

Tema 4 - Hojas de estilos

Sitio: [Aula Virtual do IES de Teis](#)

Curso: Deseño de interfaces web (DAW-dual 2024-2025)

Libro: Tema 4 - Hojas de estilos

Impreso por: Brais Bea Mascato

Data: mércores, 26 de marzo de 2025, 4:28 PM

Táboa de contidos

1. Introducción

2. Selectores

- 2.1. Selectores basados en etiquetas
- 2.2. Selectores avanzados
- 2.3. Selectores basados en clases
- 2.4. Selectores basados en identificadores

3. Añadir estilos a un documento con CSS

- 3.1. Hojas de estilo orientadas a medios de presentación
- 3.2. Paginación
- 3.3. Tipografías CSS3

4. Flexbox

- 4.1. Propiedades del contenedor flexible
- 4.2. Propiedades de los ítems flexibles

5. Modelo de Cajas (Box Model)

- 5.1. Unidades de Medida
- 5.2. Atributos de posición
- 5.3. Atributos de Margen
- 5.4. Atributos de Padding
- 5.5. Atributos de Borde

6. Atributos

- 6.1. Colores
- 6.2. Colores de fondo
- 6.3. Fuentes
- 6.4. Enlaces
- 6.5. Listas
- 6.6. Tablas
- 6.7. Visibilidad

7. Grid Layout

- 7.1. Propiedades del contenedor Grid

1. Introducción

Entre los avances más significativos en el campo del diseño web durante las últimas décadas, la aparición de las **Hojas de Estilo en Cascada (CSS, Cascading Style Sheets)** destaca como uno de los más relevantes. Su principal ventaja es permitir la **separación entre diseño (apariencia) y contenido (información)** en el desarrollo de sitios web. Esta separación facilita tanto el desarrollo como el mantenimiento de sitios web, haciéndolos más eficientes y flexibles.

Desde un punto de vista técnico, la filosofía de las CSS puede resumirse de la siguiente manera:

1. Se utiliza la etiqueta `<body>` de HTML para definir la estructura del contenido que se mostrará en el sitio (encabezados, párrafos, listas, etc.).
2. En un archivo externo o en la sección `<head>` del documento HTML, se define la apariencia del sitio utilizando CSS.

Esta arquitectura permite modificar el contenido dentro de `<body>` sin afectar el diseño definido en el archivo CSS, y viceversa: cambiar la apariencia en el CSS sin alterar los contenidos estructurados en el HTML.

Otra gran ventaja actual de CSS es su capacidad para **adaptar los sitios web a los dispositivos** desde los que se acceden. Gracias a herramientas como **Media Queries**, un sitio web puede ofrecer una experiencia de visualización optimizada para diferentes dispositivos, como tablets, smartphones, ordenadores personales, entre otros. Sin embargo, cabe aclarar que el diseñador debe crear explícitamente las hojas de estilo necesarias para cada dispositivo; no es un proceso automático o "mágico".

La organización **W3C** (World Wide Web Consortium) fue la encargada de estandarizar la primera versión de CSS, conocida como **CSS1**. El término "hojas de estilo en cascada" hace referencia a la capacidad de aplicar varios archivos CSS para controlar la apariencia de un mismo contenido. Esto introduce el concepto de **prioridad o preferencia**, que es clave para resolver conflictos cuando varios estilos afectan al mismo elemento.

Por ejemplo, un encabezado `<h1>` puede estar definido en un archivo CSS externo para que el texto sea azul, pero también puede tener una regla dentro de la propia página HTML que lo haga aparecer en rojo. En estos casos, el navegador, siguiendo las pautas de la especificación CSS, decide qué estilo aplicar según el principio de **cascada**.

CSS1 alcanzó el estatus de recomendación por la **W3C** en el año **1996**. Las reglas de CSS1 cuentan con un soporte adecuado en prácticamente todos los navegadores modernos más conocidos. Posteriormente, en **1998**, las reglas de **CSS2** también alcanzaron el estatus de recomendación, marcando un avance significativo en la evolución del diseño web.

En **junio de 2011**, el módulo de color de **CSS3 (CSS3 Color)** fue publicado como recomendación. Este módulo, junto con otros avances de CSS3, introdujo características más avanzadas y flexibles para el diseño de sitios web, como la gestión mejorada de colores y transparencias.

El objetivo principal de que el **W3C** estandarice tecnologías como CSS, HTML u otros lenguajes de Internet es **garantizar la interoperabilidad**. Esto significa que los creadores de sitios web no necesitan desarrollar versiones específicas para cada navegador existente (como Internet Explorer, Firefox, Chrome u Opera). Asimismo, busca evitar que los usuarios vean sitios web con apariencias diferentes según el navegador que utilicen, asegurando una experiencia de usuario uniforme y consistente.

Actualmente, **CSS se desarrolla de manera modular**, lo que permite la actualización y mejora de sus componentes de forma independiente. Este enfoque modular ha llevado a que las especificaciones de CSS ya no se versionen de manera monolítica (como CSS1, CSS2 o CSS3), sino que cada módulo evoluciona a su propio ritmo y alcanza diferentes niveles de madurez.

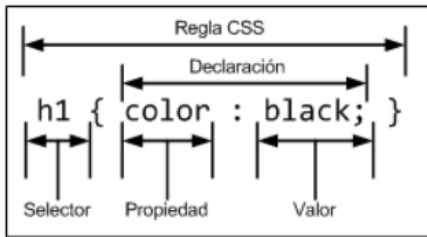
Por ejemplo, módulos como **Flexbox** y **Grid Layout**, fundamentales para el diseño de layouts modernos, han sido desarrollados y actualizados independientemente, permitiendo una implementación más ágil de nuevas funcionalidades en los navegadores.

El **W3C** publica periódicamente "instantáneas" (*snapshots*) que recopilan el estado estable de los diferentes módulos de CSS en un momento dado. Estas instantáneas reflejan las mejores prácticas actuales y las recomendaciones para los desarrolladores web.

En resumen, CSS no solo mejora la eficiencia en el diseño y el mantenimiento de sitios web, sino que también permite adaptarlos a las necesidades de los usuarios y dispositivos actuales, manteniendo un enfoque estandarizado gracias a la labor del W3C.

2. Selectores

Una hoja de estilo CSS está formada por **reglas** que definen cómo se visualizará una página web (reglas de estilo). Cada regla está compuesta por **selectores**, que determinan a qué elemento o parte de una página se aplica un estilo específico. Cada regla está compuesta de una parte de "selectores", un símbolo de "llave de apertura" ({), otra parte denominada "declaración" y por último, un símbolo de "llave de cierre" (}).



La **declaración** especifica los estilos que se aplican a los elementos y está compuesta por una o más **propiedades** CSS. Estas propiedades hacen referencia a las características que se modifican en el elemento seleccionado, como por ejemplo su tamaño de letra, su color de fondo, etc. Finalmente se establece el nuevo **valor** de la característica modificada en el elemento.

Sintaxis CSS

```
selector #id .class :pseudoclass ::pseudoelement [attr] {
  property : value ;
}
```

2.1. Selectores basados en etiquetas

Para comenzar a definir reglas de estilo, la manera más sencilla es utilizar las propias **etiquetas HTML** como selectores. Esto implica asociar a cada etiqueta una declaración del estilo que se le aplica.

Las declaraciones tienen la siguiente estructura:

```
selector {
```

```
    propiedad: valor;
```

```
}
```

- La **propiedad** hace referencia a la característica que se desea modificar, como `color`.
- El **valor** especifica la instancia o configuración de esa propiedad, como `blue`.

