1. ¿Qué es Flexbox?

Imagina que tienes una caja y dentro de ella, varios objetos. Flexbox, o Flexible Box Layout, es una forma de alinear y distribuir esos objetos en una sola dimensión: ya sea en una fila o en una columna. Piensa en él como un organizador de una sola dirección.

Es perfecto para:

- Centrar elementos: Olvídate de los trucos antiguos con márgenes automáticos. Con Flexbox, centrar algo es tan simple como un par de líneas de código.
- Crear barras de navegación: Es ideal para alinear los elementos de un menú horizontalmente.
- Distribuir espacio: Puedes hacer que los elementos ocupen el espacio disponible de manera uniforme o darle a cada uno un tamaño específico.

¿Cómo funciona?

Se basa en dos tipos de elementos:

- Contenedor flexible (`flex container`): Es el padre, la caja que contiene los elementos.
- Elementos flexibles (`flex items`): Son los hijos, los objetos dentro del contenedor.

La magia de Flexbox ocurre cuando aplicas propiedades al contenedor padre (como `display: flex;`) y a los elementos hijos.

Propiedades Clave

- flex-direction → define si los ítems van en **fila** o **columna**.
- justify-content → controla la alineación en el eje principal (inicio, final, centro, espacio entre elementos).
- align-items → alinea en el eje cruzado (vertical si la dirección es row, horizontal si es column).
- flex-wrap → permite que los elementos salten a otra línea si no caben.
- gap → espacio uniforme entre ítems.

- flex (shorthand) → controla cómo un ítem crece, se encoge y cuál es su tamaño base.

2. ¿Qué es Grid?

Ahora, si Flexbox es un organizador en una sola dimensión, Grid es un organizador en dos dimensiones. CSS Grid Layout te permite crear una cuadrícula de filas y columnas, como una tabla o una hoja de cálculo, para organizar elementos en tu página. Es ideal para crear el esqueleto principal de tu sitio web, como la cabecera, el pie de página, la barra lateral y el contenido principal.

Es perfecto para:

- Diseños complejos de página: Si necesitas una estructura que involucre filas y columnas, Grid es tu mejor amigo.
- Posicionar elementos: Puedes colocar un elemento en una celda específica de la cuadrícula, haciendo que los layouts complejos sean increíblemente sencillos.
- Crear áreas de diseño: Puedes nombrar áreas de tu cuadrícula (por ejemplo, `header`, `main`, `sidebar`) para organizar tus elementos de forma intuitiva.

¿Cómo funciona?

Al igual que Flexbox, Grid se basa en dos tipos de elementos:

- Contenedor de cuadrícula (`grid container`): El padre que define la cuadrícula.
- Elementos de cuadrícula (`grid items`): Los hijos que se colocan dentro de la cuadrícula.

Al aplicar display: grid; al contenedor padre, puedes definir la estructura de la cuadrícula con propiedades como:

grid-template-columns / grid-template-rows → definen las filas y columnas.

grid-template-areas → te permite nombrar zonas (ej. header, main, footer).

fr → unidad de fracción: reparte proporcionalmente el espacio disponible.

auto → adapta el tamaño al contenido.

gap → espacio uniforme entre filas y columnas.

grid-column y grid-row → permiten a un ítem ocupar varias celdas.

3. ¿Cuándo usar cada uno?

Esta es la pregunta clave. No son competidores; son complementos.

- Usa Flexbox cuando: Necesites alinear elementos en una sola dirección (fila o columna). Piensa en la barra de navegación, en los botones de un formulario o en un grupo de tarjetas que quieres que se distribuyan uniformemente.
- Usa Grid cuando: Necesites un diseño en dos dimensiones (filas y columnas). Es la herramienta perfecta para el layout principal de tu página web, el esqueleto.

¿Cómo se usan?

Propiedades Clave de Flexbox

🔹 flex-direction → define si los ítems van en fila o columna.

```
.contenedor {
  display: flex;
  flex-direction: row;    /* Los hijos se colocan en fila (izquierda → derecha) */
  /* flex-direction: column;  Los hijos se apilan en columna (arriba → abajo) */
}
```

🔹 justify-content → controla la alineación en el eje principal.

```
.contenedor {
  display: flex;
  justify-content: center;         /* Centra todos los hijos en el eje principal */
  /* Otras opciones: flex-start | flex-end | space-between | space-around |
space-evenly */
```

```
}
```

🔹 align-items → alinea en el eje cruzado.

```
.contenedor {
  display: flex;
  align-items: center;          /* Centra verticalmente (si direction es row) */
  /* Otras opciones: flex-start | flex-end | stretch | baseline */
}
```

🔹 flex-wrap → permite que los elementos salten a otra línea si no caben.

```
.contenedor {
  display: flex;
  flex-wrap: wrap;              /* Los ítems pasan a otra fila si no caben */
  /* nowrap (por defecto) → todos en una sola línea */
}
```

