos), CSS se encarga del **diseño**, los **colores**, las **fuentes** y la **disposición** (los adjetivos). Sin CSS, la web sería una colección de texto sin formato y enlaces.

Ejemplo Práctico: Imagina que tienes un título simple en HTML.

Código HTML (`index.html`):

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi Primera Página</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>¡Hola, Mundo!</h1>
</body>
</html>
```

Código CSS (`style.css`):

```
h1 {  
  color: #007bff; /* Color azul brillante */  
  font-family: 'Arial', sans-serif;  
  text-align: center;  
  border-bottom: 2px solid #007bff;  
  padding-bottom: 10px;  
}
```

Explicación: La regla CSS anterior toma el elemento `<h1>` y le aplica un color azul, una fuente Arial, lo centra y le añade una línea azul debajo. El resultado es un título visualmente atractivo, algo imposible de lograr solo con HTML.

1.2. Sintaxis de CSS (Selectores, Propiedades y Valores)

Teoría: Una regla CSS se compone de tres partes fundamentales: el **Selector**, la **Propiedad** y el **Valor**.

- **Selector:** Indica a qué elemento o grupo de elementos HTML se aplicará el estilo (ej: `h1`, `.clase`, `#id`).
- **Declaración:** Es el bloque que contiene las propiedades y valores, encerrado entre llaves `{}`.
- **Propiedad:** Es el aspecto visual que queremos cambiar (ej: `color`, `font-size`, `margin`).
- **Valor:** Es la especificación de cómo se cambiará esa propiedad (ej: `blue`, `20px`, `10px`).

Estructura de una Regla:

```
selector {  
  propiedad: valor; /* Declaración */  
  propiedad-otra: valor-otro; /* Otra declaración */  
}
```

Ejemplo Práctico:

```
p {  
  font-size: 16px; /* Propiedad: font-size, Valor: 16px */  
  line-height: 1.5; /* Propiedad: line-height, Valor: 1.5 */  
  margin: 10px 0;  
}
```

Explicación: El selector `p` apunta a todos los párrafos. Dentro de las llaves, definimos que el tamaño de la fuente será de 16 píxeles y el interlineado será de 1.5 veces el tamaño de la fuente.

1.3. Formas de incluir CSS (Inline, Interno y Externo)

Teoría: Existen tres métodos para vincular estilos CSS a un documento HTML. La elección depende de la necesidad, pero la **hoja de estilo externa** es la práctica recomendada para proyectos profesionales.

Método	Ubicación	Uso Recomendado	Prioridad
Inline	Directamente en la etiqueta HTML (<code>style="..."</code>)	Estilos muy específicos y puntuales (generalmente se evita).	Más Alta
Interno	Dentro de la etiqueta <code><style></code> en el <code><head></code> del HTML.	Páginas de prueba o estilos únicos para una sola página.	Media
Externo	En un archivo <code>.css</code> separado, vinculado con <code><link></code> .	Proyectos grandes y profesionales (permite reutilización y organización).	Más Baja

Ejemplo Práctico:

1. Estilo Inline:

```
<p style="color: red; font-weight: bold;">Este texto es rojo y negrita.</p>
```

2. Estilo Interno:

```
<head>
  <style>
    h2 {
      color: green;
    }
  </style>
</head>
```

3. Estilo Externo (Recomendado):

```
<head>
  <link rel="stylesheet" href="style.css">
</head>
```

(El contenido de `style.css` ya lo vimos en el ejemplo 1.1)

Explicación: El estilo **externo** es el más eficiente porque separa la estructura (HTML) de la presentación (CSS), haciendo el código más limpio y fácil de mantener.

1.4. El concepto de Cascada, Especificidad y Herencia

Teoría: El nombre *Hojas de Estilo en **Cascada*** no es casualidad. Cuando múltiples reglas CSS intentan estilizar el mismo elemento, el navegador debe decidir cuál aplicar. Esta decisión se basa en tres pilares:

1. **Cascada:** El orden en que se aplican las reglas. Las reglas que aparecen **últimas** en el código o en la hoja de estilo tienen **mayor peso**.
2. **Especificidad:** Es un cálculo numérico que determina qué selector es más “específico” o preciso. Un selector de **ID** es más específico que un selector de **Clase**, y este es más específico que un selector de **Tipo**.
3. **Herencia:** Algunas propiedades (como `color` o `font-family`) se heredan automáticamente de un elemento padre a sus hijos. Otras propiedades (como `margin` o `border`) no se heredan.

Ejemplo Práctico (Especificidad):

```
<p class="texto-rojo" id="parrafo-importante">Este es un párrafo.</p>
```

Código CSS:

```
/* 1. Selector de Tipo (Baja Especificidad) */  
p {  
    color: blue;  
}  
  
/* 2. Selector de Clase (Media Especificidad) */  
.texto-rojo {  
    color: red;  
}  
  
/* 3. Selector de ID (Alta Especificidad) */  
#parrafo-importante {  
    color: green;  
}
```

Resultado Esperado: El texto será de color **verde**. Aunque la regla `p { color: blue; }` aparece primero y la regla `.texto-rojo { color: red; }` aparece después, la regla con el selector de **ID** (`#parrafo-importante`) tiene la mayor especificidad y, por lo tanto, es la que se aplica.

Resumen del Módulo 1: Propiedades Aprendidas

Propiedad	Valor(es) Común(es)	Descripción
<code>color</code>	Nombre de color, Hexadecimal (<code>#...</code>), RGB, HSL	Define el color del texto.
<code>font-family</code>	Nombre de fuente (ej: <code>Arial</code> , <code>sans-serif</code>)	Define la tipografía del texto.
<code>text-align</code>	<code>left</code> , <code>right</code> , <code>center</code> , <code>justify</code>	Alinea el contenido de texto.
<code>border-bottom</code>	Grosor, estilo, color	Añade una línea en la parte inferior del elemento.
<code>padding-bottom</code>	Unidades (ej: <code>10px</code> , <code>1em</code>)	Espacio interno debajo del contenido.
<code>font-size</code>	Unidades (ej: <code>16px</code> , <code>1.2em</code>)	Define el tamaño de la fuente.
<code>line-height</code>	Número o unidad	Define el espacio entre líneas de texto (interlineado).
<code>margin</code>	Unidades	Espacio externo alrededor del elemento.

Ejercicio Integrador del Módulo 1

Objetivo: Aplicar los tres tipos de inclusión de CSS y comprender la Cascada.

1. Crea un archivo HTML (`ejercicio1.html`) y un archivo CSS externo (`ejercicio1.css`).
2. En el HTML, crea un título `<h1>` y un párrafo `<p>` .
3. **En el CSS Externo:** Dale al `<h1>` un color **azul** y al `<p>` un tamaño de fuente de **18px**.
4. **En el CSS Interno** (dentro de `<style>`): Intenta cambiar el color del `<h1>` a **rojo**.
5. **En el CSS Inline** (dentro de la etiqueta `<h1>`): Intenta cambiar el color del `<h1>` a **verde**.

Pregunta: ¿De qué color resulta el título `<h1>` ? ¿Por qué?

Resultado Esperado: El título será **verde** debido a que el estilo **Inline** tiene la mayor **Especificidad** y anula a los estilos Internos y Externos, independientemente del orden en que se carguen.

Módulo 2: Selectores Fundamentales

2.1. Selectores de Tipo, Clase e ID

Teoría: Los selectores son la forma en que “apuntamos” a los elementos HTML para aplicarles estilos.

- **Selector de Tipo:** Selecciona todos los elementos de un tipo específico (ej: `p`, `div`, `a`).
- **Selector de Clase:** Selecciona todos los elementos que tienen un atributo `class` con un nombre específico. Se indica con un punto (`.nombre-clase`). **Es el selector más usado** para aplicar estilos reutilizables.
- **Selector de ID:** Selecciona el único elemento que tiene un atributo `id` con un nombre específico. Se indica con un numeral (`#nombre-id`). **Debe ser único** en toda la página.

Ejemplo Práctico:

Código HTML:

```
<h2 id="titulo-principal">Título Único</h2>
<p class="destacado">Párrafo importante.</p>
<p>Párrafo normal.</p>
<p class="destacado">Otro párrafo importante.</p>
```

Código CSS:

```

/* Selector de Tipo: Aplica a todos los <p> */
p {
    font-style: italic;
}

/* Selector de Clase: Aplica a los elementos con class="destacado" */
.destacado {
    background-color: yellow;
    border: 1px solid orange;
}

/* Selector de ID: Aplica solo al elemento con id="titulo-principal" */
#titulo-principal {
    color: purple;
    text-transform: uppercase;
}

```

Explicación: Los párrafos importantes (`.destacado`) tendrán fondo amarillo y borde naranja, además de ser cursiva (por la herencia del selector de tipo `p`). El título principal será morado y en mayúsculas.

2.2. Selectores de Atributo

Teoría: Permiten seleccionar elementos basándose en la presencia o el valor de un atributo HTML, sin necesidad de usar una clase o ID. Son muy útiles para estilizar formularios o enlaces.

Selector	Descripción	Ejemplo
<code>[atributo]</code>	Selecciona elementos que tienen el atributo.	<code>[disabled]</code>
<code>[atributo="valor"]</code>	Selecciona elementos con el valor exacto.	<code>[type="submit"]</code>
<code>[atributo*="valor"]</code>	Selecciona elementos cuyo valor contiene la subcadena.	<code>[href*="google"]</code>
<code>[atributo^="valor"]</code>	Selecciona elementos cuyo valor comienza con la subcadena.	<code>[class^="btn-"]</code>
<code>[atributo\$="valor"]</code>	Selecciona elementos cuyo valor termina con la subcadena.	<code>[src\$=".png"]</code>

Ejemplo Práctico:

Código HTML:

```
<input type="text" placeholder="Nombre">
<input type="email" placeholder="Correo">
<button type="submit">Enviar</button>
<a href="https://google.com">Google</a>
<a href="/contacto">Contacto</a>
```

Código CSS:

```
/* Estiliza todos los inputs con type="text" */
input[type="text"] {
    border: 2px solid blue;
}

/* Estiliza el botón de envío */
button[type="submit"] {
    background-color: green;
    color: white;
    padding: 10px 20px;
}

/* Estiliza enlaces que contienen "google" en su href */
a[href*="google"] {
    font-weight: bold;
}
```

Explicación: El primer `input` tendrá un borde azul. El botón de envío tendrá un estilo de botón verde. El enlace a Google se mostrará en negrita.

2.3. Selectores Combinadores (Descendiente, Hijo, Adyacente, General)

Teoría: Los selectores combinadores nos permiten seleccionar elementos basándonos en su **relación** con otros elementos en la estructura HTML (el árbol DOM).

Combinador	Símbolo	Descripción	Ejemplo
Descendiente	Espacio ()	Selecciona un elemento que está dentro de otro, sin importar el nivel.	<code>div p</code>
Hijo	Mayor que (>)	Selecciona un elemento que es hijo directo de otro.	<code>ul > li</code>
Hermano Adyacente	Más (+)	Selecciona un elemento que está inmediatamente después de otro, al mismo nivel.	<code>h2 + p</code>
Hermano General	Tilde (~)	Selecciona todos los elementos que están después de otro, al mismo nivel.	<code>h2 ~ p</code>

Ejemplo Práctico:

Código HTML:

```
<div class="contenedor">
  <p>Párrafo 1 (Descendiente)</p>
  <span>
    <p>Párrafo 2 (Descendiente)</p>
  </span>
</div>
<p>Párrafo 3 (Fuera del contenedor)</p>
```

Código CSS:

```
/* Descendiente: Aplica a Párrafo 1 y Párrafo 2 */
.contenedor p {
  color: red;
}

/* Hijo: Aplica solo a Párrafo 1 (es hijo directo de .contenedor) */
.contenedor > p {
  font-weight: bold;
}
```

Explicación: El `Párrafo 1` será **rojo y negrita** porque cumple ambas reglas. El `Párrafo 2` será solo **rojo** porque es descendiente, pero no hijo directo (su padre es ``). El `Párrafo 3` no se ve afectado.

Resumen del Módulo 2: Propiedades Aprendidas

Selector	Uso	Ejemplo	Especificidad
Tipo	Selecciona etiquetas HTML.	<code>h1</code> , <code>p</code> , <code>div</code>	Baja
Clase	Selecciona elementos con <code>class="..."</code> .	<code>.btn</code> , <code>.card</code>	Media
ID	Selecciona el elemento con <code>id="..."</code> .	<code>#logo</code> , <code>#main</code>	Muy Alta
Atributo	Selecciona por atributo y/o valor.	<code>[type="email"]</code>	Media
Descendiente	Selecciona un elemento dentro de otro.	<code>nav a</code>	Combinada
Hijo	Selecciona un hijo directo.	<code>ul > li</code>	Combinada

Ejercicio Integrador del Módulo 2

Objetivo: Usar selectores de clase, atributo y combinadores para estilizar una lista de tareas.

Código HTML:

```
<div class="lista-tareas">
  <h2>Lista de Compras</h2>
  <ul id="compras">
    <li class="pendiente" data-categoria="fruta">Manzanas</li>
    <li class="completada" data-categoria="verdura">Lechuga</li>
    <li class="pendiente" data-categoria="lacteo">Leche</li>
    <li class="completada" data-categoria="fruta">Bananas</li>
  </ul>
</div>
```

Tarea CSS:

1. Aplica un color de fondo gris claro a toda la lista (`#compras`).
2. Usa un selector de **clase** para que las tareas `.completada` tengan el texto tachado (`text-decoration: line-through;`).
3. Usa un selector de **atributo** para que los elementos con `data-categoria="fruta"` tengan un color de texto **verde**.
4. Usa un selector **hijo** para que solo los `` que son hijos directos de `ul` tengan un `padding` de `5px`.

Resultado Esperado:

- **Lechuga** y **Bananas** deben estar tachadas.
 - **Manzanas** y **Bananas** deben tener el texto en color verde.
 - Todos los ítems de la lista deben tener un relleno interno de `5px`.
-

Módulo 3: Estilos de Texto y Tipografía

3.1. Propiedades de Color (`color` , `background-color`)

Teoría: Las propiedades de color son esenciales para el diseño visual.

- `color` : Define el color del **texto** (primer plano).
- `background-color` : Define el color de **fondo** del elemento.

Los valores de color se pueden especificar de varias maneras:

1. **Nombres de color:** `red` , `blue` , `white` .
2. **Hexadecimal:** `#RRGGBB` (ej: `#FF0000` para rojo).
3. **RGB:** `rgb(rojo, verde, azul)` (ej: `rgb(255, 0, 0)`).
4. **RGBA:** `rgba(rojo, verde, azul, alfa)` (el `alfa` es la opacidad, de 0 a 1).

Ejemplo Práctico:

Código CSS:

```
.caja-alerta {  
  background-color: rgba(255, 0, 0, 0.2); /* Fondo rojo con 20% de opacidad */  
  color: #8b0000; /* Texto rojo oscuro */  
  border: 1px solid rgb(255, 0, 0);  
  padding: 15px;  
}
```

Explicación: Usar `rgba` para el fondo permite que el color de fondo sea semitransparente, dejando ver lo que haya detrás, mientras que el texto y el borde mantienen su opacidad total.

3.2. Fuentes (`font-family` , `font-size` , `font-weight` , `font-style`)

Teoría: Controlar la tipografía es clave para la legibilidad y la estética.

- `font-family` : Define la fuente. Siempre se debe incluir una **fuentes de reserva** genérica (`serif` , `sans-serif` , `monospace`) por si la fuente principal no está disponible.
- `font-size` : Define el tamaño del texto. Las unidades más comunes son `px` (píxeles) y `em` o `rem` (unidades relativas, recomendadas para accesibilidad).
- `font-weight` : Define el grosor de la fuente (ej: `normal` , `bold` , o valores numéricos como `400` para normal y `700` para negrita).
- `font-style` : Define el estilo (ej: `normal` , `italic`).

Ejemplo Práctico:

Código CSS:

```
body {
  font-family: 'Helvetica Neue', Arial, sans-serif; /* Prioriza Helvetica,
si no, Arial, si no, cualquier sans-serif */
  font-size: 16px;
}

h1 {
  font-size: 2.5rem; /* 2.5 veces el tamaño de la fuente raíz (root) */
  font-weight: 900; /* Extra-negrita */
}

.cursiva {
  font-style: italic;
}
```

Explicación: El uso de `rem` en `h1` es una buena práctica de **Responsive Design** (que veremos más adelante), ya que el tamaño se ajusta si el usuario cambia el tamaño de fuente base en su navegador.

3.3. Alineación y Decoración de Texto (`text-align`, `text-decoration`, `line-height`)

Teoría: Estas propiedades controlan la presentación del texto dentro de su caja.

- `text-align`: Alinea el texto horizontalmente (`left`, `right`, `center`, `justify`).
- `text-decoration`: Añade líneas al texto (`underline`, `overline`, `line-through`, `none`).
- `line-height`: Controla el espacio vertical entre las líneas de texto (interlineado). Un valor de `1.5` es un buen estándar para la legibilidad.
- `text-transform`: Cambia la capitalización del texto (`uppercase`, `lowercase`, `capitalize`).

Ejemplo Práctico:

Código CSS:

```
.parrafo-justificado {
  text-align: justify;
  line-height: 1.6; /* Interlineado generoso para mejor lectura */
}

a {
  text-decoration: none; /* Quita el subrayado predeterminado de los enlaces */
  text-transform: capitalize; /* Convierte la primera letra de cada palabra a mayúscula */
}
```

Explicación: Es común usar `text-decoration: none;` en los enlaces (`<a>`) para quitar el subrayado predeterminado y luego añadir un efecto visual con `:hover` (que veremos en el Nivel 2).

Resumen del Módulo 3: Propiedades Aprendidas

Propiedad	Función	Unidades Comunes
color	Color del texto.	Nombres, Hex, RGB, RGBA
background-color	Color de fondo.	Nombres, Hex, RGB, RGBA
font-family	Tipo de fuente.	Nombres de fuentes, genéricas
font-size	Tamaño del texto.	px , em , rem
font-weight	Grosor del texto.	normal , bold , 100 a 900
text-align	Alineación horizontal.	left , center , justify
line-height	Espacio entre líneas.	Número (ej: 1.5), px , %
text-decoration	Decoración del texto.	none , underline , line-through

Ejercicio Integrador del Módulo 3

Objetivo: Diseñar un bloque de cita usando todas las propiedades de tipografía aprendidas.

Código HTML:

```
<blockquote class="cita-destacada">
  <p>La simplicidad es la máxima sofisticación.</p>
  <footer>– Leonardo da Vinci</footer>
</blockquote>
```

Tarea CSS:

1. Aplica un `background-color` semitransparente (`rgba`) al `blockquote` .
2. Establece la `font-family` a una fuente `serif` (ej: `Georgia`, `serif`).
3. El texto de la cita (`p`) debe tener un `font-size` de `1.4em` y estar en **cursiva**.
4. El pie de página (`footer`) debe estar alineado a la **derecha**, tener un `font-weight` de `bold` y un `font-size` más pequeño (`0.9em`).

Resultado Esperado: Una cita visualmente separada del texto principal, con un estilo clásico y legible.

Módulo 4: El Modelo de Caja (Box Model)

4.1. Concepto de Bloque e Inline

Teoría: En CSS, cada elemento HTML se genera como una caja rectangular. El **Modelo de Caja** define cómo se calculan el ancho y alto de estas cajas, incluyendo el contenido, el relleno, el borde y el margen.

Los elementos se dividen en dos categorías principales de visualización:

- **Elementos de Bloque (`display: block`):** Ocupan todo el ancho disponible, creando un “salto de línea” antes y después. Permiten controlar `width`, `height`, `margin` y `padding`. Ejemplos: `<div>`, `<p>`, `<h1>`, ``, ``.
- **Elementos en Línea (`display: inline`):** Solo ocupan el ancho necesario para su contenido. **No** permiten controlar `width`, `height`, ni `margin-top` / `margin-bottom`. Ejemplos: ``, `<a>`, ``, ``.

Ejemplo Práctico:

Código HTML:

```
<div class="caja-bloque">Soy un Bloque</div>
<div class="caja-bloque">Soy otro Bloque</div>

<span class="caja-inline">Soy Inline 1</span>
<span class="caja-inline">Soy Inline 2</span>
```

Código CSS:

```
.caja-bloque {
  background-color: lightblue;
  margin: 10px; /* Se aplica margen arriba y abajo */
  width: 150px; /* Se aplica el ancho */
}

.caja-inline {
  background-color: lightgreen;
  margin: 10px; /* Solo se aplica margen a los lados */
  width: 150px; /* NO se aplica el ancho */
}
```

Explicación: Los bloques se apilan verticalmente y respetan el ancho de 150px. Los elementos en línea se colocan uno al lado del otro y el `width` es ignorado.

4.2. width y height

Teoría: Definen las dimensiones del área de **contenido** de la caja.

- `width` : Ancho del contenido.
- `height` : Alto del contenido.

Se pueden usar unidades absolutas (`px`) o relativas (`%`, `vw`, `vh`).

- `%` : Relativo al ancho/alto del elemento padre.
- `vw` (viewport width): Porcentaje del ancho de la ventana del navegador.

- `vh` (viewport height): Porcentaje del alto de la ventana del navegador.

Ejemplo Práctico:

```
.caja-principal {  
  width: 80%; /* Ocupa el 80% del ancho del padre */  
  min-height: 100vh; /* Ocupa al menos el 100% del alto de la ventana (útil  
para landing pages) */  
  max-width: 1200px; /* No se estira más allá de 1200px */  
}
```

4.3. padding (Relleno)

Teoría: El `padding` es el **espacio interno** que existe entre el **contenido** del elemento y su **borde**. Aumenta el tamaño visual total de la caja.

Se puede aplicar a los cuatro lados: `padding-top`, `padding-right`, `padding-bottom`, `padding-left`.

Atajo (Shorthand):

- `padding: 10px;` (Los 4 lados)
- `padding: 10px 20px;` (Arriba/Abajo, Izquierda/Derecha)
- `padding: 10px 20px 30px 40px;` (Arriba, Derecha, Abajo, Izquierda - Sentido horario)

Ejemplo Práctico:

```
.boton {  
  background-color: blue;  
  color: white;  
  padding: 10px 20px; /* 10px arriba y abajo, 20px a los lados */  
}
```

4.4. border (Borde)

Teoría: El `border` es la línea que rodea el `padding` y el contenido. Se define con tres propiedades:

1. `border-width`: Grosor del borde (ej: `1px`, `medium`).
2. `border-style`: Estilo del borde (ej: `solid`, `dashed`, `dotted`).
3. `border-color`: Color del borde.

Atajo (Shorthand):

- `border: 1px solid black;`

Ejemplo Práctico:

```
.imagen-perfil {  
  border: 5px solid #ccc;  
  border-radius: 50%; /* Propiedad para hacer bordes redondeados, 50% lo  
hace circular */  
}
```

4.5. margin (Margen)

Teoría: El `margin` es el **espacio externo** que existe entre el **borde** de un elemento y los elementos vecinos. Se utiliza para separar elementos.

Se puede aplicar a los cuatro lados: `margin-top`, `margin-right`, `margin-bottom`, `margin-left`.

Atajo (Shorthand): Es idéntico al de `padding`.

Centrado Horizontal: Para centrar un elemento de **bloque** con un ancho definido, se usa:

```
.centrado {  
  width: 500px;  
  margin: 0 auto; /* 0px arriba y abajo, 'auto' a los lados */  
}
```

Colapso de Márgenes: Una particularidad del `margin` es que los márgenes verticales (arriba y abajo) de dos elementos adyacentes se **colapsan**, y solo se aplica el margen más grande.

4.6. `box-sizing: border-box;` (La solución moderna)

Teoría: Por defecto, el modelo de caja de CSS es `content-box`. Esto significa que si defines `width: 100px;`, y luego añades `padding: 10px;` y `border: 1px;`, el ancho total de la caja será: `100px` (contenido) + `20px` (padding) + `2px` (borde) = `122px`. Esto es confuso y dificulta el diseño.

La solución moderna es usar `box-sizing: border-box;`. Con esta propiedad, el `padding` y el `border` se incluyen **dentro** del `width` y `height` definidos. Si defines `width: 100px;`, el ancho total **será** `100px`.

Práctica Recomendada: Es una práctica estándar aplicarlo a todos los elementos:

```
/* La mejor práctica para empezar cualquier proyecto */
*, *::before, *::after {
  box-sizing: border-box;
}
```

Ejemplo Práctico:

```
.caja-clasica {
  width: 100px;
  padding: 10px; /* Ancho total: 120px */
  border: 1px solid black;
}

.caja-moderna {
  box-sizing: border-box;
  width: 100px;
  padding: 10px; /* Ancho total: 100px */
  border: 1px solid red;
}
```

Resumen del Módulo 4: Propiedades Aprendidas

Propiedad	Función	Relación con el Box Model
<code>display</code>	Define el tipo de caja (bloque o en línea).	Fundamental
<code>width</code> , <code>height</code>	Dimensiones del contenido.	Contenido
<code>padding</code>	Espacio interno (contenido a borde).	Relleno
<code>border</code>	Línea que rodea el relleno.	Borde
<code>margin</code>	Espacio externo (borde a elementos vecinos).	Margen
<code>box-sizing</code>	Define cómo se calcula el ancho/alto.	Modelo de Cálculo

Ejercicio Integrador del Módulo 4

Objetivo: Crear una tarjeta de producto centrada y con el modelo de caja moderno.

1. Aplica `box-sizing: border-box;` globalmente.
2. Crea un `div` con la clase `.tarjeta`.
3. Dale a `.tarjeta` un `width` de `300px`, un `padding` de `20px` y un `border` de `3px solid #ccc`.
4. Centra la tarjeta horizontalmente usando `margin: 20px auto;`.
5. Dentro de la tarjeta, crea un `h3` y un `<p>`. Dale al `h3` un `margin-top: 0;` para eliminar el espacio superior predeterminado.

Pregunta: Si la tarjeta tiene `width: 300px`, `padding: 20px` y `border: 3px`, ¿cuál es el ancho total de la caja en la pantalla?

Resultado Esperado: El ancho total es **300px** gracias a `box-sizing: border-box;`.

Nivel 2: Intermedio (Posicionamiento y Diseño Clásico)

Módulo 5: Posicionamiento de Elementos

5.1. Propiedad `position` (`static`, `relative`, `absolute`, `fixed`, `sticky`)

Teoría: La propiedad `position` es crucial para controlar la ubicación exacta de un elemento en la página.

Valor	Descripción
<code>static</code>	Valor por defecto. El elemento sigue el flujo normal del documento. Las propiedades <code>top</code> , <code>right</code> , <code>bottom</code> , <code>left</code> son ignoradas.
<code>relative</code>	El elemento se mueve relativo a su posición normal en el flujo. Deja un espacio vacío donde estaba originalmente. Las propiedades <code>top</code> , <code>right</code> , <code>bottom</code> , <code>left</code> funcionan.
<code>absolute</code>	El elemento se saca del flujo normal. Se posiciona relativo al ancestro posicionado más cercano (cualquier ancestro con <code>position</code> distinto de <code>static</code>). Si no hay ancestro posicionado, se posiciona respecto al <code><body></code> .
<code>fixed</code>	El elemento se saca del flujo normal. Se posiciona relativo a la ventana del navegador (viewport) y permanece fijo al hacer <i>scroll</i> .
<code>sticky</code>	Un híbrido. Se comporta como <code>relative</code> hasta que el usuario hace <i>scroll</i> y alcanza un punto de desplazamiento definido, momento en el que se comporta como <code>fixed</code> .

Ejemplo Práctico (Relative y Absolute):

Código HTML:

```
<div class="contenedor-relativo">
  <p>Contenedor</p>
  <div class="caja-absoluta">¡Absoluto!</div>
</div>
```

Código CSS:

```
.contenedor-relativo {
  position: relative; /* Clave para que .caja-absoluta se posicione respecto
a este div */
  width: 300px;
  height: 150px;
  border: 2px solid blue;
}

.caja-absoluta {
  position: absolute;
  top: 10px;
  right: 10px;
  background-color: red;
  color: white;
  padding: 5px;
}
```

Explicación: Al darle `position: relative` al padre (`.contenedor-relativo`), la caja hija (`.caja-absoluta`) se posiciona a 10px del borde superior y 10px del borde derecho **de su padre**. Si el padre no tuviera `position: relative` , la caja absoluta se posicionaría respecto al `<body>` .

5.2. Coordenadas (`top` , `right` , `bottom` , `left`)

Teoría: Estas propiedades definen el desplazamiento del elemento **posicionado** (es decir, con `position: relative` , `absolute` , `fixed` o `sticky`) desde el borde de su contenedor de referencia.

- `top` : Distancia desde el borde superior.
- `bottom` : Distancia desde el borde inferior.
- `left` : Distancia desde el borde izquierdo.

- `right` : Distancia desde el borde derecho.

Ejemplo Práctico (Fixed):