Por ejemplo, para aplicar un estilo a la etiqueta `<h1>`, se usaría el siguiente selector:

```
h1 {
```

```
    color: blue;
```

```
}
```

En CSS, los selectores se escriben sin las llaves angulares `< >`. Es decir, simplemente se utilizan como `h1`, `h2`, etc. Además, la declaración (propiedad: valor) debe estar contenida entre llaves `{ }`. Cualquier etiqueta HTML puede tener un estilo asignado, pero es fundamental que la descripción de las reglas siga la **sintaxis definida por la especificación CSS**. Si un navegador encuentra un selector con una sintaxis incorrecta, ignorará toda la declaración y continuará con la siguiente regla, sin importar si el error afecta solo a una propiedad, a un valor o a toda la regla.

Selectores múltiples y múltiples propiedades

CSS permite definir estilos para varios selectores al mismo tiempo, así como aplicar múltiples propiedades a un único selector.

Sintaxis para varios selectores:

```
selector1, selector2 {
```

```
    propiedad1: valor1;
```

```
    propiedad2: valor2;
```

```
}
```

Por ejemplo:

```
h1, h2 {
```

```
    color: blue;
```

```
}
```

Esto aplicará el color azul tanto a las etiquetas `<h1>` como a `<h2>`.

Sintaxis para múltiples propiedades:

```
h1 {
```

```
    color: blue;
```

```
    background-color: red;
```

```
}
```

Aquí, la etiqueta `<h1>` tendrá texto azul (`color: blue`) y un color de fondo rojo (`background-color: red`). La propiedad `background-color` define el color de fondo del texto, que por defecto coincide con el color de fondo de la página.

¿Dónde se colocan las reglas CSS?

Una vez conocida la sintaxis de las reglas, surge la pregunta: **¿Dónde deben colocarse estas declaraciones para que el navegador las lea e interprete?** Una respuesta válida (aunque existen más) es incluirlas en la cabecera del archivo HTML, dentro de la sección `<head>`, utilizando las etiquetas `<style>`.

Por ejemplo:

```
<head>

<title>Ejemplo</title>

<style>

  h1 {

    color: blue;

    background-color: red;

  }

</style>

</head>
```

Cuando el navegador lee la cabecera del archivo HTML, interpreta que todas las etiquetas `<h1>` incluidas en el `<body>` tendrán texto de color azul y un fondo de color rojo.

2.2. Selectores avanzados

Los **selectores avanzados** en CSS permiten seleccionar elementos HTML de formas más específicas y flexibles. A continuación, se muestran los selectores avanzados más importantes:

1. Selectores de atributos

Permiten seleccionar elementos basándose en los atributos y sus valores.

[atributo]: Selecciona elementos que tienen un atributo específico.

```
input[type] {  
    border: 1px solid black;  
}
```

Selecciona todos los elementos `<input>` que tienen cualquier atributo `type`.

[atributo="valor"]: Selecciona elementos con un atributo que tiene exactamente ese valor.

```
input[type="text"] {  
    background-color: lightyellow;  
}
```

Selecciona `<input>` con `type="text"`.

[atributo^="valor"]: Selecciona elementos cuyo atributo empieza con un valor.

```
a[href^="https://"] {  
    color: blue;  
}
```

Selecciona todos los enlaces (`<a>`) que comienzan con `"https://"`.

[atributo\$="valor"]: Selecciona elementos cuyo atributo termina con un valor.

```
a[href$=".pdf"] {  
    color: red;  
}
```

Selecciona todos los enlaces que terminan con `".pdf"`.

[atributo*="valor"]: Selecciona elementos cuyo atributo contiene un valor específico.

```
a[href*="example"] {  
    text-decoration: underline;  
}
```

Selecciona todos los enlaces que contienen `"example"` en su atributo `href`.

2. Selectores de hijos y descendientes

Permiten seleccionar elementos basándose en su relación jerárquica.

A B: Selecciona todos los descendientes **B** de **A**.

```
div p {  
    color: blue;  
}
```

Selecciona todos los **<p>** dentro de cualquier **<div>**.

A > B: Selecciona los hijos directos **B** de **A**.

```
ul > li {  
    list-style-type: square;  
}
```

Selecciona solo los **** que son hijos directos de ****.

A + B: Selecciona el elemento **B** inmediatamente después de **A**.

```
h1 + p {  
    font-style: italic;  
}
```

Selecciona el primer **<p>** inmediatamente después de un **<h1>**.

A ~ B: Selecciona todos los elementos **B** que son hermanos de **A**.

```
h1 ~ p {  
    color: gray;  
}
```

Selecciona todos los **<p>** que son hermanos de un **<h1>**.

3. Selectores de pseudo-clases

Se usan para aplicar estilos basados en el estado de los elementos.

:hover: Selecciona un elemento cuando el puntero del mouse pasa sobre él.

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

:focus: Selecciona un elemento cuando está enfocado (por ejemplo, al hacer clic en un campo de formulario).

```
input:focus {  
    border-color: blue;  
}
```


:nth-child(n): Selecciona el **n**-ésimo hijo de su padre.

```
li:nth-child(2) {  
  color: red;  
}
```

Selecciona el segundo elemento **** de su lista.

:nth-of-type(n): Selecciona el **n**-ésimo hijo del mismo tipo.

```
p:nth-of-type(1) {  
  font-weight: bold;  
}
```

Selecciona el primer **<p>** dentro de un contenedor, ignorando otros tipos de elementos.

:not(selector): Selecciona todos los elementos que **no** coincidan con el selector.

```
p:not(.special) {  
  color: gray;  
}
```

Selecciona todos los párrafos que no tengan la clase **special**.

4. Selectores de pseudo-elementos

Permiten aplicar estilos a partes específicas de un elemento.

::before: Inserta contenido antes del contenido de un elemento.

```
h1::before {  
  content: "👉";  
}
```

Agrega un emoji antes de cada **<h1>**.

::after: Inserta contenido después del contenido de un elemento.

```
h1::after {  
  content: "🌟";  
}
```

::first-line: Aplica estilos solo a la primera línea de texto de un elemento.

```
p::first-line {  
  font-weight: bold;  
}
```

::first-letter: Aplica estilos a la primera letra de un elemento.

```
p::first-letter {  
  
    font-size: 2em;  
  
    color: red;  
  
}
```

5. Selectores combinados

Se pueden combinar varios selectores para lograr un control detallado.

- **div[class="highlight"]**: Selecciona todos los `<div>` que tengan una clase específica.
- **ul > li:first-child**: Selecciona el primer `` hijo de un ``.
- **a[href^="https"]:hover**: Selecciona enlaces seguros que están siendo "hovered".

2.3. Selectores basados en clases

Los **selectores basados en etiquetas** tienen un uso muy claro y son fáciles de entender. Sin embargo, desde el punto de vista de la flexibilidad y la reutilización, esta alternativa presenta limitaciones.

Un ejemplo de esta falta de flexibilidad es el siguiente: si se desea establecer diferentes estilos según el tipo de párrafo, de manera que se distinga el encabezado de la página del resto del texto, los selectores basados en etiquetas no son suficientes. Por ejemplo, al declarar:

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

Todos los textos dentro de las etiquetas `<p>` `</p>` se mostrarán en color azul, sin distinción. Pero si se desea que algunos párrafos tengan un estilo y otros tengan otro diferente, esta alternativa no es válida.

Para resolver esto, es necesario usar **selectores basados en clases**.

Ventajas de las clases

Mediante las clases se pueden definir estilos abstractos, es decir, que no estén asociados directamente a una etiqueta HTML. Las clases permiten:

- Aplicar estilos específicos a diferentes elementos de la página.
- Reutilizar estilos de manera más flexible.
- Aplicar múltiples estilos a un mismo elemento.

