

CLASE 6

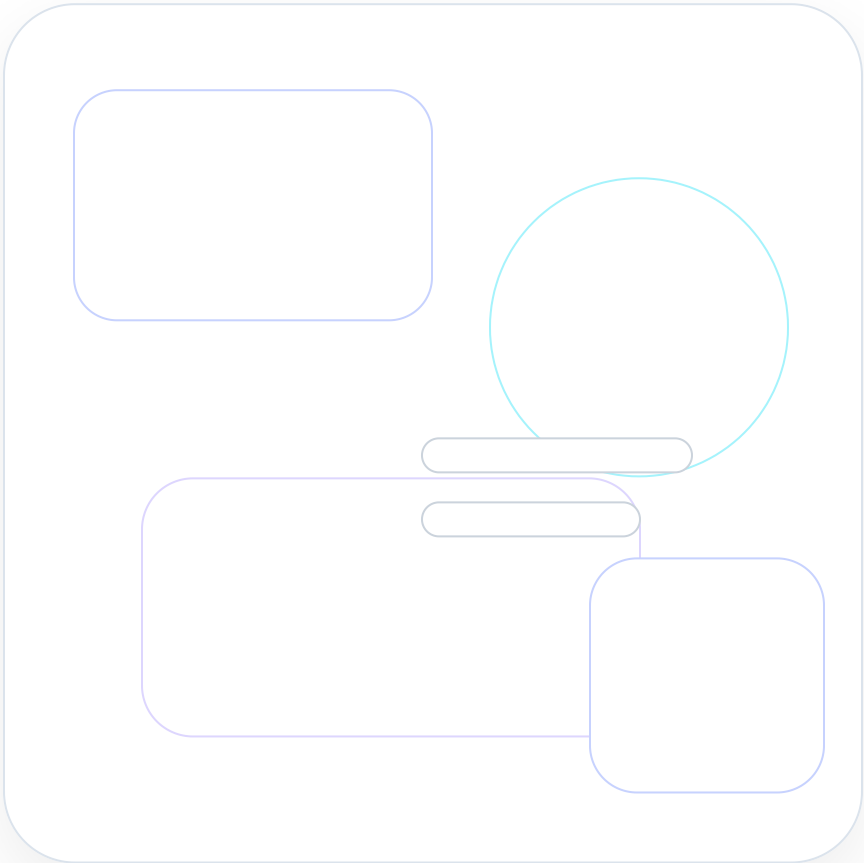
CSS: Flexbox, Grid y Responsive

Maquetación moderna para organizar elementos, construir estructuras visuales y adaptar páginas a distintos tamaños de pantalla.

IFTS 18

13/05/2026

VIRTUAL



CONTEXTO

¿Dónde estamos en la materia?

- Ya trabajamos la estructura de una página con HTML.
- Después empezamos a darle presentación visual con CSS.
- Vimos: selectores, cascada, herencia, especificidad, colores, tipografías, box model, márgenes, bordes y espaciado.
- Hoy vamos a trabajar algo clave: cómo organizar visualmente una página web.

OBJETIVO

Objetivo de la clase

- Organizar elementos con Flexbox.
- Crear estructuras con Grid.
- Diferenciar cuándo usar Flexbox y cuándo usar Grid.
- Adaptar una página a distintos tamaños de pantalla.
- Aplicar diseño responsive a una tarjeta creada en la clase anterior.

PROBLEMA

El problema que vamos a resolver

Hasta ahora podíamos crear una tarjeta visual.

```
<!-- Tarjeta simple de la clase anterior -->
<section class="presentacion">
  <h1>Hola, soy Eduardo</h1>
  <p>Estoy aprendiendo desarrollo web.</p>
  <a href="#">Ver más</a>
</section>
```

```
.presentacion {
  /* Ancho fijo: sirve para empezar, pero no es ideal en celular */
  width: 400px;

  /* Centra la tarjeta horizontalmente */
  margin: 40px auto;

  /* Espacio interno entre el borde y el contenido */
  padding: 24px;

  background-color: #f2f4f8;
  border: 1px solid #ccc;
  border-radius: 12px;
}
```

Eso sirve para un bloque individual. Pero una página real necesita organizar muchos bloques.