🔹 gap → espacio uniforme entre ítems.

```
.contenedor {
  display: flex;
  gap: 20px;                    /* Espacio de 20px entre los hijos */
}
```

🔹 flex (shorthand) → controla crecimiento, encogimiento y tamaño base.

```
.item {
  flex: 1 1 auto;               /* grow: 1, shrink: 1, basis: auto */
  /* Ejemplos:
```

```
    flex: 1;          → todos crecen por igual y se reparten el espacio.
    flex: 0 0 200px;  → ancho fijo de 200px, no crece ni se encoge.
    flex: 2;          → este ítem crece el doble que uno con flex: 1.
  */
}
```

Propiedades Clave de Grid

🔹 grid-template-columns y grid-template-rows → definen filas y columnas.

```
.grid {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;   /* 3 columnas: pequeña, grande, pequeña */
  grid-template-rows: auto 1fr auto;    /* Cabecera auto, cuerpo flexible, footer auto */
}
```

🔹 grid-template-areas → define zonas con nombres.

```
.grid {
  display: grid;
  grid-template-areas:
    "header header header"
    "sidebar main main"
    "footer footer footer";
  grid-template-columns: 200px 1fr 1fr;
  grid-template-rows: auto 1fr auto;
}

.header { grid-area: header; }
.sidebar { grid-area: sidebar; }
.main    { grid-area: main; }
```

```
.footer  { grid-area: footer; }
```

🔹 fr → unidad de fracción.

```
.grid {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;  /* 1 parte, 2 partes, 1 parte → total 4 fr */
}
```

🔹 auto → adapta el tamaño al contenido.

```
.grid {
  display: grid;
  grid-template-columns: auto 1fr;   /* 1ª columna según contenido, 2ª flexible */
}
```

🔹 gap → espacio entre filas y columnas.

```
.grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 16px;   /* Espacio de 16px en filas y columnas */
}
```

4. Analizamos la DEMO.

Esta sección desglosa las herramientas que has visto en las demos para que sepas exactamente qué hace cada control y cómo influye en tu diseño.

Flexbox: El Arte de la Alineación Unidireccional

Flexbox se trata de alinear y distribuir elementos en una sola línea. Al jugar con el playground, estás controlando el comportamiento del Contenedor Flexible. Los hijos se adaptan a las órdenes que tú le das.

- Dirección (flex-direction): Es la propiedad que define el eje principal de tu diseño. Es la primera decisión que debes tomar.
    - row: El eje principal es horizontal. Los elementos fluyen de izquierda a derecha. Es el valor por defecto.
    - column: El eje principal es vertical. Los elementos fluyen de arriba abajo.
    - row-reverse / column-reverse: Invierten el orden de los elementos a lo largo de su respectivo eje.
- Alineación en el Eje Principal (justify-content): Esta propiedad se encarga de cómo se distribuye el espacio libre entre los ítems a lo largo de la dirección que has elegido. Es tu herramienta para centrar, separar o agrupar elementos.
    - flex-start / flex-end / center: Agrupa los elementos al principio, al final o en el centro del contenedor.
    - space-between: Pega el primer y el último ítem a los bordes y distribuye el espacio libre de forma equitativa entre los demás.
    - space-around / space-evenly: Distribuyen el espacio de manera más uniforme, incluyendo los bordes del contenedor en el cálculo.
- Alineación en el Eje Cruzado (align-items): Este control alinea los ítems en la dirección perpendicular al eje principal. Si tu flex-direction es row, este eje es el vertical, y si es column, es el horizontal.
    - stretch: Es el valor por defecto. Estira los ítems para que ocupen toda la altura o anchura del contenedor.
    - flex-start / flex-end / center: Agrupan los ítems en el inicio, el final o el centro del eje cruzado, respectivamente.
- Salto de Línea (flex-wrap): Es una propiedad vital para el diseño responsivo. Controla si los ítems deben quedarse en una sola línea o si pueden "envolverse" y saltar a la siguiente.
    - nowrap: Por defecto, los ítems se comprimen para caber en una sola línea.
    - wrap: Permite que los ítems salten a una nueva línea cuando el espacio se agota.
- Espacio entre Elementos (gap): Una propiedad moderna y sencilla que crea un espacio consistente entre los ítems, sin necesidad de usar márgenes. Puedes aplicarlo a Flexbox y a Grid, lo que lo hace muy conveniente.