Existen dos tipos principales de clases:

1. Clases asociadas a etiquetas HTML específicas:

Estas se definen con la sintaxis Etiqueta.NombreDeClase, por ejemplo:

```
h1.roja {  
    color: red;  
}
```

Para indicar en cada etiqueta `<h1>` qué estilo se le quiere aplicar se usa el atributo `class` de la siguiente manera:

```
<h1 class="roja"> Un encabezamiento rojo </h1>
```

2. Clases genéricas:

Estas no están asociadas a ninguna etiqueta en particular y pueden aplicarse a cualquier elemento. Por ejemplo:

```
.citas {  
    color: gray;  
}
```

Una clase así definida puede aplicarse a cualquier elemento de la página. Del ejemplo siguiente, la primera declaración se aplica a todo el párrafo, la segunda a todo el encabezado `<h1>` y la tercera a todo el bloque `<div>`:

```
<p class="verde">Un párrafo verde</p>  
<h1 class="verde">Un encabezado verde</h1>  
<div class="verde">Un contenedor verde</div>
```

Esta técnica es fundamental para el diseño moderno de páginas web, donde cada elemento puede requerir un estilo único o una combinación de estilos.

Clases combinadas

Una característica muy interesante del uso de clases es que se puede asignar más de una clase a una misma etiqueta, teniendo en cuenta la prioridad en su aplicación.

<h3 class="textorojo fondoazul">titulo en rojo, fondo azul</h3>

Cómo se interpretan las clases combinadas

- Prioridad entre clases:

Cuando se aplican múltiples clases a un mismo elemento, los estilos definidos más tarde en el archivo CSS sobrescriben los anteriores, si afectan las mismas propiedades.

- Especificidad de los selectores:

Los selectores más específicos tienen prioridad sobre los selectores más generales.

- Orden en la declaración:

Si dos clases tienen la misma especificidad y afectan la misma propiedad, el orden en que se aplican las clases en el archivo CSS será determinante.

2.4. Selectores basados en identificadores

En la sección anterior, vimos que las clases permiten definir propiedades que pueden ser compartidas por uno o varios elementos. Es decir, las clases son reutilizables. Ahora veremos los selectores basados en identificadores, que tienen una función y sintaxis muy parecidas a las clases, pero con una diferencia importante: **los identificadores solo deben usarse en un único elemento**.

Un selector basado en identificador se define en las etiquetas `<style>` usando una almohadilla `#` antes del nombre del identificador. La sintaxis es la siguiente:

```
#nombreIdentificador {  
  
    propiedad: valor;  
  
    propiedad: valor;  
  
}
```

Esto es muy similar a la sintaxis de las clases, pero con una almohadilla `#` en lugar de un punto `.`. Para aplicar el estilo a un elemento, se utiliza el atributo `id`:

```
<head>  
  
    <style>  
  
        #rojo {  
  
            color: red;  
  
        }  
  
    </style>  
  
</head>  
  
<body>  
  
    <p id="rojo">Este párrafo será rojo</p>  
  
</body>
```

Diferencia entre clases e identificadores

Aunque a primera vista parecen similares, hay diferencias clave entre clases e identificadores:

1. Uso en múltiples elementos:

- **Clases:** Se pueden aplicar a varios elementos.

```
<div class="textorojo">Texto rojo 1</div>
```

```
<div class="textorojo">Texto rojo 2</div>
```

- **Identificadores:** Se deben aplicar a un único elemento, ya que son únicos y exclusivos.

```
<div id="cabecera">Cabecera de la página</div>
```

2. Concepto de exclusividad:

Los identificadores representan "objetos únicos". Esto es útil para designar bloques principales o secciones específicas de una página, como cabecera, contenido y pie de página:

```
<div id="cabecera">Cabecera</div>
```

```
<div id="contenido">Contenido principal</div>
```

```
<div id="piepagina">Pie de página</div>
```

3. Buen uso de estándares:

Aunque los navegadores actuales permiten reutilizar un identificador en varios elementos, no es una buena práctica. Esto puede causar problemas de semántica y funcionalidad, como en los enlaces internos (<a>), donde un identificador debe señalar a un único elemento.

Ventajas de los identificadores

1. **Enlaces internos:** Los identificadores permiten crear enlaces internos en una página, facilitando la navegación.

```
<a href="#cabecera">Ir a la cabecera</a>
```

2. **Semántica clara:** Usar identificadores para elementos únicos ayuda a estructurar mejor el diseño y la funcionalidad del sitio.
3. **Compatibilidad con JavaScript:** Los identificadores son fundamentales para interactuar con elementos específicos mediante JavaScript.

3. Añadir estilos a un documento con CSS

Sin duda, no existe ninguna desventaja en utilizar **CSS** para la maquetación de páginas web. Todo son ventajas, entre las cuales destacan las siguientes:

1. **Mayor control en el diseño de las páginas:** CSS permite alcanzar diseños que están fuera del alcance de HTML por sí solo.
2. **Menor trabajo:** Cambiar el estilo de todo un sitio web es tan sencillo como modificar un único archivo CSS.
3. **Documentos más pequeños:** Las etiquetas `` y las numerosas tablas que antes se utilizaban para dar formato desaparecen, lo que reduce significativamente la cantidad de código necesario.
4. **Documentos más estructurados:** Los documentos bien estructurados son más accesibles para diferentes dispositivos y usuarios, mejorando la experiencia y la compatibilidad.
5. **El HTML de presentación está en vías de desaparición:** Los elementos y atributos de presentación definidos en las especificaciones HTML han sido declarados obsoletos por el **W3C**.
6. **Amplio soporte:** Actualmente, casi todos los navegadores modernos soportan prácticamente toda la especificación de **CSS1**, y la mayoría también admiten las recomendaciones de **CSS2** y **CSS2.1**.

El **funcionamiento** de las hojas de estilo en cascada puede resumirse en tres pasos principales:

1. **Crear un documento HTML:** Este documento debe tener una estructura lógica y un significado semántico, empleando los elementos HTML adecuados. En esta etapa, se crea la estructura básica de la página web.
2. **Escribir las reglas de estilo:** Estas reglas definen el aspecto deseado de cada elemento. Las reglas seleccionan un elemento por su nombre (etiqueta, clase o id) y luego especifican las propiedades (como fuente, color, etc.) y sus valores.
3. **Vincular los estilos al documento:** Las reglas de estilo pueden estar en un archivo independiente que se aplique a todo el sitio, o incrustarse en la cabecera del documento para aplicarse solo a esa página específica.

Las **hojas de estilo** consisten en una o más reglas que describen cómo deben mostrarse los elementos en pantalla.

Al aplicar las reglas de estilo a un documento HTML, existen tres modos principales de hacerlo:

1. **Estilos en línea:** Se escriben directamente dentro de la etiqueta HTML utilizando el atributo `style`.

```
<h1 style="color: blue; text-align: center;">Estilo en línea</h1>
```

2. **Hojas de estilo incrustadas:** Se colocan dentro de la cabecera del documento HTML utilizando la etiqueta `<style>`.

```
<title>Estilos incrustados</title>
<style>
  h1 {
    color: green;
    text-align: center;
  }
  p {
    font-size: 18px;
    background-color: lightyellow;
  }
</style>
```

3. **Hojas de estilo externas:** Son archivos CSS independientes que se vinculan o se importan al documento HTML mediante la etiqueta `<link>` o la regla `@import`.