DISEÑO REAL

¿Qué pasa en una página real?

- Una página real no suele tener un solo elemento.
- Tiene: header, menú, sección principal, tarjetas, columnas, botones, galerías, formularios y footer.
- Necesitamos decidir qué va arriba, qué va al lado, qué queda centrado y qué baja cuando la pantalla es chica.
- Ahí aparecen Flexbox, Grid y Responsive.

CONCEPTO

¿Qué es maquetar?

"Maquetar" significa ordenar los elementos visuales de una página.

No es solamente decorar. Es construir una estructura clara.

HEADER

TÍTULO PRINCIPAL

TARJETA 1 | TARJETA 2 | TARJETA 3

FOOTER

Ese orden no lo resuelve HTML solo. Lo resolvemos con CSS.

SEPARACIÓN DE RESPONSABILIDADES

HTML estructura, CSS organiza

HTML dice qué elementos existen. CSS dice cómo se ven y cómo se ubican.

```
<!-- HTML: define la estructura del contenido -->
<main class="contenedor">
  <section class="card">
    <h2>HTML</h2>
    <p>Estructura del contenido.</p>
  </section>

  <section class="card">
    <h2>CSS</h2>
    <p>Diseño visual de la página.</p>
  </section>

  <section class="card">
    <h2>JavaScript</h2>
    <p>Interacción con el usuario.</p>
  </section>
</main>
```

Sin CSS, las secciones se apilan. Con CSS, podemos distribuirlas.

PARTE 1 — FLEXBOX

¿Qué es Flexbox?

- Flexbox es un modelo de CSS para organizar elementos dentro de un contenedor.
- Es muy útil cuando queremos distribuir elementos en una sola dirección: en fila o en columna.
- Sirve mucho para menús, botones, tarjetas, barras de navegación, centrado de elementos y alineaciones simples.

PADRE E HIJOS

La idea central de Flexbox

Flexbox tiene dos partes: el contenedor padre y los elementos hijos.

El contenedor padre recibe display: flex.

Los elementos hijos se acomodan dentro del contenedor.

```
<!-- El contenedor padre agrupa los elementos -->
<div class="contenedor">
  <div class="item">Uno</div>
  <div class="item">Dos</div>
  <div class="item">Tres</div>
</div>
```

```
.contenedor {
  /* Activa Flexbox en el contenedor padre */
  display: flex;
}
```

EJEMPLO

Ejemplo completo: primer Flexbox

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Primer Flexbox</title>
  <link rel="stylesheet" href="estilos.css">
</head>
<body>

  <!-- Contenedor padre: sobre este elemento aplicamos display:flex -->
  <section class="contenedor">
    <div class="caja">Caja 1</div>
    <div class="caja">Caja 2</div>
    <div class="caja">Caja 3</div>
  </section>

</body>
</html>
```

```
.contenedor {
  /* Pone las cajas una al lado de la otra */
  display: flex;

  /* Estos estilos son solo visuales para ver el contenedor */
  background-color: #eef2ff;
  padding: 20px;
}

.caja {
  /* Estilos visuales de cada caja */
  background-color: white;
  border: 1px solid #c7d2fe;
  padding: 20px;
  margin: 10px;
}
```

Con display: flex, las cajas dejan de apilarse y pasan a ubicarse en fila.

DIRECCIÓN

flex-direction

La propiedad flex-direction define la dirección principal.