Estas son nuestras opciones de código.

```
/* 1. EL CONTENEDOR FLEXIBLE: El elemento padre */
.contenedor-flexible {
  /* La propiedad fundamental: convierte el contenedor en un flexbox */
  display: flex;

  /* 2. PROPIEDADES DEL EJE PRINCIPAL */
  /* Define la dirección de los ítems */
  flex-direction: row; /* Opciones: row | column | row-reverse |
column-reverse */

  /* Controla si los ítems se envuelven a una nueva línea */
  flex-wrap: wrap; /* Opciones: nowrap | wrap | wrap-reverse */

  /* Abreviatura para las dos propiedades anteriores */
  /* flex-flow: row wrap; */

  /* 3. PROPIEDADES DE ALINEACIÓN Y DISTRIBUCIÓN */
  /* Distribuye los ítems a lo largo del eje principal */
  justify-content: center; /* Opciones: flex-start | flex-end | center |
space-between | space-around | space-evenly */

  /* Alinea los ítems a lo largo del eje cruzado (perpendicular) */
  align-items: center; /* Opciones: flex-start | flex-end | center | stretch |
baseline */

  /* Alinea todo el contenido flexbox si hay varias líneas */
  align-content: stretch; /* Opciones: flex-start | flex-end | center |
space-between | space-around | stretch */

  /* 4. ESPACIO ENTRE LOS ÍTEMS */
```

```css
  /* Crea un espacio uniforme entre los ítems */
  gap: 16px; /* También puedes usar row-gap y column-gap */

  /* 5. ESTILOS VISUALES (para el ejemplo) */
  height: 300px;
  background-color: #f0f0f0;
  border: 2px solid #333;
}

/* 6. PROPIEDADES DE LOS ÍTEMS (los hijos del contenedor) */
.item-flexible {
  /* Controla cómo crece o se encoge un ítem */
  flex: 1 1 auto; /* Abreviatura de: flex-grow, flex-shrink, flex-basis */

  /* Alinear un ítem individualmente (sobrescribe align-items del padre) */
  align-self: auto; /* Opciones: auto | flex-start | flex-end | center |
stretch | baseline */

  /* Cambiar el orden de un ítem */
  order: 0; /* Por defecto es 0, puedes usar números positivos o negativos */

  /* Estilos visuales (para el ejemplo) */
  background-color: #007BFF;
  color: white;
  padding: 20px;
  text-align: center;
}
```

Atributos Clave y su Propósito

- display: flex;: La propiedad más importante. Transforma un elemento en un contenedor flexible, haciendo que sus hijos se conviertan en ítems flexibles. Sin esto, ninguna otra propiedad de Flexbox funcionará. .
- flex-direction: Define el eje principal de tu diseño. Elige row si quieres que tus ítems se organicen horizontalmente (ideal para barras de navegación) o column si los quieres en vertical (perfecto para una lista de ítems o una columna de texto).
- justify-content: Controla cómo se distribuye el espacio a lo largo del eje principal. Es la clave para centrar, separar o agrupar elementos horizontalmente en una fila, o verticalmente en una columna.
- align-items: Controla la alineación de los ítems a lo largo del eje cruzado (perpendicular). Úsalo para centrar elementos verticalmente en una fila o horizontalmente en una columna.
- flex: Esta es una de las propiedades más poderosas para los ítems individuales. Es una abreviatura para controlar cómo un ítem crece (flex-grow), se encoge (flex-shrink) y su tamaño base (flex-basis). Por ejemplo, flex: 1; le dice al ítem que crezca para ocupar el espacio disponible.
- gap: Una propiedad moderna y muy útil que crea un espacio consistente entre los ítems, tanto en horizontal como en vertical, sin necesidad de usar márgenes.

En la demo que te he preparado puedes hacerte una idea de cómo funciona una flex y sus hijos pero si quieres verlo de verdad. Copia este html y empieza a probar el css anterior.

```
<!DOCTYPE html>

<html lang="es">

<head>

    <meta charset="UTF-8" />

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

    <title>Demostración de Flexbox</title>

    <link rel="stylesheet" href="1.css" />

</head>

<body>
```

```
    <h2>Ajusta el CSS para ver los cambios en Flexbox</h2>

    <p>Abre este archivo en tu navegador y, en tu editor de código, cambia los valores del archivo styles.css para experimentar con las propiedades de Flexbox.</p>



    <div class="contenedor-flexible">

        <div class="item-flexible">Item 1</div>

        <div class="item-flexible">Item 2</div>

        <div class="item-flexible">Item 3</div>

    </div>



</body>

</html>
```

Está comentado y puedes hacerlo tú sólo pero me paro en una línea que vas a encontrar mucho y necesitas saber que es.

```
 /* Controla cómo crece o se encoge un ítem */

    flex: 1 1 auto; /* Abreviatura de: flex-grow, flex-shrink, flex-basis */
```

Esa línea de código, flex: 1 1 auto;, es una propiedad abreviada que controla cómo un **ítem flexible** se redimensiona dentro de un contenedor Flexbox. Es una de las propiedades más poderosas para los elementos hijos.

Esa línea de código, flex: 1 1 auto;, es una propiedad abreviada que controla cómo un **ítem flexible** se redimensiona dentro de un contenedor Flexbox. Es una de las propiedades más poderosas para los elementos hijos.

Entendiendo la propiedad flex

La propiedad flex es una abreviatura para tres propiedades individuales que, en conjunto, definen el comportamiento de un ítem: flex-grow, flex-shrink y flex-basis.