Archivo CSS (estilos.css):

```
/* Archivo estilos.css */
h1 {
  color: red;
  text-align: center;
}
p {
  font-size: 20px;
  background-color: lightblue;
}
```

Documento HTML (index.html)

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Estilos externos</title>
  <!-- Vincular la hoja de estilo externa -->
  <link rel="stylesheet" href="estilos.css">
</head>
<body>
  <h1>Título con estilo externo</h1>
  <p>Párrafo con estilos definidos en un archivo externo.</p>
</body>
</html>
```

El uso de CSS no solo optimiza el diseño y el mantenimiento de las páginas web, sino que también mejora la accesibilidad, la compatibilidad entre navegadores y la eficiencia del desarrollo.

3.1. Hojas de estilo orientadas a medios de presentación

CSS2 introdujo la posibilidad de orientar las hojas de estilo hacia medios de presentación específicos. Para ello, se utiliza el atributo **media** dentro del elemento `<link>`. Esto permite aplicar diferentes estilos dependiendo del medio en el que se visualice la página (pantallas, impresoras, dispositivos móviles, etc.).



Valores del atributo media

La siguiente tabla muestra los valores que puede tomar el atributo media y los medios de presentación a los que se aplican:

Medio de presentación	Valor del atributo media	Descripción
Todos los medios	all	Aplica los estilos a todos los medios definidos.
Dispositivos Braille	braille	Para dispositivos táctiles que emplean el sistema Braille.
Impresoras Braille	embossed	Para impresoras que emplean el sistema Braille.
Dispositivos de mano	handheld	Para dispositivos portátiles como teléfonos móviles, PDA, etc.
Impresión	print	Para impresoras y navegadores en el modo "vista previa para imprimir".
Proyección	projection	Para proyectores y dispositivos usados en presentaciones.
Pantallas	screen	Para pantallas de ordenador y dispositivos con resolución estándar.
Navegadores de voz	speech	Para sintetizadores de voz utilizados por personas con discapacidades visuales.
Dispositivos textuales	tty	Para dispositivos limitados al texto, como teletipos y terminales de texto.
Televisores	tv	Para televisores y dispositivos con baja resolución.

Uso del atributo media

En el siguiente ejemplo, se muestra cómo orientar estilos a medios específicos utilizando el atributo media en el elemento `<link>`:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Ejemplo de media</title>
  <!-- Estilo para pantallas -->
  <link rel="stylesheet" href="estilo-pantalla.css" media="screen">
  <!-- Estilo para impresión -->
  <link rel="stylesheet" href="estilo-impresion.css" media="print">
</head>
<body>
  <h1>Ejemplo de uso de media</h1>
  <p>Contenido con estilo diferente en pantallas y al imprimir.</p>
</body>
</html>
```

Uso de la regla @media

También se pueden definir estilos específicos dentro de un archivo CSS utilizando la regla @media.

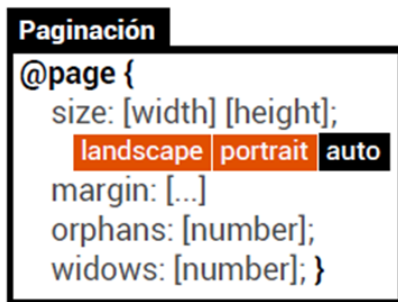
```
/* Estilos para pantallas */
@media screen {
  body {
    background-color: lightblue;
    color: black;
  }
}

/* Estilos para impresión */
@media print {
  body {
    background-color: white;
    color: black;
  }
}
```

Importante: Los estilos generales deben definirse antes de las reglas para impresión (@media print).

3.2. Paginación

La regla **@page** permite personalizar el diseño de las páginas cuando el contenido se presenta en un medio paginado, como una impresora. Puedes configurar aspectos como el tamaño de la página, los márgenes, y otras características relacionadas con la paginación.



Propiedades comunes en @page

size

- Define el tamaño y la orientación de la página.
- Valores posibles:
 - Específicos: **A4**, **letter**, **A3**, etc.
 - Personalizados: size: 210mm 297mm; (ancho y alto en mm o px).
 - Orientación:
 - **portrait** (vertical).
 - **landscape** (horizontal).
 - **auto** (predeterminado).

margin

- Establece los márgenes de la página.

orphans

- Define el número mínimo de líneas de un párrafo que deben aparecer en la parte inferior de una página antes de que el párrafo se divida.

widows

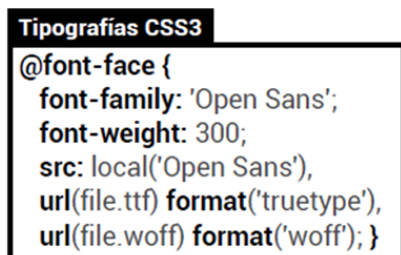
- Define el número mínimo de líneas de un párrafo que deben aparecer en la parte superior de una nueva página si el párrafo se divide.

```
@page {
  size: A4 portrait; /* Página tamaño A4 en orientación vertical */
  margin: 20mm;      /* Márgenes de 20mm en toda la página */
  orphans: 3;        /* Mínimo 3 líneas al final de la página */
  widows: 3;         /* Mínimo 3 líneas al comienzo de la página */
}

@page :first {
  margin: 30mm; /* Márgenes diferentes para la primera página */
}
```

3.3. Tipografías CSS3

La regla CSS **@font-face** permite incluir y usar tipografías personalizadas en un sitio web, incluso si no están instaladas en el dispositivo del usuario. Esta regla forma parte de **CSS3** y es muy útil para garantizar una experiencia de diseño uniforme en todos los navegadores y dispositivos.



La regla @font-face en CSS admite varios parámetros que permiten configurar fuentes personalizadas de manera flexible.

font-family (Obligatorio)

- Define el nombre de la fuente que usarás en tu CSS.
- Este nombre es como un alias que puedes usar en las propiedades font-family del resto de tu código.

src (Obligatorio)

- Define la ubicación de la fuente.
- Admite:
 - **local()**: Busca la fuente instalada en el dispositivo del usuario.
 - **url()**: Especifica la URL del archivo fuente que el navegador debe descargar.
 - **format()**: Indica el formato del archivo (por ejemplo, truetype, woff, woff2, opentype).

font-style

- Especifica el estilo de la fuente.
- Valores posibles:
 - **normal**: Fuente sin inclinación.
 - **italic**: Fuente inclinada (cursiva).
 - **oblique**: Fuente inclinada artificialmente.

font-weight

- Define el grosor de la fuente.
- Valores posibles:
 - Números de 100 (fino) a 900 (negrita).
 - Palabras clave: **normal** (400) o **bold** (700).

font-stretch

- Controla cómo se estrecha o ensancha la fuente.
- Valores posibles:
 - **normal**
 - **condensed** (estrecho)
 - **expanded** (ancho)
 - También admite valores como **ultra-condensed**, **extra-expanded**, etc.

unicode-range

- Especifica los caracteres admitidos por la fuente.
- Permite cargar parcialmente una fuente, optimizando su uso para idiomas o caracteres específicos.

ascent-override, descent-override, line-gap-override (Opcional)

- Permiten ajustar las métricas de la fuente.
- **ascent-override**: Ajusta la altura máxima de la fuente.
- **descent-override**: Ajusta la altura mínima de la fuente.
- **line-gap-override**: Ajusta el espacio entre líneas.

@font-face funciona de la siguiente manera.

1. **Descarga automática:** Si el navegador del usuario no encuentra la fuente localmente (local()), descarga el archivo especificado en la URL.
2. **Compatibilidad de formatos:** Los navegadores modernos suelen admitir formatos como **WOFF** y **WOFF2**, mientras que navegadores antiguos pueden requerir **TrueType (TTF)** o **OpenType (OTF)**.
3. **Prioridad:** El navegador usa el primer archivo compatible que encuentra en la lista src.

Ventajas de @font-face:

1. Permite usar tipografías personalizadas en cualquier sitio web.
2. Garantiza consistencia en el diseño, independientemente de las fuentes instaladas en el dispositivo del usuario.
3. Mejora la accesibilidad y personalización del diseño web.

Rendimiento:

- Usa formatos comprimidos como **WOFF** o **WOFF2** para optimizar el rendimiento y reducir el tamaño de los archivos.
- Define un **fallback** (fuente alternativa) en caso de que la fuente personalizada no esté disponible.

Verificación de fuente instalada

Para comprobar que si tenemos una **fuentes instalada** en el sistema podemos proceder como sigue:

1. Abre la Configuración (Windows + I).
2. Ve a Personalización > Fuentes.
3. Busca la fuente en la barra de búsqueda o explora la lista.

Fuentes variables y fuentes estáticas

Una **fuentes variable** es un tipo de fuente tipográfica que permite modificar varias características como peso, ancho, inclinación y más, usando un solo archivo de fuente. A diferencia de las fuentes estáticas, donde cada variante (negrita, itálica, condensada, etc.) requiere un archivo separado, las fuentes variables contienen múltiples estilos en un solo archivo, lo que mejora la flexibilidad y reduce el peso de la web.