```
.barra-navegacion-fija {  
  position: fixed;  
  top: 0; /* Pegado al borde superior de la ventana */  
  left: 0; /* Pegado al borde izquierdo de la ventana */  
  width: 100%; /* Ocupa todo el ancho de la ventana */  
  background-color: #333;  
  color: white;  
  padding: 10px 0;  
}
```

Explicación: Esta barra de navegación permanecerá visible en la parte superior de la pantalla, sin importar cuánto *scroll* haga el usuario.

5.3. Control de Capas (`z-index`)

Teoría: La propiedad `z-index` controla el orden de apilamiento de los elementos **posicionados** en el eje Z (profundidad). Un valor más alto significa que el elemento estará más cerca del usuario (encima de otros).

- Solo funciona en elementos con `position` distinto de `static`.
- Los valores pueden ser positivos o negativos.

Ejemplo Práctico:

```
.modal-fondo {  
  position: fixed;  
  top: 0;  
  left: 0;  
  width: 100%;  
  height: 100%;  
  background-color: rgba(0, 0, 0, 0.5);  
  z-index: 100; /* Fondo de la ventana modal */  
}  
  
.modal-contenido {  
  position: fixed;  
  top: 50%;  
  left: 50%;  
  transform: translate(-50%, -50%); /* Centrado perfecto (veremos transform  
en Nivel 4) */  
  background-color: white;  
  padding: 30px;  
  z-index: 101; /* Debe estar por encima del fondo */  
}
```

Explicación: El `modal-contenido` tiene un `z-index` más alto (101) que el `modal-fondo` (100), asegurando que el contenido de la ventana modal siempre se vea por encima del fondo semitransparente.

Resumen del Módulo 5: Propiedades Aprendidas

Propiedad	Valor	Descripción
<code>position</code>	<code>static</code>	Flujo normal (por defecto).
<code>position</code>	<code>relative</code>	Relativo a su posición normal.
<code>position</code>	<code>absolute</code>	Relativo al ancestro posicionado más cercano.
<code>position</code>	<code>fixed</code>	Relativo a la ventana del navegador.
<code>position</code>	<code>sticky</code>	Híbrido entre <code>relative</code> y <code>fixed</code> .
<code>top</code> , <code>right</code> , <code>bottom</code> , <code>left</code>	Unidades (<code>px</code> , <code>%</code>)	Coordenadas de desplazamiento.
<code>z-index</code>	Número entero	Orden de apilamiento en el eje Z.

Ejercicio Integrador del Módulo 5

Objetivo: Crear un ícono de “Notificación” en la esquina superior derecha de un elemento.

1. Crea un `div` con la clase `.tarjeta-producto` y dale `position: relative;`.
2. Dentro, crea un `span` con la clase `.notificacion` y dale `position: absolute;`.
3. Usa las coordenadas para posicionar `.notificacion` a `5px` de `top` y `5px` de `right` del `.tarjeta-producto`.
4. Dale un `background-color: red;` y un `z-index: 10;` a la notificación para que se vea por encima de cualquier otro contenido de la tarjeta.

Resultado Esperado: Un pequeño círculo rojo o un badge numérico en la esquina superior derecha de la tarjeta, que se mueve con la tarjeta.

Módulo 6: Fondos y Visuales

6.1. Imágenes de Fondo (`background-image` , `background-repeat` , `background-position`)

Teoría: Podemos usar imágenes en lugar de colores como fondo de un elemento.

- `background-image` : Define la URL de la imagen (ej: `url('ruta/imagen.jpg')`).
- `background-repeat` : Controla si la imagen se repite. Valores comunes: `repeat` (por defecto), `no-repeat` , `repeat-x` , `repeat-y` .
- `background-position` : Define la posición inicial de la imagen. Valores comunes: `center` , `top` , `right` , o coordenadas (`50% 50%`).
- `background-size` : Controla el tamaño de la imagen de fondo. Valores comunes:
 - `cover` : Escala la imagen para cubrir todo el contenedor (puede recortar la imagen).
 - `contain` : Escala la imagen para que sea visible por completo (puede dejar espacio vacío).

Ejemplo Práctico:

Código CSS:

```
.hero-section {  
  background-image: url('img/fondo-principal.jpg');  
  background-repeat: no-repeat;  
  background-position: center center; /* Centra la imagen horizontal y  
verticalmente */  
  background-size: cover; /* Asegura que la imagen cubra toda la sección */  
  height: 400px;  
}
```

Explicación: Esta es la configuración estándar para crear una sección de “héroe” con una imagen de fondo que se adapta al tamaño del contenedor sin repetirse.

6.2. Fondos Múltiples y Atajo (background)

Teoría: CSS permite aplicar **múltiples imágenes de fondo** a un solo elemento, separándolas por comas. Esto es útil para aplicar patrones o efectos de superposición.

El atajo `background` permite definir todas las propiedades de fondo en una sola línea, lo que es más limpio y rápido.

Ejemplo Práctico (Fondo Múltiple):

```
.superposicion {  
    /* Se listan de adelante hacia atrás. El degradado va primero (más cerca  
    del usuario) */  
    background-image:  
        linear-gradient(rgba(0, 0, 0, 0.5), rgba(0, 0, 0, 0.5)), /* Degradado  
        semitransparente */  
        url('img/patron.png'),  
        url('img/fondo-principal.jpg');  
  
    background-repeat: no-repeat, repeat, no-repeat;  
    background-size: cover, auto, cover;  
}
```

Explicación: El elemento tendrá un fondo principal (`fondo-principal.jpg`), sobre el cual se repetirá un patrón pequeño (`patron.png`), y encima de todo, una capa semitransparente oscura (el `linear-gradient`) para mejorar la legibilidad del texto.

6.3. Opacidad y Transparencia (opacity , colores RGBA)

Teoría: Hay dos formas principales de manejar la transparencia:

1. `opacity` : Aplica transparencia a **todo el elemento**, incluyendo su contenido (texto, imágenes, etc.). El valor va de `0` (totalmente transparente) a `1` (totalmente opaco).
2. `rgba()` : Aplica transparencia **solo al color** (fondo o texto), dejando el contenido del elemento con opacidad total.

Ejemplo Práctico:

```

.caja-opaca {
  background-color: black;
  opacity: 0.5; /* La caja y su texto serán 50% transparentes */
  color: white;
}

.caja-transparente-fondo {
  background-color: rgba(0, 0, 0, 0.5); /* Solo el fondo es 50% transparente */
  color: white; /* El texto es 100% opaco */
}

```

Explicación: Si el objetivo es tener un fondo semitransparente pero un texto perfectamente legible, **siempre** se debe usar `rgba()` o `hsla()` en la propiedad de color, y **evitar** usar `opacity` en el contenedor.

6.4. Sombras (`box-shadow` , `text-shadow`)

Teoría: Las sombras añaden profundidad y realismo a los elementos.

- `box-shadow` : Aplica una o más sombras a la caja del elemento.
- `text-shadow` : Aplica una o más sombras al texto.

Sintaxis de `box-shadow` : `box-shadow: [offset-x] [offset-y] [blur-radius] [spread-radius] [color] [inset];`

Ejemplo Práctico:

```

.tarjeta-elevada {
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); /* Sombra suave hacia abajo */
  padding: 20px;
  background-color: white;
}

.titulo-sombra {
  text-shadow: 2px 2px 4px #000000; /* Sombra negra de 2px de desplazamiento
y 4px de desenfoque */
  color: white;
}

```

Explicación: La sombra de la tarjeta (`box-shadow`) se desplaza 4px hacia abajo, tiene un desenfoque de 8px y es de color negro semitransparente, creando un efecto de “elevación” muy usado en el diseño moderno.

Resumen del Módulo 6: Propiedades Aprendidas

Propiedad	Función	Valores Clave
<code>background-image</code>	Imagen de fondo.	<code>url(...)</code> , <code>linear-gradient(...)</code>
<code>background-repeat</code>	Repetición de la imagen.	<code>no-repeat</code> , <code>repeat</code> , <code>cover</code>
<code>background-position</code>	Posición inicial de la imagen.	<code>center</code> , <code>top right</code> , <code>50% 50%</code>
<code>background-size</code>	Tamaño de la imagen.	<code>cover</code> , <code>contain</code> , <code>100%</code>
<code>opacity</code>	Transparencia de todo el elemento.	0 a 1
<code>box-shadow</code>	Sombra de la caja.	Desplazamiento, desenfoque, color
<code>text-shadow</code>	Sombra del texto.	Desplazamiento, desenfoque, color

Ejercicio Integrador del Módulo 6

Objetivo: Crear un botón con un efecto visual de sombra y fondo degradado.

1. Crea un botón con la clase `.btn-degradado`.
2. Dale un `background-image` que sea un degradado lineal (`linear-gradient`) de azul claro a azul oscuro.
3. Aplica un `box-shadow` suave que simule que el botón está ligeramente presionado (sombra pequeña y oscura).
4. Aplica un `text-shadow` muy sutil al texto del botón para que resalte.
5. Asegúrate de que el texto sea blanco (`color: white;`).

Resultado Esperado: Un botón moderno con profundidad y un fondo de color dinámico.

Módulo 7: Pseudoclases y Pseudoelementos

7.1. Pseudoclases de Interacción (`:hover` , `:active` , `:focus`)

Teoría: Las **Pseudoclases** nos permiten aplicar estilos a un elemento cuando está en un estado específico, sin cambiar el HTML. Se escriben con un solo dos puntos (`:`).

- `:hover` : El estilo se aplica cuando el cursor del mouse está sobre el elemento.
- `:active` : El estilo se aplica cuando el elemento está siendo activado (ej: al hacer clic en un botón).
- `:focus` : El estilo se aplica cuando un elemento de formulario (input, textarea, button) está seleccionado o listo para recibir entrada.

Ejemplo Práctico:

```
.btn-principal {
  background-color: #007bff;
  color: white;
  transition: background-color 0.3s ease; /* Veremos transiciones en Nivel 4 */
}