1. **flex-grow (Crecer) - El primer valor:** Es un número que le dice al ítem cuánto debe **crecer** si hay espacio extra en el contenedor. Si todos los ítems tienen un flex-grow: 1, crecerán de manera uniforme para llenar el espacio. Si un ítem tiene flex-grow: 2, crecerá el doble que los demás.
2. **flex-shrink (Encoger) - El segundo valor:** Un número que le dice al ítem cuánto debe **encogerse** si no hay suficiente espacio en el contenedor. Un valor de 1 (el valor por defecto) le permite encogerse, mientras que un valor de 0 evita que se haga más pequeño que su tamaño inicial.
3. **flex-basis (Tamaño base) - El tercer valor:** Es el tamaño inicial del ítem antes de que el espacio libre se distribuya (flex-grow) o se contraiga (flex-shrink). Puede ser un valor absoluto como 200px o un valor relativo como 20%. El valor auto (como en tu ejemplo) le dice al ítem que utilice el tamaño definido en sus propiedades de ancho o alto.

En tu ejemplo: flex: 1 1 auto;

- flex-grow: 1;: El ítem **crecerá** si hay espacio extra disponible. Si todos los ítems tienen este valor, crecerán de forma equitativa.
- flex-shrink: 1;: El ítem **se encogerá** si no hay suficiente espacio, hasta su tamaño mínimo.
- flex-basis: auto;: El tamaño base del ítem estará determinado por su contenido o por cualquier propiedad de width que tenga definida.

En resumen, la propiedad flex te da un control completo sobre cómo los elementos se ajustan dinámicamente, lo que es fundamental para crear diseños flexibles y responsivos.

Claro. En CSS Grid, el valor `auto` para el tamaño de una columna o fila se refiere a la **capacidad de adaptación al contenido que tiene dentro**. Es una de las características más potentes de Grid.

¿Cómo funciona auto? 🤔

Cuando usas auto, le estás diciendo al navegador: "Permite que esta columna (o fila) tenga el tamaño que necesite para mostrar su contenido, ni más ni menos".

El navegador calcula el tamaño del contenido y ajusta el tamaño de la columna para que quepa perfectamente. Por ejemplo, si una columna tiene solo una palabra, será estrecha. Si tiene una frase larga, se ensanchará para que la frase quepa sin desbordarse. .

La diferencia entre auto y 1fr

Es común confundir auto con 1fr, pero tienen propósitos distintos:

- 1fr (unidad de fracción): Reparte el espacio disponible que le queda al contenedor después de que otros elementos ya tienen un tamaño. Si tienes dos columnas de 1fr, se reparten el 50% del espacio sobrante cada una.
- auto: Se refiere al espacio que necesita el contenido. Se expande o se encoge según lo requiera el texto o la imagen que contiene.

Combinando auto y fr

La verdadera magia ocurre cuando los combinas. Por ejemplo:

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

1. La primera columna (auto) tomará exactamente el tamaño de su contenido. Si es un menú de navegación, su ancho será el que necesite el texto más largo.
2. Las dos columnas restantes (1fr 1fr) se repartirán el espacio que sobre en el contenedor, de forma equitativa.

Esto te permite crear diseños donde un elemento se adapta al tamaño de su contenido, mientras que otros elementos se reparten de manera flexible el espacio restante. Es una técnica fundamental para crear layouts eficientes y responsivos.

¿Cómo se usan Grid y Flex? Aunque ya lo hemos dicho lo repetimos aunque como siempre esto es sólo una manera de hacerlo.

Imagina que estás construyendo una página. Usarías Grid para definir la estructura general: un área para el encabezado, una para la barra lateral y otra para el contenido principal. Dentro del área del contenido principal, podrías usar Flexbox para alinear una serie de tarjetas de productos, centrar una imagen y su título, o distribuir los botones de un formulario.

En resumen: Usa Grid para el macro-layout (el esqueleto de la página) y Flexbox para el micro-layout (alinear y organizar elementos dentro de esos esqueletos).

Dominar ambas herramientas te dará un control sin precedentes sobre tus diseños y te permitirá crear páginas web que no solo se vean bien, sino que también sean robustas y adaptables a cualquier dispositivo.

5. DICCIONARIO.

Grid: sistema de **rejilla** para el **layout** general.

- **Flex**: sistema de **línea/columna** para **alinear y distribuir** elementos dentro de un bloque.
- **fr**: fracción del espacio disponible (solo en Grid).
- **em**: unidad relativa al tamaño de letra del elemento (1em = su font-size).
- **gap**: hueco entre elementos **hijos** en Grid o Flex.
- **padding**: relleno **interno** de un elemento (entre su contenido y su borde).
- **justify-content**: alinea en el **eje principal**.
- **align-items**: alinea en el **eje cruzado**.
- **wrap**: permite pasar a **otra línea** si no caben (Flex).