No todos los navegadores son compatibles con **fuentes variables**, por lo que habría que utilizar **fuentes estáticas**. La compatibilidad la podemos comprobar en:

<https://caniuse.com/?search=variable%20font>

4. Flexbox

Flexbox (o Flexible Box Layout) es un modelo de diseño en CSS que facilita la creación de layouts flexibles y alineados. Es especialmente útil para distribuir elementos en filas o columnas y ajustar su tamaño y posición dinámicamente, incluso en pantallas de diferentes tamaños.

- **Contenedor flexible (flex container):**

Es el elemento al que aplicas la propiedad `display: flex`; Este contenedor agrupa los elementos secundarios, llamados **ítems flexibles**.

CSS:

```
.contenedor {  
  display: flex; /* Hace que los hijos sean ítems flexibles */  
}
```

html:

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

doc:

Elemento 1Elemento 2Elemento 3

- **Ítems flexibles (flex items):**

Son los hijos directos del contenedor flexible. Estos ítems heredan las propiedades de flex que se apliquen al contenedor.

4.1. Propiedades del contenedor flexible

Estas propiedades controlan la dirección, alineación y distribución de los ítems.

display: flex;

- Activa el modelo de diseño Flexbox en el contenedor.

flex-direction

- Define la dirección principal de los ítems (eje principal).
- Valores:
 - **row** (predeterminado): Los ítems se distribuyen en una fila (de izquierda a derecha).
 - **row-reverse**: Los ítems se distribuyen en una fila inversa (de derecha a izquierda).
 - **column**: Los ítems se distribuyen en una columna (de arriba hacia abajo).
 - **column-reverse**: Los ítems se distribuyen en una columna inversa (de abajo hacia arriba).

Ejemplo 1:

CSS

```
.contenedor {
  display: flex;
  flex-direction: row;
}
```

doc:

Elemento 1Elemento 2Elemento 3

Ejemplo 2:

CSS

```
.contenedor {
  display: flex;
  flex-direction: row-reverse;
}
```

doc:

Elemento 3Elemento 2Elemento 1

Ejemplo 3:

CSS

```
.contenedor {
  display: flex;
  flex-direction: column;
}
```

doc:

Elemento 1
Elemento 2
Elemento 3

Ejemplo 4:

CSS

```
.contenedor {  
  display: flex;  
  flex-direction: column-reverse;  
}
```

doc:

Elemento 3
Elemento 2
Elemento 1

justify-content

- Controla la alineación de los ítems en el eje principal (horizontal por defecto).
- Valores:
 - **flex-start** (predeterminado): Los ítems se alinean al inicio.
 - **flex-end**: Los ítems se alinean al final.
 - **center**: Los ítems se alinean al centro.
 - **space-between**: Espacio uniforme entre ítems, pero no al principio ni al final.
 - **space-around**: Espacio uniforme alrededor de cada ítem.

Ejemplo 5:

CSS

```
.contenedor {  
  display: flex;  
  justify-content: flex-start;  
}
```

doc:

Elemento 1Elemento 2Elemento 3

Ejemplo 6:

CSS

```

.contenedor {
  display: flex;
  justify-content: flex-end;
}

```

doc:

Elemento 1Elemento 2Elemento 3

Ejemplo 7:

CSS

```

.contenedor {
  display: flex;
  justify-content: center;
}

```

doc:

Elemento 1Elemento 2Elemento 3

Ejemplo 8:

CSS

```

.contenedor {
  display: flex;
  justify-content: space-between;
}

```

doc:

Elemento 1

Elemento 2

Elemento 3

Ejemplo 9:

CSS

```

.contenedor {
  display: flex;
  justify-content: space-around;
}

```

doc:

Elemento 1

Elemento 2

Elemento 3

align-items

- Controla la alineación de los ítems en el eje perpendicular (vertical por defecto).
- Valores:
 - **stretch** (predeterminado): Los ítems se estiran para llenar el contenedor.
 - **flex-start**: Los ítems se alinean al inicio.
 - **flex-end**: Los ítems se alinean al final.
 - **center**: Los ítems se alinean al centro.
 - **baseline**: Los ítems se alinean según su línea base de texto.

Ejemplo 10:

CSS

```

.contenedor {
  display: flex;
  flex-direction: column;
  align-items: stretch;
}
div {
  background-color: lightblue;
}
div div {
  background-color: blue;
}

```

doc:

Elemento 1
Elemento 2
Elemento 3

Ejemplo 11:

CSS

```

.contenedor {
  display: flex;
  flex-direction: column;
  align-items: flex-start;
}
div {
  background-color: lightblue;
}
div div {
  color: lightcyan;
  background-color: blue;
}

```

doc:

Elemento 1
Elemento 2
Elemento 3

Ejemplo 12:

CSS

```

.contenedor {
  display: flex;
  flex-direction: column;
  align-items: flex-end;
}
div {
  background-color: lightblue;
}
div div {
  color: lightcyan;
  background-color: blue;
}

```

doc:

Elemento 1
Elemento 2
Elemento 3

Ejemplo 13:

CSS

```

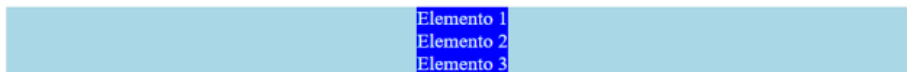
.contenedor {
  display: flex;
  flex-direction: column;
  align-items: center;
}

div {
  background-color: lightblue;
}

div div {
  color: lightcyan;
  background-color: blue;
}

```

doc:

Ejemplo 14:

CSS

```

.contenedor {
  display: flex;
  flex-direction: row;
  align-items: baseline;
}

div {
  background-color: lightblue;
}

div div {
  color: lightcyan;
  background-color: blue;
}

div:nth-child(1) {
  font-size: large;
}

div:nth-child(3) {
  font-size: small;
}

```

doc:

**flex-wrap**

- Define si los ítems deben ajustarse (hacer "wrap") cuando no caben en una sola fila o columna.
- Valores:
 - **nowrap** (predeterminado): Todos los ítems se mantienen en una sola línea.
 - **wrap**: Los ítems se ajustan a múltiples líneas.
 - **wrap-reverse**: Los ítems se ajustan a múltiples líneas en orden inverso.

Ejemplo 15:

CSS

```

.contenedor {
  display: flex;
  flex-wrap: nowrap;
}
div {
  background-color: lightblue;
}
div div {
  color: lightcyan;
  background-color: blue;
}
div:nth-child(1) {
  font-size: large;
}
div:nth-child(3) {
  font-size: small;
}

```

doc:

Elemento 1	Elemento 1	Elemento 2	Elemento 2	Elemento 3	Elemento 3	Elemento 4	Elemento 4
Elemento 1	Elemento 1	Elemento 2	Elemento 2	Elemento 3	Elemento 3	Elemento 4	Elemento 4
Elemento 1		Elemento 2		Elemento 3		Elemento 4	

Ejemplo 16:

CSS

```

.contenedor {
  display: flex;
  flex-wrap: wrap;
}
div {
  background-color: lightblue;
}
div div {
  color: lightcyan;
  background-color: blue;
}
div:nth-child(1) {
  font-size: large;
}
div:nth-child(3) {
  font-size: small;
}

```

doc:

Elemento 1	Elemento 1	Elemento 1	Elemento 1	Elemento 1					
Elemento 2	Elemento 2	Elemento 2	Elemento 2	Elemento 2	Elemento 3	Elemento 3	Elemento 3	Elemento 3	Elemento 3
Elemento 4	Elemento 4	Elemento 4	Elemento 4	Elemento 4					

Ejemplo 17:

CSS

```

.contenedor {
  display: flex;
  flex-wrap: wrap-reverse;
}
div {
  background-color: lightblue;
}
div div {
  color: lightcyan;
  background-color: blue;
}
div:nth-child(1) {
  font-size: large;
}
div:nth-child(3) {
  font-size: small;
}

```

doc:

Elemento 4 Elemento 4 Elemento 4 Elemento 4 Elemento 4
 Elemento 2 Elemento 2 Elemento 2 Elemento 2 Elemento 2 Elemento 3 Elemento 3 Elemento 3 Elemento 3 Elemento 3
 Elemento 1 Elemento 1 Elemento 1 Elemento 1 Elemento 1

4.2. Propiedades de los ítems flexibles

flex

- Es una propiedad abreviada para definir cómo crecen, encogen y ocupan espacio los ítems.
- Formato: **flex: [grow] [shrink] [basis];**
 - **grow**: Factor de crecimiento (cuánto espacio extra puede tomar el ítem).
 - **shrink**: Factor de encogimiento (cuánto espacio pierde el ítem si no hay suficiente espacio).
 - **basis**: Tamaño base del ítem antes de aplicar grow o shrink.