```
.contenedor {
  display: flex;

  /* row es el valor por defecto: elementos en fila */
  flex-direction: row;
}
```

Valores principales:

```
row
column
row-reverse
column-reverse
```

Ejemplo en columna:

```
.contenedor {
  display: flex;

  /* column apila los elementos verticalmente */
  flex-direction: column;
}
```

Esto hace que los elementos se ordenen uno debajo del otro.

EJEMPLO

Ejemplo completo: fila y columna

```
<!-- Primer contenedor: elementos en fila -->
<section class="contenedor fila">
  <div class="caja">HTML</div>
  <div class="caja">CSS</div>
  <div class="caja">JS</div>
</section>

<!-- Segundo contenedor: elementos en columna -->
<section class="contenedor columna">
  <div class="caja">HTML</div>
  <div class="caja">CSS</div>
  <div class="caja">JS</div>
</section>
```

```
.contenedor {
  display: flex;
  padding: 20px;
  margin-bottom: 20px;
  background-color: #f8fafc;
  border: 1px solid #e2e8f0;
}

.fila {
  /* Dirección horizontal */
  flex-direction: row;
}

.columna {
  /* Dirección vertical */
  flex-direction: column;
}

.caja {
  padding: 16px;
  background-color: #ffffff;
  border: 1px solid #94a3b8;
}
```

DISTRIBUCIÓN HORIZONTAL

justify-content

justify-content distribuye los elementos sobre el eje principal.

Cuando usamos flex-direction: row, controla la distribución horizontal.

```
.contenedor {  
  display: flex;  
  
  /* Centra los hijos sobre el eje principal */  
  justify-content: center;  
}
```

Valores comunes:

```
flex-start  
center  
flex-end  
space-between  
space-around  
space-evenly
```

Sirve para ubicar elementos dentro del espacio disponible.

HEADER REAL

Ejemplo completo: menú con Flexbox

```
<header class="header">
  <!-- Logo a la izquierda -->
  <h1 class="logo">Mi Sitio</h1>

  <!-- Menú de navegación a la derecha -->
  <nav class="menu">
    <a href="#">Inicio</a>
    <a href="#">Cursos</a>
    <a href="#">Contacto</a>
  </nav>
</header>
```

```
.header {
  /* Ubica logo y menú en una misma fila */
  display: flex;

  /* Manda el logo a la izquierda y el menú a la derecha */
  justify-content: space-between;

  /* Centra verticalmente logo y menú */
  align-items: center;

  padding: 24px 40px;
  background-color: #0f172a;
}

.logo {
  color: white;
  font-size: 24px;
  margin: 0;
}

.menu {
  /* Ubica los enlaces del menú en fila */
  display: flex;
  gap: 20px;
}

.menu a {
  color: white;
  text-decoration: none;
  font-weight: bold;
}
```

Este es un caso real: logo a la izquierda, menú a la derecha.

ALINEACIÓN VERTICAL

align-items

align-items alinea los elementos sobre el eje secundario.

En una fila, controla la alineación vertical.

```
.contenedor {
  display: flex;

  /* Centra los hijos sobre el eje secundario */
  align-items: center;
}
```

Valores comunes:

```
flex-start
center
flex-end
stretch
```

Es muy útil para centrar elementos verticalmente dentro de un contenedor.

CENTRADO

Ejemplo completo: centrar contenido

```
<section class="hero">
  <!-- Bloque interno que queremos centrar -->
  <div class="contenido">
    <h1>Aprendé Front-End</h1>
    <p>HTML, CSS y JavaScript desde cero.</p>
    <a href="#" class="boton">Comenzar</a>
  </div>
</section>
```

```
.hero {
  height: 400px;

  /* Activa Flexbox para centrar el bloque interno */
  display: flex;
  justify-content: center; /* centro horizontal */
  align-items: center;     /* centro vertical */

  background-color: #dbeafe;
}

.contenido {
  text-align: center;
  background-color: white;
  padding: 32px;
  border-radius: 16px;
  max-width: 500px;
}

.boton {
  display: inline-block;
  padding: 12px 20px;
  background-color: #2563eb;
  color: white;
  text-decoration: none;
  border-radius: 8px;
}
```

Flexbox permite centrar todo el bloque en el centro de la sección.

ESPACIADO ENTRE HIJOS

gap

gap define el espacio entre elementos.

Es más prolijo que poner márgenes a cada hijo.