Ahora visita este enlace y empecemos a practicar

https://maquetandoweb.netlify.app/

6. **MANOS A LA OBRA**

Vamos a crear desde cero una página muy sencilla donde:

- La página completa usa CSS Grid para colocar header, banner lateral, main, aside y footer.
- Dentro del main, habrá un grid de 9 tarjetas (3×3).
- En cada tarjeta practicaremos Flexbox para ordenar texto e imágenes.

Iremos por pasos cortos y con pruebas visibles. Al final podrás replicar exactamente el resultado.

Lo primero es copiar el html. Esta vez nos vamos a centrar en el css.
Te doy el html para que lo copies tal cual por que lo que nos vamos a centrar es en ver el css
copialo y abrelo en tu navegador ¿se parece en algo a la página que te di?

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Grid global + Flex en tarjetas</title>


</head>
<body>
  <div class="grid-container">
    <header class="site-header">
      <h1>Mi Página</h1>
      <nav>
        <ul class="menu">
          <li><a href="#">Inicio</a></li>
          <li><a href="#">Servicios</a></li>
          <li><a href="#">Contacto</a></li>
        </ul>
      </nav>
    </header>


    <aside class="banner">Banner lateral</aside>


    <main>
      <section class="articulos">
        <!-- 9 tarjetas (3x3) -->
```

```
      <article class="card card--meta">
        <p class="card-title">Tarjeta 1 — Texto breve</p>
        <time datetime="2025-09-17" class="card-date">17/09/2025</time>
      </article>
      <article class="card card--meta">
        <p class="card-title">Tarjeta 2 — Texto breve</p>
        <time datetime="2025-09-18" class="card-date">18/09/2025</time>
      </article>
      <article class="card card--meta">
        <p class="card-title">Tarjeta 3 — Texto breve</p>
        <time datetime="2025-09-19" class="card-date">19/09/2025</time>
      </article>

      <article class="card card--media">
        <img src="https://placehold.co/120x120" alt="Imagen ejemplo" />
        <p class="card-caption">Tarjeta 4 — Texto principal</p>
      </article>
      <article class="card card--media">
        <img src="https://placehold.co/120x120" alt="Imagen ejemplo" />
        <p class="card-caption">Tarjeta 5 — Texto principal</p>
      </article>
      <article class="card card--media">
        <img src="https://placehold.co/120x120" alt="Imagen ejemplo" />
        <p class="card-caption">Tarjeta 6 — Texto principal</p>
      </article>

      <article class="card card--row">
        <img class="avatar" src="https://placehold.co/80x80" alt="Avatar" />
        <div class="row-text">
          <p class="card-title">Tarjeta 7 — Con acciones</p>
          <small class="muted">Texto corto de apoyo</small>
```

```
          </div>
          <button class="btn">Ver</button>
        </article>
        <article class="card card--row card--row-center">
          <img class="avatar" src="https://placehold.co/80x80" alt="Avatar" />
          <div class="row-text">
            <p class="card-title">Tarjeta 8 — Centrada</p>
            <small class="muted">Alineación diferente</small>
          </div>
          <button class="btn">Abrir</button>
        </article>
        <article class="card card--row card--row-space">
          <img class="avatar" src="https://placehold.co/48x48" alt="Avatar" />
          <div class="row-text">
            <p class="card-title">Tarjeta 9 — Espaciada</p>
            <small class="muted">justify-content: space-between</small>
          </div>
          <button class="btn">Ir</button>
        </article>
      </section>
    </main>

    <aside class="aside">Aside — Enlaces o info extra</aside>
    <footer class="site-footer">© 2025 — Ejercicio Grid & Flex</footer>
  </div>
</body>
</html>
```

Ahora sí empezamos!!!!!

¿Te acuerdas que era lo primero?? Ahora que estamos trabajando con el espacio el anular los estilos predefinidos es fundamental. Anulamos los estilos acuérdate selector universal.

```
/* ============================================================
   RESET BÁSICO Y TIPOGRAFÍA GLOBAL
   ============================================================ */

/* El * (selector universal) afecta a todos los elementos.
   box-sizing: border-box → hace que padding y borde se incluyan
   dentro del ancho/alto declarado, evitando cálculos confusos. */
* { box-sizing: border-box; }

/* Estilo base del body (toda la página):
   - margin: 0 → quita márgenes que los navegadores ponen por defecto.
   - font-family → tipografía moderna y legible, con varias alternativas.
   - color → texto principal en gris muy oscuro (casi negro). */
body {
  margin: 0;
  font-family: Roboto, Arial, sans-serif;
  color: #1f1f1f;
}
```

Volvemos al html que es lo que te llama la atención?? ¿donde está metido el body?

así que ahora en nuestro css tendremos que definir nuestro grid
esta rejilla es la planilla para toda la página y la definimos así

```
/* ============================================================
   GRID GLOBAL DE LA PÁGINA
   ============================================================ */

.grid-container {
  display: grid; /* Activa el modelo Grid en este contenedor */
```