.btn-principal:hover {
  background-color: #0056b3; /* Se oscurece al pasar el mouse */
  cursor: pointer;
}

input:focus {
  border-color: #007bff;
  box-shadow: 0 0 5px rgba(0, 123, 255, 0.5); /* Resalta el input al seleccionarlo */
  outline: none; /* Quita el borde de enfoque predeterminado del navegador */
}
```

Explicación: El efecto `:hover` es fundamental para la usabilidad, ya que proporciona *feedback* visual al usuario. El `:focus` es esencial para la accesibilidad, indicando claramente qué campo de formulario está activo.

7.2. Pseudoclases Estructurales (`:nth-child` , `:first-child` , `:last-child`)

Teoría: Estas pseudoclases seleccionan elementos basándose en su posición dentro de un grupo de hermanos.

- `:first-child` : Selecciona el primer elemento hijo de su padre.
- `:last-child` : Selecciona el último elemento hijo de su padre.
- `:nth-child(n)` : Selecciona el hijo en la posición `n` . También acepta fórmulas:
 - `nth-child(odd)` : Selecciona los impares.
 - `nth-child(even)` : Selecciona los pares.
 - `nth-child(3n)` : Selecciona cada tercer elemento.

Ejemplo Práctico:

```
/* Estiliza la primera fila de una tabla */
tr:first-child {
  font-weight: bold;
  background-color: #f0f0f0;
}

/* Estiliza las filas pares de una lista (efecto cebra) */
li:nth-child(even) {
  background-color: #f9f9f9;
}

/* Estiliza el último elemento de un menú para quitarle un separador */
.menu li:last-child {
  border-right: none;
}
```

Explicación: El efecto “cebra” (`:nth-child(even)`) mejora la legibilidad de listas y tablas largas.

7.3. Pseudoelementos (`::before` , `::after`) para contenido generado

Teoría: Los **Pseudoelementos** nos permiten estilizar partes específicas de un elemento o **insertar contenido decorativo** sin modificar el HTML. Se escriben con doble dos puntos (`::`).

- `::before` : Inserta contenido **antes** del contenido real del elemento.
- `::after` : Inserta contenido **después** del contenido real del elemento.

Requisito: Para que funcionen, deben tener la propiedad `content` definida, incluso si está vacía (`content: ''`).

Ejemplo Práctico:

```
.enlace-externo::after {
  content: ' '; /* Inserta un ícono después del enlace */
  font-size: 0.8em;
}

.titulo-decorado::before {
  content: '';
  display: block;
  width: 50px;
  height: 5px;
  background-color: orange;
  margin-bottom: 5px;
}
```

Explicación: El pseudoelemento `::after` se usa para añadir un ícono de enlace externo. El `::before` se usa para crear una línea decorativa debajo del título. Es importante notar que, por defecto, los pseudoelementos son `inline`, por lo que a menudo se les da `display: block;` para controlar su tamaño.

Resumen del Módulo 7: Pseudoclases y Pseudoelementos

Tipo	Sintaxis	Función	Ejemplo de Uso
Pseudoclase	<code>:estado</code>	Estiliza un elemento en un estado.	<code>:hover</code> , <code>:focus</code>
Pseudoclase	<code>:posición</code>	Estiliza un elemento por su posición.	<code>:first-child</code> , <code>:nth-child(odd)</code>
Pseudoelemento	<code>::parte</code>	Inserta o estiliza una parte del elemento.	<code>::before</code> , <code>::after</code>

Ejercicio Integrador del Módulo 7

Objetivo: Crear una lista de ítems con un separador y un efecto de *hover*.

Código HTML:

```
<ul class="lista-items">
  <li>Item Uno</li>
  <li>Item Dos</li>
  <li>Item Tres</li>
</ul>
```

Tarea CSS:

1. Usa `::after` en cada `li` para insertar un separador vertical (`content: ' | '` ;).
2. Usa `:last-child` para eliminar el separador del último ítem.
3. Usa `:hover` para cambiar el color de fondo de cada `li` a un gris muy claro (`#eee`) al pasar el mouse.

Resultado Esperado: Una lista horizontal con separadores entre los ítems, y un *feedback* visual al interactuar con ellos.

Módulo 8: Diseño de Formularios y Tablas

8.1. Estilizando Inputs y Botones

Teoría: Los elementos de formulario suelen tener estilos predeterminados poco atractivos. CSS nos permite darles un diseño moderno y consistente.

Inputs: Se estilizan con propiedades de `border`, `padding`, `background-color` y `font-size`. Es crucial usar `:focus` para mejorar la usabilidad.

Botones: Se estilizan con `background-color`, `color`, `padding`, `border-radius` y `cursor: pointer`. El uso de `:hover` y `:active` es obligatorio.

Ejemplo Práctico:

```

/* Estilo base para todos los inputs de texto y áreas de texto */
input[type="text"], textarea {
  width: 100%;
  padding: 10px;
  margin-bottom: 15px;
  border: 1px solid #ccc;
  border-radius: 4px;
  box-sizing: border-box; /* ¡Siempre! */
}

/* Estilo al enfocar */
input[type="text"]:focus, textarea:focus {
  border-color: #007bff;
  box-shadow: 0 0 5px rgba(0, 123, 255, 0.5);
}

/* Estilo del botón */
.btn-enviar {
  background-color: #28a745;
  color: white;
  padding: 12px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  font-size: 16px;
}

.btn-enviar:hover {
  background-color: #1e7e34;
}

```

8.2. Estilos para Tablas

Teoría: Las tablas son elementos estructurales para datos tabulares. CSS se usa para mejorar su legibilidad.

- `border-collapse: collapse;` : Elimina el espacio entre celdas, haciendo que los bordes se fusionen.
- `padding` : Se aplica a las celdas (`<td>` y `<th>`) para dar espacio interno.
- `tr:nth-child(even)` : Se usa para el efecto “cebra” en las filas.

Ejemplo Práctico:

```
table {
  width: 100%;
  border-collapse: collapse; /* Bordes unidos */
}

th, td {
  border: 1px solid #ddd;
  padding: 8px;
  text-align: left;
}

th {
  background-color: #f2f2f2;
  color: #333;
}

/* Efecto cebra para mejorar la lectura */
tr:nth-child(even) {
  background-color: #f9f9f9;
}

/* Resaltar la fila al pasar el mouse */
tr:hover {
  background-color: #f1f1f1;
}
```

8.3. Ocultar elementos (`display: none;` vs `visibility: hidden;`)

Teoría: Existen varias formas de ocultar un elemento, pero las dos más comunes tienen efectos muy diferentes en el diseño:

1. `display: none;` : El elemento es **eliminado completamente** del flujo del documento. No ocupa espacio, no es interactivo y no es leído por lectores de pantalla. Es la forma más común de ocultar elementos.
2. `visibility: hidden;` : El elemento se vuelve **invisible**, pero **sigue ocupando su espacio** en el diseño. Es útil si se quiere mantener el diseño intacto pero ocultar el contenido.

Ejemplo Práctico:

```
.oculto-totalmente {
  display: none; /* Desaparece y el espacio es ocupado por otros elementos
*/
}

.oculto-pero-ocupa-espacio {
  visibility: hidden; /* El espacio sigue ahí, pero el elemento no se ve */
}
```

Resumen del Módulo 8: Propiedades Aprendidas

Elemento	Propiedades Clave	Pseudoclases/Elementos
Inputs	<code>border</code> , <code>padding</code> , <code>width</code>	<code>:focus</code>
Botones	<code>background-color</code> , <code>padding</code> , <code>cursor</code>	<code>:hover</code> , <code>:active</code>
Tablas	<code>border-collapse</code> , <code>padding</code>	<code>tr:nth-child(even)</code>
Ocultar	<code>display</code> , <code>visibility</code>	N/A

Ejercicio Integrador del Módulo 8

Objetivo: Crear un formulario de contacto simple con un botón de envío que cambia de color al pasar el mouse.

1. Crea un formulario con un `input[type="text"]` y un `button` con la clase `.btn-submit`.
2. Estiliza el `input` para que tenga un borde redondeado y un `padding` interno de `12px`.
3. Estiliza el botón con un fondo azul y texto blanco.
4. Aplica un efecto `:hover` al botón para que su fondo se vuelva un azul más oscuro.
5. Asegúrate de que el `input` y el `button` usen `box-sizing: border-box;` para que el ancho sea predecible.

Resultado Esperado: Un formulario limpio, moderno y con *feedback* visual en el botón.

Nivel 3: Avanzado (Layouts Modernos y Adaptativos)

Módulo 9: Diseño Flexible con Flexbox

9.1. Conceptos de Contenedor y Elementos Flex

Teoría: Flexbox (Módulo de Caja Flexible) es un modelo de diseño unidimensional (trabaja en una fila o en una columna) diseñado para distribuir el espacio entre los elementos de una interfaz y alinearlos de manera eficiente.

- **Contenedor Flex (Flex Container):** El elemento padre al que se le aplica `display: flex;`.
- **Elementos Flex (Flex Items):** Los hijos directos del contenedor.

Ejes:

- **Eje Principal (Main Axis):** La dirección en la que se colocan los elementos (por defecto, horizontal).
- **Eje Cruzado (Cross Axis):** El eje perpendicular al principal (por defecto, vertical).

9.2. Propiedades del Contenedor (`display: flex`, `flex-direction`, `flex-wrap`)

Teoría: Estas propiedades se aplican al **Contenedor Flex** para definir el flujo general de los elementos.

- `display: flex;`: Convierte el elemento en un contenedor flex.

- `flex-direction` : Define el Eje Principal. Valores: `row` (por defecto, horizontal), `row-reverse` , `column` (vertical), `column-reverse` .
- `flex-wrap` : Define si los elementos deben ajustarse a múltiples líneas. Valores: `nowrap` (por defecto), `wrap` , `wrap-reverse` .

Ejemplo Práctico:

```
.galeria {
  display: flex;
  flex-direction: row; /* Los elementos se alinean horizontalmente */
  flex-wrap: wrap; /* Si no caben, saltan a la siguiente línea */
  gap: 10px; /* Espacio entre los elementos (propiedad moderna) */
}
```

9.3. Alineación en el Eje Principal (`justify-content`)

Teoría: Controla cómo se distribuye el espacio **entre** los elementos a lo largo del **Eje Principal**.

Valor	Descripción
<code>flex-start</code>	Alinea los elementos al inicio del eje.
<code>flex-end</code>	Alinea los elementos al final del eje.
<code>center</code>	Centra los elementos en el eje.
<code>space-between</code>	Distribuye el espacio uniformemente entre los elementos (los bordes quedan pegados).
<code>space-around</code>	Distribuye el espacio uniformemente alrededor de los elementos.
<code>space-evenly</code>	Distribuye el espacio de manera que el espacio entre elementos y los bordes sea igual.

Ejemplo Práctico:

```
.menu-navegacion {  
  display: flex;  
  justify-content: space-between; /* El primer elemento a la izquierda, el  
  último a la derecha, y el espacio restante se reparte entre ellos */  
}
```

9.4. Alineación en el Eje Cruzado (`align-items` , `align-content`)

Teoría: Controlan la alineación de los elementos a lo largo del **Eje Cruzado**.

- `align-items` : Alinea los elementos **dentro de una sola línea**. Valores: `flex-start` , `flex-end` , `center` , `stretch` (por defecto).
- `align-content` : Controla la distribución del espacio **entre las líneas** cuando `flex-wrap: wrap`; está activo. Valores: `flex-start` , `flex-end` , `center` , `space-between` , `space-around` .

Ejemplo Práctico:

```
.contenedor-centrado {  
  display: flex;  
  height: 200px;  
  justify-content: center; /* Centrado horizontal */  
  align-items: center; /* Centrado vertical (en una sola línea) */  
}
```

9.5. Propiedades de los Elementos (`flex-grow` , `flex-shrink` , `flex-basis` , `order`)

Teoría: Estas propiedades se aplican a los **Elementos Flex** para controlar su tamaño y orden.

- `flex-basis` : Define el tamaño inicial del elemento antes de que se distribuya el espacio restante.
- `flex-grow` : Factor de crecimiento. Si hay espacio extra, cuánto debe crecer el elemento. `flex-grow: 1`; hace que el elemento ocupe el espacio restante.

- `flex-shrink` : Factor de encogimiento. Si no hay suficiente espacio, cuánto debe encogerse el elemento.
- `order` : Define el orden visual de los elementos (por defecto es 0).

Atajo (Shorthand): `flex: [grow] [shrink] [basis];`

Ejemplo Práctico:

```
.item-principal {  
  flex: 1 1 200px; /* Crece 1, se encoge 1, tamaño base 200px */  
}  
  
.item-secundario {  
  flex: 0 0 100px; /* No crece, no se encoge, tamaño fijo 100px */  
}  
  
.item-movido {  
  order: 1; /* Aparecerá después de los elementos con order: 0 (por defecto) */  
}
```

Resumen del Módulo 9: Propiedades de Flexbox

Aplicación	Propiedad	Función
Contenedor	<code>display: flex</code>	Activa Flexbox.
Contenedor	<code>flex-direction</code>	Define el eje principal.
Contenedor	<code>justify-content</code>	Alineación en el eje principal.
Contenedor	<code>align-items</code>	Alineación en el eje cruzado (una línea).
Elemento	<code>flex</code>	Atajo para <code>grow</code> , <code>shrink</code> , <code>basis</code> .
Elemento	<code>order</code>	Cambia el orden visual.

Ejercicio Integrador del Módulo 9

Objetivo: Crear un *header* con un logo a la izquierda, un menú centrado y un botón a la derecha.

1. Crea un `header` con `display: flex;` y `justify-content: space-between;`.
2. Crea tres hijos: `.logo`, `.menu` y `.boton`.
3. Aplica `align-items: center;` al `header` para alinear todo verticalmente.
4. Usa `flex-grow: 1;` en el `.menu` y `text-align: center;` para que ocupe el espacio restante y centre su contenido.

Resultado Esperado: Un diseño de barra de navegación moderno y perfectamente alineado.

Módulo 10: Diseño de Cuadrícula con CSS Grid

10.1. Conceptos de Contenedor y Elementos Grid

Teoría: **CSS Grid Layout** es un sistema de diseño bidimensional (trabaja con filas Y columnas simultáneamente) que permite crear diseños complejos de manera más sencilla que Flexbox.

- **Contenedor Grid (Grid Container):** El elemento padre al que se le aplica `display: grid;`.
- **Elementos Grid (Grid Items):** Los hijos directos del contenedor.
- **Líneas, Pistas y Celdas:** El Grid se compone de líneas horizontales y verticales que definen pistas (filas y columnas) y celdas (la intersección de una fila y una columna).

10.2. Definición de la Cuadrícula (`grid-template-columns`, `grid-template-rows`)

Teoría: Estas propiedades se aplican al **Contenedor Grid** para definir la estructura de la cuadrícula.

- `grid-template-columns` : Define el número y ancho de las columnas.
- `grid-template-rows` : Define el número y alto de las filas.

Unidad `fr` (Fracción): La unidad `fr` es exclusiva de Grid y representa una fracción del espacio disponible en el contenedor. Es ideal para diseños fluidos.

Ejemplo Práctico:

```
.layout-principal {  
  display: grid;  
  /* 3 columnas: 100px fija, 2 fracciones (ocupa el doble de espacio que la  
  primera fr), 1 fracción */  
  grid-template-columns: 100px 2fr 1fr;  
  /* 2 filas: 50px fija, el resto del espacio */  
  grid-template-rows: 50px auto;  
  gap: 10px; /* Espacio entre celdas */  
}
```

10.3. Posicionamiento de Elementos (`grid-column` , `grid-row`)

Teoría: Estas propiedades se aplican a los **Elementos Grid** para especificar dónde deben colocarse dentro de la cuadrícula, haciendo referencia a las líneas.

- `grid-column-start` , `grid-column-end` : Define las líneas de columna donde comienza y termina el elemento.
- `grid-row-start` , `grid-row-end` : Define las líneas de fila donde comienza y termina el elemento.

Atajo (Shorthand):

- `grid-column: [start] / [end];`
- `grid-row: [start] / [end];`

Ejemplo Práctico:

```

.header {
  grid-column: 1 / 4; /* Comienza en la línea 1 y termina en la línea 4
(ocupa las 3 columnas) */
  grid-row: 1 / 2; /* Ocupa solo la primera fila */
}

.sidebar {
  grid-column: 1 / 2; /* Ocupa solo la primera columna */
  grid-row: 2 / 3; /* Ocupa la segunda fila */
}

```

10.4. Áreas Nombradas (grid-template-areas)

Teoría: Una forma más legible y visual de diseñar la cuadrícula es nombrar las áreas y luego usarlas para posicionar los elementos.

1. Se nombran las áreas en el **Contenedor Grid** usando `grid-template-areas`.
2. Se asigna el nombre del área a cada **Elemento Grid** usando `grid-area`.

Ejemplo Práctico:

Código CSS (Contenedor):

```

.layout-nombrado {
  display: grid;
  grid-template-columns: 1fr 3fr;
  grid-template-rows: auto 1fr auto;
  grid-template-areas:
    "header header"
    "nav    main"
    "footer footer";
}

```

Código CSS (Elementos):

```

.encabezado {
  grid-area: header;
}

.navegacion {
  grid-area: nav;
}

.contenido-principal {
  grid-area: main;
}

.pie-pagina {
  grid-area: footer;
}

```

Explicación: Esta sintaxis permite ver el diseño completo de la página de un vistazo, facilitando la adaptación a diferentes tamaños de pantalla (Responsive Design).

Resumen del Módulo 10: Propiedades de Grid

Aplicación	Propiedad	Función
Contenedor	<code>display: grid</code>	Activa Grid.
Contenedor	<code>grid-template-columns</code>	Define el ancho de las columnas.
Contenedor	<code>grid-template-rows</code>	Define el alto de las filas.
Contenedor	<code>grid-template-areas</code>	Define el diseño por nombres.
Elemento	<code>grid-column</code>	Posicionamiento en columnas.
Elemento	<code>grid-area</code>	Asigna el elemento a un área nombrada.

Ejercicio Integrador del Módulo 10

Objetivo: Crear un diseño de blog de 3 columnas para el contenido principal.

1. Crea un `div` con `display: grid;` y la clase `.blog-layout`.

2. Define 3 columnas de igual ancho usando la unidad `fr`: `grid-template-columns: 1fr 1fr 1fr;`.
3. Añade un `gap` de `20px`.
4. Crea un elemento `.post-destacado` que ocupe las dos primeras columnas. Usa `grid-column: 1 / 3;`.
5. Crea un elemento `.sidebar` que ocupe la tercera columna.

Resultado Esperado: Un diseño de blog donde el post principal es el doble de ancho que la barra lateral.

Módulo 11: Diseño Responsivo (Responsive Design)

11.1. Concepto de Mobile First

Teoría: El **Diseño Responsivo** es la práctica de crear sitios web que se ven bien en todos los dispositivos (móviles, tabletas, escritorios).

La estrategia **Mobile First** (Móvil Primero) es la práctica recomendada:

1. Diseñar y escribir el CSS para el **dispositivo más pequeño** (móvil) primero.
2. Usar **Media Queries** para añadir estilos que *sobrescriban* los estilos móviles solo cuando la pantalla es lo suficientemente grande.

Ventaja: Asegura que el sitio sea rápido y funcional en móviles, que es donde se encuentra la mayor parte del tráfico web.

11.2. Viewport Meta Tag

Teoría: Este *meta tag* es **obligatorio** para que el diseño responsivo funcione correctamente. Le indica al navegador móvil que el ancho de la página debe ser igual al ancho del dispositivo.

Código HTML (Obligatorio en el `<head>`):

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

11.3. Media Queries (Sintaxis y Puntos de Ruptura)

Teoría: Las **Media Queries** son la herramienta clave del diseño responsivo. Permiten aplicar un bloque de estilos CSS solo si se cumplen ciertas condiciones (ej: ancho de pantalla).

Sintaxis:

```
@media (condición) {  
    /* Estilos que solo se aplican si la condición es verdadera */  
}
```

Puntos de Ruptura (Breakpoints): Son los anchos de pantalla donde el diseño debe cambiar. Los más comunes, siguiendo la filosofía *Mobile First* (`min-width`):

Dispositivo	Ancho Mínimo (<code>min-width</code>)	Uso
Móvil	N/A (Estilos base)	Diseño de una sola columna.
Tableta	768px	Introducir un diseño de 2 columnas.
Escritorio	1024px	Introducir un diseño de 3 o más columnas.

Ejemplo Práctico:

```

/* Estilos base (Mobile First) */
.contenedor {
  width: 100%;
  padding: 10px;
  display: block; /* Por defecto, todo es una columna */
}

/* Media Query para Tablet y superiores */
@media (min-width: 768px) {
  .contenedor {
    padding: 20px;
  }
  .menu {
    display: flex; /* Cambia el menú a horizontal */
  }
}

/* Media Query para Escritorio y superiores */
@media (min-width: 1024px) {
  .contenedor {
    max-width: 1200px;
    margin: 0 auto; /* Centra el contenedor en pantallas grandes */
  }
}

```

11.4. Imágenes y Videos Fluidos

Teoría: Para que las imágenes y videos se adapten al tamaño de su contenedor sin desbordarse, se debe limitar su ancho máximo.

Práctica Recomendada:

```

img, video {
  max-width: 100%; /* Nunca será más ancha que su contenedor */
  height: auto; /* Mantiene la proporción original */
  display: block; /* Elimina el espacio extra debajo de las imágenes (son
inline por defecto) */
}

```

Resumen del Módulo 11: Conceptos de Responsive Design

Concepto	Herramienta	Función
Estrategia	Mobile First	Diseñar primero para el móvil.
Configuración	<code><meta name="viewport"></code>	Ajusta el ancho de la página al dispositivo.
Adaptación	<code>@media</code>	Aplica estilos solo si se cumplen condiciones.
Elementos	<code>max-width: 100%</code>	Hace que imágenes y videos sean fluidos.

Ejercicio Integrador del Módulo 11

Objetivo: Adaptar un diseño de tarjeta de producto de una columna a dos columnas en tabletas.

Código HTML:

```
<div class="productos">
  <div class="tarjeta">...</div>
  <div class="tarjeta">...</div>
  <div class="tarjeta">...</div>
</div>
```

Tarea CSS:

- Estilos base (móvil): `.tarjeta` debe tener `width: 100%`; y estar apilada verticalmente.
- Usa una **Media Query** para `min-width: 768px`.
- Dentro de la Media Query, cambia `.productos` a `display: flex;` y `flex-wrap: wrap;`.
- Dentro de la Media Query, haz que `.tarjeta` ocupe aproximadamente la mitad del ancho: `width: calc(50% - 10px);` (usando `calc()` para restar el gap).

Resultado Esperado: Las tarjetas se apilan en móviles y se muestran dos por fila en tabletas.

Nivel 4: Experto (Efectos, Rendimiento y Arquitectura)

Módulo 12: Transiciones y Transformaciones

12.1. Transiciones (`transition-property`, `duration`, `timing-function`, `delay`)

Teoría: Las **Transiciones** permiten que los cambios de estado de CSS (ej: de `:hover` a normal) ocurran de forma suave a lo largo del tiempo, en lugar de instantáneamente.

- `transition-property` : La propiedad CSS a la que se aplica la transición (ej: `background-color`, `all`).
- `transition-duration` : El tiempo que dura la transición (ej: `0.5s`, `500ms`).
- `transition-timing-function` : La curva de velocidad de la transición (ej: `ease`, `linear`, `ease-in-out`).
- `transition-delay` : El tiempo de espera antes de que comience la transición.

Atajo (Shorthand): `transition: [property] [duration] [timing-function] [delay];`

Ejemplo Práctico:

```

.caja-animada {
  background-color: blue;
  width: 100px;
  /* Transición aplicada a todas las propiedades, dura 0.4 segundos, con un
  inicio y fin suaves */
  transition: all 0.4s ease-in-out;
}

.caja-animada:hover {
  background-color: red;
  width: 200px;
}

```

Explicación: Al pasar el mouse, el cambio de color y el aumento de ancho no serán instantáneos, sino que durarán 0.4 segundos, creando un efecto profesional.

12.2. Transformaciones 2D (`translate` , `rotate` , `scale` , `skew`)

Teoría: La propiedad `transform` permite modificar la forma y posición de un elemento en el espacio 2D o 3D.

- `translate(x, y)` : Mueve el elemento horizontalmente (x) y verticalmente (y).
- `rotate(ángulo)` : Rota el elemento (ej: `rotate(45deg)`).
- `scale(x, y)` : Escala el elemento (ej: `scale(1.2)` lo agranda un 20%).
- `skew(x-angle, y-angle)` : Inclina el elemento.

Ejemplo Práctico:

```

.imagen-rotativa {
  transition: transform 0.5s ease;
}

.imagen-rotativa:hover {
  transform: rotate(360deg) scale(1.1); /* Rota 360 grados y agranda un 10%
  */
}

```

12.3. Transformaciones 3D (perspective , rotateX , rotateY)

Teoría: Para crear efectos 3D, se necesita la propiedad `perspective` en el **elemento padre** para definir la profundidad de la escena.

- `rotateX(ángulo)` : Rota el elemento alrededor del eje X (horizontal).
- `rotateY(ángulo)` : Rota el elemento alrededor del eje Y (vertical).

Ejemplo Práctico:

```
.contenedor-3d {  
  perspective: 1000px; /* Define la profundidad de la escena */  
}  
  
.tarjeta-flip {  
  transition: transform 1s;  
  transform-style: preserve-3d; /* Permite que los hijos se posicionen en 3D */  
}  
  
.tarjeta-flip:hover {  
  transform: rotateY(180deg); /* Gira la tarjeta 180 grados en el eje Y */  
}
```

Resumen del Módulo 12: Transiciones y Transformaciones

Propiedad	Función	Valores Clave
<code>transition</code>	Suaviza los cambios de estado.	<code>all 0.3s ease</code>
<code>transform</code>	Modifica la posición, rotación, escala.	<code>translate()</code> , <code>rotate()</code> , <code>scale()</code>
<code>perspective</code>	Define la profundidad para efectos 3D.	<code>1000px</code>

Ejercicio Integrador del Módulo 12

Objetivo: Crear un efecto de “rebote” sutil en un botón al hacer clic.

1. Crea un botón con la clase `.btn-rebote` .

2. Aplica una transición de `transform` de `0.1s`.
3. Usa la pseudoclase `:active` para aplicar un `transform: scale(0.95);` (encogerlo un 5%).

Resultado Esperado: Al hacer clic, el botón se encoge ligeramente, dando una sensación de “presión” inmediata.

Módulo 13: Animaciones con Keyframes

13.1. Sintaxis de `@keyframes`

Teoría: Las **Animaciones** permiten crear secuencias de estilos más complejas que las transiciones. Se definen usando la regla `@keyframes`.

- `@keyframes`: Define la secuencia de la animación, asignándole un nombre.
- Dentro de `@keyframes`, se definen los estilos en diferentes puntos de la animación, usando porcentajes (`0%` a `100%`) o las palabras clave `from` (`0%`) y `to` (`100%`).

Ejemplo Práctico:

```
@keyframes parpadeo {  
  0% { opacity: 1; }  
  50% { opacity: 0.2; }  
  100% { opacity: 1; }  
}
```

13.2. Propiedades de Animación (`animation-name`, `duration`, `iteration-count`, `fill-mode`)

Teoría: Una vez definidos los *keyframes*, se aplican al elemento usando las propiedades `animation-`.

- `animation-name`: El nombre de la animación (ej: `parpadeo`).
- `animation-duration`: El tiempo que dura un ciclo de la animación.

- `animation-iteration-count` : Cuántas veces se repite (ej: `infinite` para repetición infinita).
- `animation-timing-function` : Curva de velocidad.
- `animation-fill-mode` : Define qué estilos se aplican al elemento antes y después de la animación (ej: `forwards` mantiene el estilo del 100%).

Atajo (Shorthand): `animation: [name] [duration] [timing-function] [delay] [iteration-count] [direction] [fill-mode];`

Ejemplo Práctico:

```
.alerta-parpadeante {  
  /* Atajo: nombre, duración, repetición, modo de relleno */  
  animation: parpadeo 1.5s infinite alternate;  
}
```

Explicación: El elemento `.alerta-parpadeante` ejecutará la animación `parpadeo` indefinidamente, durando 1.5 segundos por ciclo, yendo y viniendo (`alternate`).

13.3. Ejemplos de Animaciones Comunes

Ejemplo: Carga (Spinner):

```
@keyframes rotar {  
  to { transform: rotate(360deg); }  
}  
  
.spinner {  
  border: 4px solid rgba(0, 0, 0, 0.1);  
  border-top-color: #333;  
  border-radius: 50%;  
  width: 30px;  
  height: 30px;  
  animation: rotar 1s linear infinite;  
}
```

Resumen del Módulo 13: Animaciones

Propiedad	Función	Aplicación
@keyframes	Define la secuencia de la animación.	Global
animation-name	Asigna la animación al elemento.	Elemento
animation-duration	Tiempo de un ciclo.	Elemento
animation-iteration-count	Número de repeticiones.	Elemento

Ejercicio Integrador del Módulo 13

Objetivo: Crear un efecto de “fade-in” (aparecer suavemente) al cargar un elemento.

1. Define un `@keyframes` llamado `aparecer`.
2. En `0%`, establece `opacity: 0;` y `transform: translateY(20px);`.
3. En `100%`, establece `opacity: 1;` y `transform: translateY(0);`.
4. Aplica la animación a un elemento `.contenido-nuevo` con una duración de `1s` y `animation-fill-mode: forwards;` (para que se quede en el estado final).

Resultado Esperado: El contenido aparece desde abajo y se desvanece suavemente.

Módulo 14: Selectores Avanzados y Funciones

14.1. Selectores de Negación (`:not()`)

Teoría: El selector `:not()` es una pseudoclase funcional que selecciona elementos que **no** coinciden con el selector pasado como argumento.

Ejemplo Práctico:

```

/* Aplica un borde a todos los elementos <p> excepto al que tiene la clase
.sin-borde */
p:not(.sin-borde) {
    border: 1px solid gray;
}

/* Aplica un estilo a todos los inputs que NO están deshabilitados */
input:not([disabled]) {
    opacity: 1;
}

```

14.2. Selectores de Relación (:has())

Teoría: El selector `:has()` es un selector muy potente que selecciona un elemento si **contiene** o está **relacionado** con otro elemento.

Ejemplo Práctico:

```

/* Estiliza un <div> si contiene un <img> */
div:has(img) {
    border: 5px solid gold;
}

/* Estiliza un <a> si contiene un <span> con la clase .icono */
a:has(span.icono) {
    padding-right: 20px;
}

```

14.3. Variables CSS (Propiedades Personalizadas)

Teoría: Las **Variables CSS** (o Propiedades Personalizadas) permiten definir valores que se pueden reutilizar en todo el documento. Esto facilita la gestión de temas de color, tipografía y espaciado.

- Se definen con doble guion (`--nombre-variable`).
- Se suelen definir en el selector `:root` (el elemento `<html>`).
- Se usan con la función `var(--nombre-variable)` .

Ejemplo Práctico:

```
:root {
  --color-principal: #007bff;
  --espaciado-base: 16px;
}

.btn-principal {
  background-color: var(--color-principal);
  padding: calc(var(--espaciado-base) / 2) var(--espaciado-base);
}

.titulo {
  color: var(--color-principal);
}
```

Ventaja: Para cambiar el color principal de todo el sitio, solo se necesita modificar el valor en `:root`.

14.4. Funciones CSS (`calc()`, `min()`, `max()`)

Teoría: CSS incluye funciones matemáticas para realizar cálculos dinámicos.

- `calc()` : Permite realizar operaciones matemáticas básicas (+, -, *, /) con diferentes unidades.
- `min()` : Selecciona el valor más pequeño de una lista de valores.
- `max()` : Selecciona el valor más grande de una lista de valores.

Ejemplo Práctico:

```
.caja-ancho-dinamico {
  /* Ocupa el 90% del ancho, pero nunca más de 1000px */
  width: min(90%, 1000px);
}

.columna-flexible {
  /* El ancho será el 30% del padre, menos 20px de margen */
  width: calc(30% - 20px);
}
```

Módulo 15: Arquitectura y Buenas Prácticas

15.1. Metodologías CSS (BEM, OOCSS, SMACSS - Breve Introducción)

Teoría: A medida que los proyectos crecen, el CSS puede volverse difícil de mantener. Las metodologías de arquitectura ayudan a organizar las clases y los archivos.

- **BEM (Block, Element, Modifier):** La más popular. Nombra las clases de forma estricta para indicar la relación entre ellas.
 - **Bloque:** Entidad independiente (ej: `.card`).
 - **Elemento:** Parte del bloque (ej: `.card__title`).
 - **Modificador:** Variación del bloque o elemento (ej: `.card--dark`, `.card__title--large`).
- **OOCSS (Object-Oriented CSS):** Separa la estructura del *skin* (apariencia) y el contenedor del contenido. Fomenta la reutilización.
- **SMACSS (Scalable and Modular Architecture for CSS):** Organiza el CSS en categorías: Base, Layout, Módulos, Estado y Tema.

Ejemplo BEM:

```
.menu {}  
.menu__item {}  
.menu__item--activo {}
```

15.2. Preprocesadores (Sass/Less - Breve Mención)

Teoría: Los preprocesadores son lenguajes que extienden las capacidades de CSS (variables, anidamiento, funciones, mixins) y luego se compilan a CSS estándar.

- **Sass/SCSS:** El más popular. Permite anidar selectores, usar variables y crear código más DRY (*Don't Repeat Yourself*).

Ejemplo (Sass):

```
$color-primario: #007bff;

.btn {
  background-color: $color-primario;
  &:hover { /* Anidamiento */
    background-color: darken($color-primario, 10%);
  }
}
```

15.3. Optimización y Rendimiento (Reducción de Repintado y Reflujo)

Teoría: Un CSS eficiente es clave para la velocidad de carga.

- **Reflujo (Reflow/Layout):** El navegador debe recalcular la posición y geometría de los elementos. Es la operación más costosa.
- **Repintado (Repaint):** El navegador debe redibujar los píxeles. Menos costoso que el reflujo.

Buenas Prácticas de Rendimiento:

1. **Evitar propiedades que causan Reflujo:** Cambios en `width`, `height`, `margin`, `padding`, `top` / `left` (con `position: absolute/relative`).
2. **Usar `transform` y `opacity`:** Estas propiedades se manejan directamente por la GPU y solo causan Repintado o Compositing, siendo mucho más rápidas.
3. **Minificar el CSS:** Eliminar espacios y comentarios para reducir el tamaño del archivo.

Resumen del Módulo 15: Arquitectura y Rendimiento

Concepto	Descripción	Beneficio
BEM	Metodología de nombrado de clases.	Mantenibilidad y escalabilidad.
Preprocesadores	Lenguajes que extienden CSS (Sass).	Variables, anidamiento, código DRY.
Rendimiento	Optimizar para evitar Reflujo.	Mayor velocidad de carga y fluidez.

Ejercicio Integrador del Módulo 15

Objetivo: Refactorizar un CSS simple usando Variables y BEM.

CSS Original (Malo):

```
.header-nav {  
  background-color: #333;  
}  
.header-nav a {  
  color: white;  
}  
.header-nav a:hover {  
  color: #ffcc00;  
}  
.footer-nav {  
  background-color: #333;  
}  
.footer-nav a {  
  color: white;  
}  
.footer-nav a:hover {  
  color: #ffcc00;  
}
```

Tarea CSS (Refactorizado):

1. Define variables `--color-oscuro` y `--color-acento` en `:root`.
2. Aplica la metodología BEM al menú (ej: `.nav`, `.nav__link`, `.nav__link--hover`).
3. Usa las variables en el CSS.

Resultado Esperado: Un código más limpio, reutilizable y fácil de mantener.

Proyecto Integrador: Landing Page Moderna y Responsiva

Objetivo: Aplicar todos los conocimientos de CSS para construir una *landing page* moderna de una sola página.

Estructura HTML (Sugerida):

```
<header class="main-header">
  <div class="logo">Logo</div>
  <nav class="main-nav">
    <a href="#inicio">Inicio</a>
    <a href="#servicios">Servicios</a>
    <a href="#contacto">Contacto</a>
  </nav>
</header>

<section id="inicio" class="hero-section">
  <h1>Diseño Web con CSS Experto</h1>
  <p>Aprende a crear interfaces fluidas y atractivas.</p>
  <a href="#contacto" class="btn-principal">¡Comienza Ahora!</a>
</section>

<section id="servicios" class="grid-servicios">
  <div class="tarjeta-servicio">...</div>
  <div class="tarjeta-servicio">...</div>
  <div class="tarjeta-servicio">...</div>
</section>

<footer class="main-footer">
  <p>&copy; 2025 Basdonax AI</p>
</footer>
```

Requisitos CSS a Aplicar:

1. **Mobile First:** Estilos base para móviles, luego Media Queries para tabletas (768px) y escritorio (1024px).
2. **Header (Flexbox):** Usar Flexbox para alinear el logo y el menú horizontalmente.

3. **Hero Section (Box Model/Tipografía):** Centrar el contenido vertical y horizontalmente (usando Flexbox o posicionamiento). Usar `background-image: cover;` y un degradado semitransparente.
4. **Grid Servicios (CSS Grid):** En móvil, las tarjetas se apilan (1 columna). En tableta, 2 columnas. En escritorio, 3 columnas.
5. **Efectos:** Aplicar `transition` y `transform: scale()` a las tarjetas de servicio en `:hover`.
6. **Variables:** Usar variables CSS para el color principal y el color de fondo.

Resultado Esperado: Una página completa, funcional y visualmente profesional, que demuestra el dominio de todos los niveles de CSS.

Apéndice

Espacios para Anotaciones y Pruebas de Código

(Espacio reservado para que el lector tome notas y realice sus propias pruebas de código)

Glosario de Propiedades CSS

Propiedad	Nivel	Descripción Breve
color	Básico	Color del texto.
background-color	Básico	Color de fondo.
margin	Básico	Espacio externo de la caja.
padding	Básico	Espacio interno de la caja.
box-sizing	Básico	Define el modelo de cálculo de la caja.
position	Intermedio	Define el método de posicionamiento.
z-index	Intermedio	Orden de apilamiento en el eje Z.
:hover	Intermedio	Pseudoclase para el estado del mouse.
::before	Intermedio	Pseudoelemento para contenido antes del elemento.
display: flex	Avanzado	Activa el modelo de caja flexible.
justify-content	Avanzado	Alineación en el eje principal de Flexbox.
display: grid	Avanzado	Activa el modelo de cuadrícula.
grid-template-columns	Avanzado	Define las columnas de la cuadrícula.
@media	Avanzado	Define un bloque de estilos condicionales (Responsive).
transition	Experto	Suaviza los cambios de estilo.
transform	Experto	Modifica la posición, rotación o escala.
@keyframes	Experto	Define una secuencia de animación.
var()	Experto	Utiliza una variable CSS.

Recursos Adicionales

- **MDN Web Docs (Mozilla Developer Network):** La referencia más completa y autorizada para CSS.
- **Flexbox Froggy:** Juego interactivo para aprender Flexbox.
- **Grid Garden:** Juego interactivo para aprender CSS Grid.
- **Can I Use:** Herramienta para verificar la compatibilidad de propiedades CSS en diferentes navegadores.