Ejemplo 18:

CSS

```
.contenedor {
  display: flex;
}
.item1 {
  background-color: lightcoral;
  flex: 1 1 150px; /* Puede crecer, encogerse, tamaño base
150px */
}
.item2 {
  background-color: lightblue;
  flex: 2 1 100px; /* Crece el doble que item1, encogimiento
igual, base 100px */
}
.item3 {
  background-color: lightgreen;
  flex: 3 0 200px; /* Crece el triple que item1, no se encoge,
base 200px */
}
```

html

```
<div class="contenedor">
  <div class="item1">Elemento 1</div>
  <div class="item2">Elemento 2</div>
  <div class="item3">Elemento 3</div>
</div>
```

doc:



order

- Cambia el orden en el que se muestran los ítems (sin alterar el HTML).
- Valor predeterminado: 0.

Ejemplo 19:

CSS

```

.contenedor {
  display: flex;
}
.item1 {
  order: 2;
}
.item2 {
  order: 3;
}
.item3 {
  order: 1;
}

```


html

```

<div class="contenedor">
  <div class="item1">Elemento 1</div>
  <div class="item2">Elemento 2</div>
  <div class="item3">Elemento 3</div>
</div>

```

doc:


align-self

- Permite alinear un ítem individualmente (anula align-items del contenedor).
- Valores:
 - **auto** (predeterminado): Usa el valor de align-items.
 - Otros valores: **flex-start**, **flex-end**, **center**, **baseline**, **stretch**.

Ejemplo 20:

CSS

```

.contenedor {
  display: flex;
  flex-direction: column;
  align-items: center;
}
.item {
  align-self: flex-end;
}

```

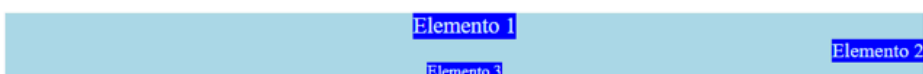
html

```

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

```

doc



5. Modelo de Cajas (Box Model)

El **modelo de cajas** es el sistema fundamental de CSS que define cómo se estructura y muestra cada elemento en una página web. Cada elemento se representa como una caja rectangular compuesta por cuatro áreas principales:

1. **Contenido (Content):**

- Es el área donde se coloca el texto, las imágenes u otros elementos.
- Puede tener dimensiones específicas (width, height) o ajustarse automáticamente según el contenido.

2. **Relleno (Padding):**

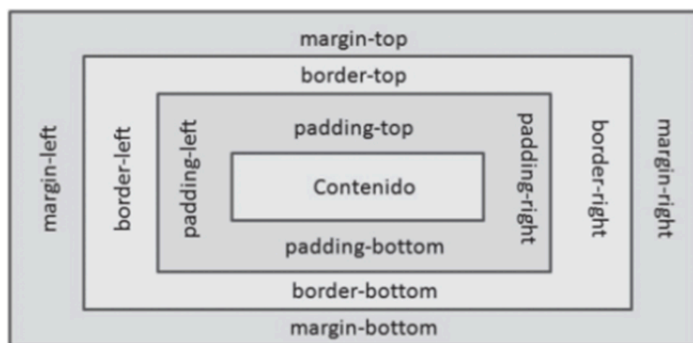
- Es el espacio entre el contenido y el borde de la caja.
- Aumenta el tamaño total de la caja sin afectar al contenido.

3. **Borde (Border):**

- Es el contorno alrededor del contenido y el padding.
- Puedes personalizar su color, grosor y estilo.

4. **Margen (Margin):**

- Es el espacio exterior que separa la caja de otros elementos.



5.1. Unidades de Medida

1. **Unidades absolutas:** Son fijas y no dependen del entorno del elemento.
 - **px** (píxeles): Una unidad fija, comúnmente usada para tamaños precisos.
 - **cm, mm** (centímetros, milímetros): Más útiles para impresión.
 - **in** (pulgadas): Rara vez se usa en diseño web.
2. **Unidades relativas:** Dependen del tamaño del elemento contenedor o del entorno.
 - **%**: Porcentaje relativo al contenedor padre.
 - **em**: Relativo al tamaño de fuente del elemento padre.
 - **rem**: Relativo al tamaño de fuente de la raíz (<html>).
 - **vw, vh**: Relativo al 1% del ancho (vw) o la altura (vh) del viewport (área visible de la ventana del navegador)
 - **vmin, vmax**: Relativo al 1% del menor (vmin) o mayor (vmax) lado del viewport.

5.2. Atributos de posición

Los atributos de posición controlan cómo se coloca una caja dentro de su contenedor o en relación con otros elementos. CSS ofrece varios tipos de posicionamiento:

Posicionamiento				
position:	static	absolute	relative	fixed
top/right/bottom/left:	[size]		auto	
clip-path:	url(shape.svg)		shape	auto
overflow:	visible	hidden	scroll	auto

1. **position:** Define el esquema de posicionamiento de un elemento.

- **static** (predeterminado): El elemento se posiciona según el flujo natural del documento.
- **relative**: Se posiciona en relación a su posición normal.
- **absolute**: Se posiciona en relación al contenedor más cercano con position: relative o absolute.
- **fixed**: Se posiciona en relación a la ventana del navegador (permanece fija al hacer scroll).
- **sticky**: Se posiciona en relación al scroll hasta que alcanza un punto específico.

2. **Propiedades relacionadas:**

- **top, right, bottom, left**: Controlan la distancia del elemento desde sus bordes contenedores según el esquema de posicionamiento.

3. **clip-path** define una región visible de un elemento, recortando las partes que quedan fuera de esa región. Esto crea efectos visuales de máscaras o recortes, sin alterar la posición ni el tamaño del elemento. La sintaxis general es:

```
clip-path: shape | url();
```

Formas básicas: se pueden usar formas geométricas para recortar el elemento.

- **circle(50%)**: Forma circular, con un radio del 50%.
- **ellipse(50% 25%)**: Elipse con un ancho del 50% y una altura del 25%.
- **polygon[x1 y1, x2 y2, ...]**: Define un polígono con puntos en coordenadas.
- **inset(top right bottom left)**: Define un rectángulo interior basado en distancias de los bordes.

```
clip-path: circle(50%); /* Recorta el elemento en forma de círculo */
```

URL de una máscara SVG: se puede usar un archivo SVG para definir una forma personalizada de recorte. Se trata de una opción **no recomendada** por razones de seguridad.

```
clip-path: url('shape.svg');
```

Lo mejor es incrustar el SVG dentro del HTML o utilizar mask-image.