```
  /* grid-template-areas → define el "mapa" de la página.
     Cada fila entre comillas representa una fila del grid.
     Los nombres (header, banner, main, aside, footer) son áreas
     que luego asignaremos a los elementos. */
  grid-template-areas:
    "header header header" /* Fila 1: header ocupa las 3 columnas */
    "banner main aside"    /* Fila 2: 3 columnas distintas */
    "footer footer footer";/* Fila 3: footer ocupa todo el ancho */

  /* grid-template-columns → ancho de las columnas:
     - 150px para el banner (izquierda)
     - 1fr (fracción) para el contenido central
     - 220px para el aside (derecha) */
  grid-template-columns: 150px 1fr 220px;

  /* grid-template-rows → alto de las filas:
     - auto: altura según contenido (header)
     - 1fr: ocupa el espacio sobrante (cuerpo principal)
     - auto: altura según contenido (footer) */
  grid-template-rows: auto 1fr auto;

  /* min-height: 100vh → asegura que la página ocupe
     al menos el 100% de la altura de la ventana. */
  min-height: 100vh;
}

/* Asignamos cada área a su zona correspondiente.
   Estos nombres deben coincidir con grid-template-areas. */
.site-header { grid-area: header; }  /* ✅ header debe ser hijo directo de
.grid */
```

```
.banner      { grid-area: banner; }
main         { grid-area: main; }
.aside       { grid-area: aside; }
.site-footer { grid-area: footer; }
```

Atención estos nombres que ponemos en las áreas en el mapa de áreas no son casualidad: sólo los hijos directos de nuestro grid container pueden ser mapeados

Es importante que te fijes en una referencia de tamaño ¿cuál es?

Exacto la medida fr.

La unidad fr representa una parte del espacio disponible en el contenedor de cuadrícula. El navegador suma todas las fracciones que has declarado y luego reparte el espacio restante de forma proporcional a cada una de ellas.

Imagina que tu contenedor tiene 900px de ancho y has declarado:

grid-template-columns: 1fr 2fr 1fr;

1. Suma de fracciones: El navegador suma las fracciones declaradas: 1+2+1=4 fracciones en total.
2. Valor de una fracción: El espacio disponible se divide entre el total de fracciones. En este caso, 900px/4=225px por cada 1fr.
3. Tamaño de las columnas:
    - La primera columna (1fr) tendrá un ancho de 225px.
    - La segunda columna (2fr) tendrá un ancho de 450px (2×225px).
    - La tercera columna (1fr) tendrá un ancho de 225px.

Lo genial de esto es que el cálculo se hace automáticamente. Si el contenedor cambia de tamaño, las columnas se adaptan de forma fluida, manteniendo siempre la misma proporción.

Tenemos otra medida también que quizás no conozcas: em.
m = unidad relativa al tamaño de letra del elemento.

Si el font-size actual es 16px, 1em = 16px; 1.5em = 24px. Muy útil para padding/márgenes que escalen con el texto.

Y ya luego los px que se han usado para facilitar pero em es una mejor práctica porque se adaptan siempre a la letra raíz.

Bueno que tenemos ahora?? Ya nuestro documento sabe que queremos tres áreas. Pero todavía queda un poco para que quede todo ordenado. Nos vamos al html. Al final el css empieza en el html, y normalmente cuando algo del estilo no te funciona el error está más en el html que en el css.

Header, menu, aside y banner ya sabes cómo se hacen.

Pasamos a las tarjetas.
¿Qué es un grid o un flex?

La sección class article es una rejilla: una estructura. Podríamos hacerlo con tres flex pero ¿qué perdemos? ¿qué es lo que no gestiona flex?

¿Quieres probarlo?

Lo que no gestiona flex es la bidireccionalidad con lo que luego tendremos que estar inventando para cuadrar las cajas horizontalmente o verticalmente.

¿Qué se pierde exactamente?

Al pasar de un solo Grid a tres Flexbox, pierdes las siguientes ventajas clave:

- Control de la estructura global: Con Grid, el contenedor padre (.articulos en tu ejemplo) define el layout 3x3 de todas las tarjetas. Al usar Flexbox, cada contenedor solo controla una fila. Tienes tres contenedores distintos, cada uno con su propio conjunto de reglas de Flexbox, lo que complica la gestión de un layout unificado.
- Posicionamiento y alineación en 2D: Grid te permite alinear los ítems en el eje horizontal y vertical al mismo tiempo (justify-items y align-items). Flexbox solo alinea a lo largo de su eje principal (con justify-content) o su eje cruzado (con align-items). Esto significa que si quisieras centrar la Tarjeta 1 en su celda con Flexbox, tendrías que aplicar estilos al ítem mismo, en lugar de al contenedor padre, lo que rompe el patrón de control centralizado.
- Gestión de las filas y columnas: La rejilla es consciente de sus filas y columnas. Por ejemplo, en tu código de main, el .articulos sabe que tiene 3 filas y 3 columnas. Con Flexbox, un contenedor solo sabe que tiene 3 elementos. Si añades un décimo elemento, tendrías que crear un cuarto contenedor Flexbox para él, o el diseño se desbordaría.
- Control del "salto de línea": Si usas flex-wrap: wrap; para que los elementos salten a la siguiente línea, Flexbox no tiene un conocimiento de las columnas. Simplemente los apila en la siguiente línea cuando no caben. Grid, en cambio, los coloca en celdas y filas específicas, manteniendo la estructura rígida que has definido.

- Capacidad de extensión: Con Grid, puedes hacer que una tarjeta se extienda para ocupar varias columnas o filas de forma sencilla (grid-column: span 2;). Esta capacidad no existe en un layout Flexbox simple. .

En resumen, mientras que Flexbox es la herramienta perfecta para organizar el contenido de un componente en una sola dirección, la rejilla es la única herramienta adecuada para diseñar un layout complejo y bidimensional. Cambiarlo por Flexbox te haría perder la simplicidad, la eficiencia y el control de la estructura que has construido.

Y con esto llegamos al quid de la cuestión. El grid que se complementa con flex. La rejilla que tiene nueve elementos flex box dentro. Las cajas las vamos a ordenar dentro de una rejilla pero lo que hay dentro de la caja, la imagen, el texto, la fecha, lo que hemos metido en la caja lo queremos ordenar con flex box.

Volvemos a la casa. Imagina que estás diseñando una casa.

CSS Grid es como el arquitecto que planea toda la estructura del edificio: dónde irán las paredes, cuántos pisos tendrá, y el tamaño de cada habitación. Es ideal para crear el diseño general de una página web, como la distribución de la cabecera, el menú, el contenido principal y el pie de página. Con Grid, controlas tanto las filas como las columnas al mismo tiempo. .

CSS Flexbox es como el diseñador de interiores que se encarga de organizar los muebles y objetos dentro de una sola habitación. Su objetivo es alinear y distribuir elementos a lo largo de una única dirección (en fila o en columna). Es perfecto para organizar componentes pequeños y específicos, como los elementos dentro de una tarjeta o los enlaces de un menú. .

En el código que analizamos, las dos herramientas trabajan juntas de forma muy eficiente:

1. Grid crea el esqueleto principal de la página. El contenedor <section class="rejilla"> usa Grid para organizar las nueve tarjetas en un patrón de 3x3.
2. Flexbox se encarga del diseño interno de cada tarjeta. Cada <article class="card"> usa Flexbox para alinear su propio contenido (el título, la imagen o el botón) de forma flexible.

En resumen:

- Grid para la macro-organización de tu página.
- Flexbox para la micro-organización dentro de tus componentes.

Esta combinación es la manera más moderna, flexible y escalable de construir layouts complejos en la web.

aqui tienes la declaración del contenedor rejilla