```

.contenedor {
  display: flex;

  /* Espacio uniforme entre los hijos */
  gap: 20px;
}

```

Mal enfoque:

```

.card {
  /* Evitar esto si solo buscamos separar varias tarjetas */
  margin-right: 20px;
}

```

Mejor enfoque:

```

.contenedor {
  /* Mejor: el espacio lo controla el contenedor */
  gap: 20px;
}

```

El espacio pertenece al contenedor, no a cada tarjeta.

CUANDO NO ENTRA TODO

flex-wrap

Por defecto, Flexbox intenta meter todos los elementos en una sola línea.

Si no entran, pueden quedar demasiado apretados.

Para permitir que bajen a otra línea usamos:

```
flex-wrap: wrap;
```

Ejemplo:

```
.contenedor {
  display: flex;

  /* Permite que los hijos bajen a otra línea si no entran */
  flex-wrap: wrap;

  gap: 20px;
}
```

Esto es clave para tarjetas responsive.

TARJETAS

Ejemplo completo: tarjetas con Flexbox

```
<section class="tarjetas">
  <article class="card">
    <h2>HTML</h2>
    <p>Sirve para estructurar el contenido.</p>
  </article>

  <article class="card">
    <h2>CSS</h2>
    <p>Sirve para diseñar visualmente la página.</p>
  </article>

  <article class="card">
    <h2>JavaScript</h2>
    <p>Sirve para agregar interacción.</p>
  </article>
</section>
```

```
.tarjetas {
  /* Distribuye las tarjetas en fila */
  display: flex;

  /* Si no entran, bajan a la línea siguiente */
  flex-wrap: wrap;

  gap: 20px;
  padding: 40px;
  background-color: #f1f5f9;
}

.card {
  width: 260px;
  padding: 24px;
  background-color: white;
  border: 1px solid #cbd5e1;
  border-radius: 16px;
}

.card h2 {
  margin-top: 0;
  color: #1e293b;
}

.card p {
  color: #475569;
}
```

CRITERIO PRÁCTICO

Cuándo usar Flexbox

- Conviene usar Flexbox cuando queremos ordenar elementos en una dirección principal.
- Casos típicos: menú horizontal, botones alineados, tarjetas en fila, centrar contenido y distribuir elementos dentro de un header.
- También sirve para alinear ícono + texto.
- Flexbox es ideal cuando el problema es principalmente de una dimensión.

PARTE 2 — GRID

¿Qué es CSS Grid?

CSS Grid es un sistema de maquetación en dos dimensiones.

Permite trabajar con filas y columnas.

Es ideal para estructuras más completas.

HEADER

SIDEBAR | CONTENIDO

FOOTER

O también:

CARD | CARD | CARD

CARD | CARD | CARD

COMPARACIÓN

Diferencia entre Flexbox y Grid

Flexbox trabaja mejor en una dimensión.

Elemento | Elemento | Elemento

Grid trabaja mejor en dos dimensiones.

Elemento | Elemento | Elemento
Elemento | Elemento | Elemento

Regla simple:

- Flexbox: distribución en fila o columna.
- Grid: estructura con filas y columnas.
- No compiten. Se complementan.

EJEMPLO

Primer ejemplo con Grid

```
<section class="grilla">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
  <div class="item">5</div>
  <div class="item">6</div>
</section>
```

```
.grilla {
  /* Activa CSS Grid */
  display: grid;

  /* Crea tres columnas iguales */
  grid-template-columns: 1fr 1fr 1fr;

  gap: 20px;
  padding: 40px;
  background-color: #f8fafc;
}

.item {
  padding: 30px;
  background-color: white;
  border: 1px solid #cbd5e1;
  border-radius: 12px;
  text-align: center;
}
```

1		2		3
4		5		6

Primera grilla

COLUMNAS

grid-template-columns

La propiedad grid-template-columns define las columnas.

Ejemplo con tres columnas iguales:

```
.grilla {  
  display: grid;  
  