- Con mask-image. Podemos descargar una forma SVG desde internet (por ejemplo <https://www.svgshapes.in/>) o crear un SVG nuevo con el editor (<https://editor.method.ac/>)

```
mask-image: url('assets/shape.svg');
mask-size: 100% 100%;
```

- Incrustarlo convirtiendo el SVG a código mediante chatgpt u otro programa en línea. O editando el svg directamente con el Notepad.

```
clip-path: url(#miClip); /* Referencia el ID del clipPath */
```

```

<body>
  <!-- Definir el SVG con clipPath -->
  <svg width="0" height="0">
    <defs>
      <clipPath id="miClip" clipPathUnits="objectBoundingBox">
        <path d="M1.01 0.5c0-0.05-0.04-0.09-0.09-0.09h-0.07l0.06-
0.04c0.05-0.03 0.06-0.08 0.03-0.13-0.03-0.05-0.08-0.06-0.13-0.03l-0.06
0.04 0.04-0.06c0.03-0.05 0.01-0.1-0.03-0.13-0.05-0.03-0.1-0.01-0.13
0.03l-0.04 0.06V0.09c0-0.05-0.04-0.09-0.09-0.05 0-0.09 0.04-0.09
0.09v0.07l-0.04-0.06c-0.03-0.05-0.08-0.06-0.13-0.03-0.05 0.03-0.06 0.08-
0.03 0.13l0.04 0.06-0.06-0.04c-0.05-0.03-0.1-0.01-0.13 0.03-0.03 0.05-
0.01 0.1 0.03 0.13l0.06 0.04h-0.07c-0.05 0-0.09 0.04-0.09 0.09 0 0.05
0.04 0.09 0.09 0.09h0.07l-0.06 0.04c-0.05 0.03-0.06 0.08-0.03 0.13 0.03
0.05 0.08 0.06 0.13 0.03l0.06-0.04-0.04 0.06c-0.03 0.05-0.01 0.1 0.03
0.13 0.05 0.03 0.1 0.01 0.13-0.03l0.04-0.06V0.91c0 0.05 0.04 0.09 0.09
0.09 0.05 0 0.09-0.04 0.09-0.09v-0.07l0.04 0.06c0.03 0.05 0.08 0.06 0.13
0.03 0.05-0.03 0.06-0.08 0.03-0.13l-0.04-0.06 0.06 0.04c0.05 0.03 0.1
0.01 0.13-0.03 0.03-0.05 0.01-0.1-0.03-0.13l-0.06-0.04h0.07c0.05 0 0.09-
0.04 0.09-0.09Z"></path>
      </clipPath>
    </defs>
  </svg>
  <!-- ----- -->

```

4. **overflow:** controla cómo se comporta el contenido cuando supera las dimensiones del contenedor definido (tamaño del Modelo de Caja).

- **visible (por defecto):** El contenido que excede los límites del contenedor es visible.
- **hidden:** El contenido que excede los límites del contenedor se oculta.
- **scroll:** Siempre muestra barras de desplazamiento, incluso si no hay contenido adicional que lo requiera.
- **auto:** Muestra barras de desplazamiento solo cuando el contenido excede los límites del contenedor.
- Aplicar overflow en direcciones específicas:
 - **overflow-x:** Controla el contenido horizontal.
 - **overflow-y:** Controla el contenido vertical.

5.3. Atributos de Margen

El margen es el espacio exterior entre la caja de un elemento y otros elementos.

margin: Define el espacio exterior de la caja. Puede aplicarse individualmente:

- **margin-top**
- **margin-right**
- **margin-bottom**
- **margin-left.**

Aplicación de valores:

- Un valor: Se aplica a los cuatro lados.
- Dos valores: El primero se aplica a top y bottom, el segundo a left y right.
- Tres valores: El primer valor es para top, el segundo para left y right, y el tercero para bottom.
- Cuatro valores: Se aplican en el orden top, right, bottom, left.
- **auto**: Centra horizontalmente un elemento bloque dentro de su contenedor si el ancho está definido.

Márgenes y espaciados				
margin/padding:	top	right	bottom	left
margin/padding:	top	right left	bottom	
margin/padding:	top	bottom	left right	
margin/padding:	top	right bottom	left	
	*-top *-left *-right *-bottom			

5.4. Atributos de Padding

El padding es el espacio interior entre el contenido y el borde de la caja.

padding funciona igual que margin en cuanto a valores y sintaxis.

```
padding: 10px; /* Aplica a los cuatro lados */  
padding: 10px 20px; /* Vertical | Horizontal */  
padding: 10px 20px 30px; /* Top | Horizontal | Bottom */  
padding: 10px 20px 30px 40px; /* Top | Right | Bottom | Left */
```

El padding aumenta el área de la caja sin afectar a otros elementos y es útil para separar el contenido de los bordes visuales.

5.5. Atributos de Borde

El borde rodea el contenido y el padding de una caja.

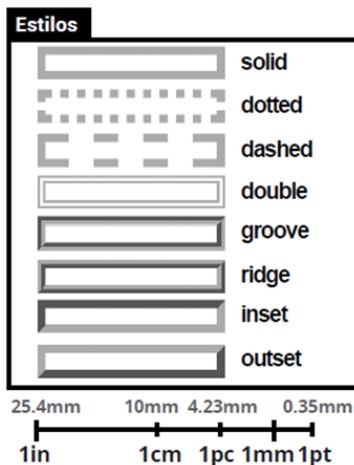


border es una propiedad abreviada para definir el ancho, estilo y color del borde.

```
border: 2px solid red;
```

Propiedades específicas:

- **border-width**: Grosor del borde.
- **border-style**: Estilo del borde: **none**, **solid**, **dotted**, **dashed**, **double**, **groove**, **ridge**, **inset**, **outset**.
- **border-color**: Color del borde.
- **border-radius**: Para redondear las esquinas



Se puede definir cada lado por separado:

```
border-top: 2px solid red;
border-right: 2px dashed green;
border-bottom: 2px dotted blue;
border-left: 2px double black;
border-radius: 10px;
```

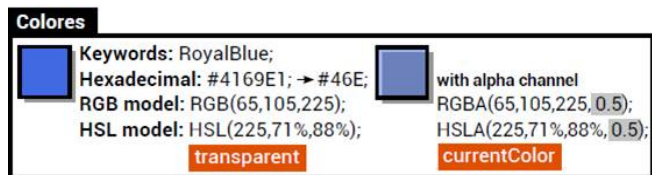
6. Atributos

En esta sección se muestran atributos adicionales de carácter más general y relacionados con la apariencia de fondos, textos, listas, tablas y enlaces.

6.1. Colores

DWEC04.- Estructuras definidas por el usuario en Javascript.

Existen diferentes formas de definir y manejar colores en CSS.



1. **Palabras clave (Keywords):** En CSS, puedes usar nombres de colores predefinidos (keywords) para definir un color.

```
background-color: RoyalBlue;
```

2. **Valores Hexadecimales (Hexadecimal):** Los colores también se pueden definir con un código hexadecimal. Un código hexadecimal consta de tres pares de dígitos que representan los valores de rojo, verde y azul en formato hexadecimal.

- o **Forma completa:** Por ejemplo #4169E1

41: Cantidad de rojo.

69: Cantidad de verde.

E1: Cantidad de azul.

- o **Forma corta:** Si los valores de los pares son iguales, puedes usar la versión corta. En este caso sería #46E

```
background-color: #4169E1; /* Código completo */
background-color: #46E; /* Forma corta */
```

3. **Modelo RGB (Red, Green, Blue):** Define colores usando valores decimales de rojo, verde y azul en una escala de 0 a 255.

```
background-color: rgb(65, 105, 225);
```

4. **Modelo HSL (Hue, Saturation, Lightness):** Describe los colores en términos de:

- o **Hue (Tono):** Representa el matiz en grados (0°-360°).

225°: Azul.

- o **Saturation (Saturación):** Intensidad del color (0%-100%).

71%: Moderadamente saturado.

- o **Lightness (Luminosidad):** Brillo del color (0%-100%).