```
.rejilla {
  display: grid; /* Activamos Grid */
  grid-template-columns: repeat(3, 1fr); /* 3 columnas iguales */
  grid-template-rows: 100px 200px 100px; /* Fila 1 y 3 bajas, fila 2 más alta
*/
  gap: 12px;      /* Espacio entre tarjetas */
  padding: 16px;  /* Margen interno alrededor */
}
```

¿qué ahi que sea obligatorio definir? display: grid es decir decirle que es un grid; lo demás todo voluntario. Si no le dices las filas y las columnas por ejemplo serán 1 si no le dices el gap se juntaran ocuparan todo el espacio disponible igual que el padding.

Imprescindible para tener un grid: display: grid en el contenedor.

Casi siempre definirás también:
Alguna plantilla de pistas: grid-template-columns y/o grid-template-rows.
Normalmente necesario (para tener un layout útil) Definir pistas (tracks) de columnas o filas: grid-template-columns: ... o grid-template-rows: ...
Ej.: grid-template-columns: repeat(3, 1fr);
Si no defines nada, el grid existe, pero por defecto tendrás 1 columna y las filas se irán creando “implícitamente” (cada ítem cae en una fila nueva). Eso rara vez es lo que quieres para un layout.

Opcional pero frecuente: gap y alineaciones (justify/align/ place-*). También las áreas como en la rejilla general la que ocupa todo el body.

En los ítems (hijos):. Nada obligatorio. Se colocan solos por defecto.

Al crear las filas y las columnas vuelve a salir el concepto de fr (fracción) ya dijimos loq ue significaba el espacio completo lo dividire en tantas columnas como haga y le asignare un 1 es decir tendré tres hueco y le digo que cada una ocupe uno

Cuando usas `repeat(3, 1fr)`, le estás diciendo a la cuadrícula:

1. **"Quiero 3 columnas"** (`repeat(3, ...)`)
2. **"Y quiero que cada una ocupe una fracción del espacio disponible"** (`1fr`).

El navegador toma el espacio que le queda al contenedor y lo divide entre el número total de fracciones que has declarado.

Ejemplo Práctico:

Imagina que tu contenedor tiene 900 píxeles de ancho.

- `grid-template-columns: 1fr 1fr 1fr;` (que es lo mismo que `repeat(3, 1fr)`)

El navegador suma las fracciones (1+1+1=3). Luego, divide el ancho total entre ese número: 900px/3=300px. Por lo tanto, cada `1fr` equivale a 300px.

El resultado es un layout con tres columnas que se ajustan al 33.3% del ancho del contenedor, sin importar si este es de 900px, 1200px o 500px.

.

Esto hace que `fr` sea increíblemente útil para el diseño responsivo, ya que tus columnas se escalarán de forma proporcional a medida que la pantalla cambie de tamaño.

¿ Y si queremos que la columna central sea mas grande por ejemplo? Le decimos cuanto mas grande por ejemplo 3fr que sus compañeras que será 1fr y hay dos compañeras pues se dividirá el espacio total entre 5 y a la de en medio se le asignara 3.

/* grid-template-columns: repeat(3, 1fr); → Todas del mismo tamaño */

grid-template-columns: 1fr 2fr 1fr;

¿ qué pasa cuando pegamos rejilla?

pues ahora ya vamos a entrar a definir las tarjetas
fijate que tienen una clase padre y luego clases hijos. Está declarada en la misma clase card card-meta por ejemplo. Eso significa que hay un estilo general que puedes usar para todas las tarjetas y luego uno especial para cada una de ellas.

Lo primero es convertirlos en contenedores flex.

`display: flex` en el contenedor.

Normalmente útil (tamaño y crecimiento)

- `flex-grow` (0 por defecto): cuánto **crece** un ítem si sobra espacio.
- `flex-shrink` (1 por defecto): cuánto **encoge** un ítem si falta espacio.
- `flex-basis` (`auto` por defecto): **tamaño base** preferido del ítem (antes de crecer/encoger).

  Muy práctico: `flex-basis` sustituye a `width`/`height` **en el eje principal**.

- **Shorthand `flex`:**
  `flex: <grow> <shrink> <basis>`
  Ejemplos típicos:
  - `flex: 1;` → `1 1 0` (reparte espacio a partes iguales).
  - `flex: 0 0 auto;` → tamaño según contenido, ni crece ni encoge.
  - `flex: 0 0 200px;` → ítem “de 200px” (en el eje principal), fijo si no puede encoger.

Ajustes individuales

- `align-self` → alineación del **ítem** en el eje cruzado (sobrescribe `align-items` del contenedor).