  /* Tres columnas iguales: cada una ocupa una fracción */  
  grid-template-columns: 1fr 1fr 1fr;  
}
```

fr significa fracción del espacio disponible.

Tres veces 1fr significa: tres columnas del mismo ancho.

COLUMNAS FIJAS Y FLEXIBLES

Ejemplo con columnas distintas

```
.layout {  
  display: grid;  
  
  /* Primera columna fija, segunda flexible */  
  grid-template-columns: 250px 1fr;  
  
  gap: 24px;  
}
```

Esto crea una columna fija de 250px y una columna flexible que ocupa el resto.



Esto se usa mucho en paneles, dashboards o páginas con menú lateral.

LAYOUT

Ejemplo completo: layout con sidebar

```
<div class="layout">
  <!-- Columna lateral -->
  <aside class="sidebar">
    <h2>Menú</h2>
    <ul>
      <li>Inicio</li>
      <li>Cursos</li>
      <li>Perfil</li>
    </ul>
  </aside>

  <!-- Columna principal -->
  <main class="contenido">
    <h1>Panel principal</h1>
    <p>Este es el contenido principal de la página.</p>
  </main>
</div>
```

```
.layout {
  display: grid;

  /* Sidebar de 240px + contenido que ocupa el resto */
  grid-template-columns: 240px 1fr;

  gap: 24px;
  padding: 24px;
}

.sidebar {
  background-color: #e0f2fe;
  padding: 24px;
  border-radius: 16px;
}

.contenido {
  background-color: #ffffff;
  padding: 24px;
  border: 1px solid #e2e8f0;
  border-radius: 16px;
}
```

REPETICIÓN

repeat()

Cuando queremos repetir columnas, podemos usar repeat().

En vez de escribir:

```
grid-template-columns: 1fr 1fr 1fr;
```

Podemos escribir:

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

Es más corto y más mantenible.

```
.grilla {  
  display: grid;  
  
  /* Repite 3 veces una columna de 1fr */  
  grid-template-columns: repeat(3, 1fr);  
  
  gap: 24px;  
}
```

MÍNIMO Y MÁXIMO

minmax()

minmax() permite definir un tamaño mínimo y máximo.

```
/* Cada columna mide como mínimo 200px y como máximo 1fr */
grid-template-columns: repeat(3, minmax(200px, 1fr));
```

Esto significa que cada columna mide como mínimo 200px y como máximo ocupa una fracción disponible.

Pero todavía falta algo para que sea realmente responsive.

Ahí aparece auto-fit.

RESPONSIVE AUTOMÁTICO

Grid responsive con auto-fit

Una forma muy útil de hacer grillas responsive es:

```
.grid {
  display: grid;

  /* auto-fit crea tantas columnas como entren */
  /* minmax evita que cada tarjeta sea demasiado angosta */
  grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));

  gap: 24px;
}
```

Esto permite que el navegador decida cuántas columnas entran.

En pantalla grande:

Card | Card | Card

En pantalla mediana:

Card | Card
Card

En celular:

Card
Card
Card

TARJETAS RESPONSIVE

Ejemplo completo: grilla responsive de tarjetas

```
<section class="cursos">
  <article class="curso">
    <h2>HTML</h2>
    <p>Aprendé a estructurar páginas web.</p>
    <a href="#">Ver curso</a>
  </article>

  <article class="curso">
    <h2>CSS</h2>
    <p>Aprendé a diseñar interfaces visuales.</p>
    <a href="#">Ver curso</a>
  </article>

  <article class="curso">
    <h2>JavaScript</h2>
    <p>Agregá interacción a tus páginas.</p>
    <a href="#">Ver curso</a>
  </article>
</section>
```

```
.cursos {
  display: grid;

  /* Grilla responsive sin media query */
  grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));

  gap: 24px;
  padding: 40px;
  background-color: #f8fafc;
}

.curso {
  background-color: white;
  padding: 24px;
  border: 1px solid #cbd5e1;
  border-radius: 16px;
}

.curso h2 {
  margin-top: 0;
  color: #0f172a;
}

.curso p {
  color: #475569;
}

.curso a {
  display: inline-block;
  margin-top: 12px;
  color: white;
```

PARTE 3 — RESPONSIVE DESIGN

¿Qué significa responsive?

- Un diseño responsive es un diseño que se adapta a diferentes pantallas.
- No debería verse bien solo en una notebook.
- También debería funcionar en celulares, tablets, monitores grandes y pantallas chicas.
- La idea no es hacer una página distinta para cada dispositivo. La idea es que el mismo sitio se adapte.

ANCHO FIJO

Problema típico de una página no responsive

Ejemplo problemático:

```
.card {  
  /* Problema: en celulares chicos puede generar scroll horizontal */  
  width: 600px;  
}
```

En una pantalla grande puede verse bien.

Pero en un celular de 360px de ancho, esa tarjeta no entra.

Entonces aparece scroll horizontal o el contenido queda cortado.

Una regla más flexible sería:

```
.card {  
  /* Se adapta al ancho disponible */  
  width: 100%;  
  
  /* Pero no crece indefinidamente en escritorio */  
  max-width: 600px;  
}
```

BUENAS PRÁCTICAS

Medidas fijas vs medidas flexibles

Medidas fijas:

```
/* Medida fija: poco flexible */  
width: 600px;
```

Medidas flexibles:

```
/* Medidas flexibles: mejor para responsive */  
width: 100%;  
max-width: 600px;
```

La segunda opción es mejor para responsive.

Permite que el elemento crezca hasta cierto límite, pero también se achique si la pantalla es menor.

CONFIGURACIÓN NECESARIA

Viewport

El viewport es el área visible de la página en el navegador.

Para que una página se adapte correctamente en celulares, el HTML debe tener esta línea:

```
<!-- Indica al navegador que use el ancho real del dispositivo -->
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Sin esta línea, el navegador móvil puede mostrar la página como si fuera una versión de escritorio reducida.

CSS CONDICIONAL

Media queries

Las media queries permiten aplicar CSS según el tamaño de pantalla.

Ejemplo:

```
@media (max-width: 600px) {
  .contenedor {
    /* En pantallas chicas, los elementos se apilan */
    flex-direction: column;
  }
}
```

Esto significa: cuando la pantalla tenga hasta 600px de ancho, el contenedor pasa a columna.

RESPONSIVE APLICADO

Ejemplo completo: header responsive

```
<header class="header">
  <h1 class="logo">Mi Sitio</h1>

  <nav class="menu">
    <a href="#">Inicio</a>
    <a href="#">Cursos</a>
    <a href="#">Contacto</a>
  </nav>
</header>
```

```
.header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 24px 40px;
  background-color: #0f172a;
}

.logo {
  color: white;
  margin: 0;
}

.menu {
  display: flex;
  gap: 20px;
}

.menu a {
  color: white;
  text-decoration: none;
}

@media (max-width: 600px) {
  .header {
    /* El header deja de ser horizontal */
    flex-direction: column;
    gap: 16px;
    text-align: center;
  }

  .menu {
    /* Los enlaces del menú también se apilan */
    flex-direction: column;
    gap: 10px;
  }
}
```

En escritorio, el menú está en fila. En celular, se apila.

ENFOQUE RECOMENDADO

Mobile first

Una buena práctica es diseñar primero para celular.

Eso se llama mobile first.

Primero hacemos el diseño simple para pantallas chicas.

Después agregamos mejoras para pantallas más grandes.

```
.cards {
  display: grid;

  /* Base mobile: una sola columna */
  grid-template-columns: 1fr;
  gap: 20px;
}

@media (min-width: 768px) {
  .cards {
    /* Desde tablet/escritorio: tres columnas */
    grid-template-columns: repeat(3, 1fr);
  }
}
```

ENFOQUE ALTERNATIVO

Desktop first

También existe el enfoque contrario: desktop first.

Primero diseñamos para escritorio.