```
   TARJETA BASE (estilos comunes) clase base que se reutiliza con una clase
concreata
   ============================================================ */


.card {
  border: 1px solid #ddd;  /* Borde gris suave */
  background: #fafafa;     /* Fondo gris muy claro */
```

```
  border-radius: 8px;      /* Esquinas redondeadas */
  padding: 10px;           /* Espaciado interno */
}

/* ============================================================
   FILA 1: TEXTO + FECHA (Flex en columna)
   ============================================================ */

.card--meta {
  display: flex;
  flex-direction: column; /* Eje principal vertical */
  justify-content: center;/* Centra verticalmente dentro de su celda */
  align-items: flex-center;/* Texto alineado a la izquierda */
  gap: 6px;               /* Espacio entre título y fecha */
}

/* Estilo del título dentro de la tarjeta */
.card-title { margin: 0; font-weight: 600; }

/* Estilo de la fecha: más pequeña y gris suave */
.card-date  { font-size: 0.9rem; color: #6b6b6b; }

/* ============================================================
   FILA 2: IMAGEN CENTRADA + TEXTO (Flex en columna)
   ============================================================ */

.card--media {
  display: flex;
  flex-direction: column; /* Apila imagen + caption */
  align-items: center;    /* Centra horizontal */
  justify-content: center;/* Centra vertical */
```

```
  gap: 8px;                /* Espacio entre imagen y texto */
  background: #f3f3f3;     /* Fondo ligeramente más oscuro */
}

/* Imagen dentro de la tarjeta media */
.card--media img {
  width: 120px;
  height: 120px;
  max-width: 100%;         /* Evita que se desborde en pantallas pequeñas */
  border-radius: 10px;     /* Bordes redondeados */
  object-fit: cover;       /* Rellena el recuadro recortando sin deformar */
}

/* Texto debajo de la imagen, centrado */
.card-caption { margin: 0; text-align: center; }

/* ============================================================
   FILA 3: AVATAR + TEXTO + BOTÓN (Flex en fila)
   ============================================================ */

.card--row {
  display: flex;           /* Elementos en fila */
  align-items: center;     /* Centrado vertical */
  gap: 10px;               /* Separación entre elementos */
  padding: 12px;
  background: #fff;        /* Fondo blanco */
  border: 1px solid #ccc;  /* Borde más fuerte */
}

/* Avatar circular dentro de la tarjeta */
.card--row .avatar {
```

```
  width: 48px;
  height: 48px;
  border-radius: 50%;   /* Lo hace circular */
  object-fit: cover;    /* Ajusta la imagen sin deformar */
}
```