Después corregimos para celular.

```
.cards {
  display: grid;

  /* Base escritorio: tres columnas */
  grid-template-columns: repeat(3, 1fr);
  gap: 24px;
}

@media (max-width: 768px) {
  .cards {
    /* Corrección para pantallas chicas */
    grid-template-columns: 1fr;
  }
}
```

Funciona, pero muchas veces lleva a parches.

Para proyectos nuevos, suele ser más ordenado pensar mobile first.

HTML COMPLETO

Ejemplo completo integrador

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Landing de Cursos</title>
  <link rel="stylesheet" href="estilos.css">
</head>
<body>

  <!-- Encabezado principal de la página -->
  <header class="header">
    <h1>Front-End IFTS</h1>

    <nav class="menu">
      <a href="#">Inicio</a>
      <a href="#">Cursos</a>
      <a href="#">Contacto</a>
    </nav>
  </header>

  <main>
    <!-- Sección principal o hero -->
    <section class="hero">
      <h2>Aprendé desarrollo web desde cero</h2>
      <p>HTML, CSS y JavaScript aplicados a páginas reales.</p>
      <a href="#" class="boton">Empezar ahora</a>
    </section>

    <!-- Grilla de tarjetas -->
    <section class="cards">
      <article class="card">
        <h3>HTML</h3>
        <p>Estructura y contenido semántico.</p>
      </article>

      <article class="card">
        <h3>CSS</h3>
        <p>Diseño visual y maquetación.</p>
      </article>

      <article class="card">
        <h3>JavaScript</h3>
        <p>Interacción con el usuario.</p>
      </article>
    </section>
  </main>

</body>
</html>
```

CSS COMPLETO

CSS del ejemplo integrador

```
* {
  /* Hace más predecible el cálculo de anchos */
  box-sizing: border-box;
}

body {
  margin: 0;
  font-family: Arial, sans-serif;
  background-color: #f8fafc;
  color: #0f172a;
}

.header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 24px 40px;
  background-color: #0f172a;
  color: white;
}

.header h1 {
  margin: 0;
}

.menu {
  display: flex;
  gap: 20px;
}

.menu a {
  color: white;
  text-decoration: none;
}

.hero {
  max-width: 900px;
  margin: 40px auto;
  padding: 40px 24px;
  text-align: center;
}

.hero h2 {
  font-size: 40px;
  margin-bottom: 16px;
}

.hero p {
  font-size: 18px;
  color: #475569;
}

.boton {
  display: inline-block;
  margin-top: 20px;
  padding: 12px 20px;
```

MEDIA QUERY

Responsive del ejemplo integrador

```
@media (max-width: 768px) {
  .header {
    /* Header vertical en pantallas chicas */
    flex-direction: column;
    gap: 16px;
    text-align: center;
  }

  .menu {
    /* Menú vertical en celular */
    flex-direction: column;
    gap: 10px;
  }

  .hero h2 {
    /* Título menos grande en celular */
    font-size: 30px;
  }

  .cards {
    /* Las tres tarjetas pasan a una sola columna */
    grid-template-columns: 1fr;
  }
}
```

Con estas reglas:

- El header deja de estar en fila.
- El menú baja en columna.
- El título se achica.
- Las tarjetas pasan de tres columnas a una sola.

EVITAR ESTOS PROBLEMAS

Errores comunes

Error 1: usar width fijo para todo.

```
.card {
  /* Error común: ancho fijo demasiado grande */
  width: 700px;
}
```

Error 2: no usar box-sizing: border-box.

```
* {
  /* Buena práctica: evita cálculos raros con padding y border */
  box-sizing: border-box;
}
```

- Error 3: forzar márgenes en vez de usar gap.
- Error 4: usar Flexbox para estructuras complejas donde conviene Grid.
- Error 5: no probar la página en tamaños chicos.

CRITERIO PRÁCTICO

Flexbox o Grid: decisión práctica

- Usar Flexbox cuando queremos alinear elementos en fila, centrar elementos, armar un menú, ubicar botones uno al lado del otro o resolver una distribución simple.
- Usar Grid cuando tenemos filas y columnas, queremos una galería, queremos un layout completo, queremos tarjetas distribuidas en varias columnas o queremos controlar una estructura más grande.

TRABAJO EN CLASE

Ejercicio rápido en clase

Vamos a tomar el ejercicio de la clase anterior: la tarjeta de presentación visual hecha con HTML y CSS.

Ahora hay que mejorarla aplicando lo visto hoy.

La idea es transformar una tarjeta individual en una pequeña sección responsive.

ACTIVIDAD

Consigna del ejercicio rápido

Tomar el archivo de la clase anterior y agregar:

- 1. Un contenedor principal con varias tarjetas. Crear al menos tres tarjetas: presentación personal, curso que están haciendo, habilidad o tecnología, o proyecto ficticio.
- 2. Aplicar Flexbox o Grid. La recomendación para este ejercicio es usar Grid.
- 3. Agregar separación entre tarjetas usando gap: 20px.
- 4. Hacerlo responsive. En escritorio, las tarjetas pueden verse en columnas. En celular, deben verse una debajo de la otra.

display: flex;

display: grid;

gap: 20px;

@media (max-width: 768px) {
 .contenedor {
 grid-template-columns: 1fr;
 }
}

HTML SUGERIDO

Estructura sugerida del ejercicio

```
<main class="contenedor">
  <section class="tarjeta">
    <h2>Eduardo</h2>
    <p>Estudiante de desarrollo web.</p>
    <a href="#">Ver perfil</a>
  </section>

  <section class="tarjeta">
    <h2>HTML</h2>
    <p>Estoy aprendiendo estructura semántica.</p>
    <a href="#">Ver tema</a>
  </section>

  <section class="tarjeta">
    <h2>CSS</h2>
    <p>Estoy aprendiendo diseño visual y responsive.</p>
    <a href="#">Ver tema</a>
  </section>
</main>
```

CSS SUGERIDO

CSS sugerido del ejercicio

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

body {
  margin: 0;
  font-family: Arial, sans-serif;
  background-color: #f1f5f9;
}

.contenedor {
  /* Grilla principal del ejercicio */
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 24px;

  max-width: 1000px;
  margin: 40px auto;
  padding: 20px;
}

.tarjeta {
  background-color: white;
  padding: 24px;
  border: 1px solid #cbd5e1;
  border-radius: 16px;
}

.tarjeta h2 {
  margin-top: 0;
  color: #0f172a;
}

.tarjeta p {
  color: #475569;
}

.tarjeta a {
  display: inline-block;
  margin-top: 12px;
  padding: 10px 16px;
  background-color: #2563eb;
  color: white;
  text-decoration: none;
  border-radius: 8px;
}

@media (max-width: 768px) {
  .contenedor {
    /* En celular, una tarjeta debajo de la otra */
    grid-template-columns: 1fr;
  }
}
```

VALIDACIÓN

Resultado esperado

En pantalla grande:

Tarjeta 1 | Tarjeta 2 | Tarjeta 3

En pantalla chica:

Tarjeta 1
Tarjeta 2
Tarjeta 3

- El contenido debe seguir siendo legible.
- No debe aparecer scroll horizontal.
- Las tarjetas deben mantener separación y buen espaciado.

CIERRE

Cierre de la clase

- Hoy vimos tres herramientas centrales para maquetar interfaces.
- Flexbox: para alinear y distribuir elementos en una dimensión.
- Grid: para construir estructuras con filas y columnas.
- Responsive design: para que la página funcione en distintos tamaños de pantalla.
- Estas herramientas son la base para pasar de una página “armada” a una página realmente usable.

PUENTE

Próxima clase

- En la próxima clase empezamos con JavaScript.
- Vamos a trabajar introducción a JavaScript, variables, tipos de datos, operadores y condicionales.
- A partir de ahí vamos a empezar a incorporar comportamiento a las páginas.