88%: Muy brillante.

```
background-color: hsl(225, 71%, 88%);
```

5. **Canal Alpha (Transparencia):** Los modelos RGB y HSL pueden incluir un canal alpha para definir la opacidad. Se usa el modelo rgba o hsla añadiendo un cuarto parámetro que representa la transparencia. El rango del canal Alpha va desde 0 (completamente transparente) a 1 (completamente opaco).

```
background-color: rgba(65, 105, 225, 0.5);
background-color: hsla(225, 71%, 88%, 0.5);
```


6. **Modelo HWB:** Es otro modelo de color soportado en CSS para definir colores, introducido en CSS Color Module Level 4. Es una alternativa al modelo HSL, pero más intuitiva para ciertos ajustes. Este modelo es más descriptivo y cercano a cómo las personas perciben los colores.

HWB significa:

- o **H: Hue (Tono):** Define el matiz en grados (0°-360°), igual que en HSL.
- o **W: Whiteness (Blancura):** Es la cantidad de blanco que se mezcla con el color (0%-100%).
- o **B: Blackness (Negro):** Es la cantidad de negro que se mezcla con el color (0%-100%).
- o **alpha** (opcional): Nivel de opacidad (0 para completamente transparente, 1 para completamente opaco).

```
background-color: hwb(120 10% 10% / 0.5);
```

7. Propiedades especiales:

- o **transparent** hace que el color sea completamente transparente.

```
background-color: transparent;
```

- o **currentColor** usa el color actual del texto definido por color como el color del fondo o del borde.

```
color: red;  
border: 2px solid currentColor; /* El borde será rojo */
```

6.2. Colores de fondo

Colores y fondos

```
background-color: [color];           color: [color];
background-image: url(image.jpg); none
background-repeat: repeat repeat-x repeat-y no-repeat
background-attachment: scroll fixed
background-position: [pos-x] [pos-y];
background: color image repeat attachment position
```

background-color

Define el color de fondo de un elemento.

```
div{
  background-color: lightblue;
}
```

background-image

Especifica una imagen para usar como fondo del elemento.

```
div {
  background-image: url('background.jpg');
}
```

none: Elimina cualquier imagen de fondo (valor predeterminado).

```
background-image: none;
```

background-repeat

Controla cómo se repite una imagen de fondo.

- Valores:
 - **repeat:** Repite la imagen en ambas direcciones (horizontal y vertical).
 - **repeat-x:** Repite solo en la dirección horizontal.
 - **repeat-y:** Repite solo en la dirección vertical.
 - **no-repeat:** No repite la imagen.

```
body {
  background-image: url('background.png');
  background-repeat: no-repeat;
}
```

background-attachment

Controla si la imagen de fondo se mueve con el contenido o se queda fija.

Valores:

- **scroll:** La imagen se desplaza junto con el contenido (valor predeterminado).
- **fixed:** La imagen de fondo permanece fija y no se desplaza con el contenido.

```
body {  
  background-image: url('background.png');  
  background-repeat: no-repeat;  
  background-attachment: fixed;  
}
```

background-position

Establece la posición inicial de la imagen de fondo dentro del elemento.

- Sintaxis: background-position: [pos-x] [pos-y];
- **[pos-x]**: Posición horizontal (e.g., left, center, right, 50%, 10px).
- **[pos-y]**: Posición vertical (e.g., top, center, bottom, 50%, 10px).

```
body {  
  background-image: url('background.png');  
  background-repeat: no-repeat;  
  background-attachment: fixed;  
  background-position: center center;  
}
```

background

Propiedad abreviada: Permite definir todas las propiedades del fondo en una sola línea.

Sintaxis: background: [color] [image] [repeat] [attachment] [position];

```
body {  
  background: url('background.png') no-repeat fixed center;  
}
```

6.3. Fuentes

Fuentes

font-family: [font1], [font2], [font3], ... ;

serif sans-serif cursive fantasy monospace

font-size: [size] xx-small x-small small medium

large x-large xx-large smaller larger

font-style: normal italic oblique

font-weight: [100-900] normal bold lighter bolder

font: style variant weight size/height family

1. Propiedades principales:

font-family

- Especifica la fuente que se debe usar en el texto.
- Admite varias fuentes separadas por comas, donde la primera disponible en el sistema será utilizada.
- También puedes usar familias genéricas como **serif**, **sans-serif**, **cursive**, **fantasy**, o **monospace**.

```
font-family: 'Arial', 'Helvetica', sans-serif;
```

font-size

- Define el tamaño de la fuente.
- Valores admitidos:
 - Palabras clave: **small**, **medium**, **large**, etc.
 - Valores relativos: %, **em**, **rem**.
 - Valores absolutos: **px**, **pt**.

```
font-size: 18px;
```

font-style

- Cambia el estilo de la fuente.
- Valores admitidos:
 - **normal**: Sin inclinación.
 - **italic**: Texto en cursiva.
 - **oblique**: Texto inclinado artificialmente.

```
font-style: italic;
```

font-weight

- Controla el grosor de la fuente.
- Valores admitidos:
 - Palabras clave: **normal**, **bold**, **lighter**, **bolder**.
 - Números: De **100** (muy delgado) a **900** (muy grueso).

```
font-weight: 700; /* Negrita */
```

font

- Propiedad abreviada que permite especificar varias propiedades relacionadas con la fuente en una sola línea.
- Sintaxis: **font: style variant weight size/line-height family;**

```
font: italic 700 18px 'Arial', 'Helvetica', sans-serif;
```

2. Alineaciones y espaciado:

Fuente (alineaciones y espaciado)	
letter-spacing: [size];	normal
line-height: [size];	normal
text-indent: [size];	
word-spacing: [size];	normal
white-space:	normal no-wrap pre pre-line pre-wrap
tab-size: [size];	
text-align:	left center right justify
vertical-align: [size]	baseline
	sub super top middle bottom
	text-top text-bottom

letter-spacing

- Ajusta el espacio entre caracteres.
- Valores admitidos: **normal**, unidades (**px**, **em**, **%**, etc.).

```
letter-spacing: 10px;
```

line-height

- Ajusta la altura de la línea.
- Valores admitidos: **normal**, unidades (**px**, **%**, **em**, etc.).

```
line-height: 3.5;
```

text-indent

- Define el espacio de sangría en la primera línea de un párrafo.
- Valores admitidos: unidades (**px**, **%**, **em**, etc.).

```
text-indent: 80px;
```

word-spacing

- Ajusta el espacio entre palabras.
- Valores admitidos: **normal**, unidades (**px**, **em**, etc.).

```
word-spacing: 10px;
```

white-space

- Controla cómo se maneja el espacio en blanco.
- Valores admitidos:
 - **normal**: Colapsa espacios y saltos de línea.
 - **nowrap**: No permite saltos de línea.
 - **pre**: Preserva espacios y saltos de línea.
 - **pre-line**: Colapsa espacios pero preserva saltos de línea.
 - **pre-wrap**: Preserva espacios y saltos, y permite ajustar líneas.

```
white-space: pre-wrap;
```

text-align

- Alinea el texto horizontalmente.

- Valores admitidos: **left**, **right**, **center**, **justify**.

```
text-align: right;
```

vertical-align

- Alinea el texto verticalmente dentro de su contenedor.
- Valores admitidos: **baseline**, **top**, **middle**, **bottom**, **sub**, **super**.

```
vertical-align: middle;
```

3. Variaciones:

Fuentes (variaciones)			
font-variant:	normal	small-caps	
text-decoration:	none	underline	overline
		line-through	
text-transform:	none	capitalize	
		lowercase	uppercase

font-variant

- Controla el uso de mayúsculas pequeñas.
- Valores admitidos:
 - **normal**: Sin cambios.
 - **small-caps**: Convierte las letras minúsculas en mayúsculas pequeñas.

```
font-variant: small-caps;
```

text-decoration

- Añade decoraciones al texto.
- Valores admitidos:
 - **none**: Sin decoración.
 - **underline**: Subraya el texto.
 - **overline**: Línea sobre el texto.
 - **line-through**: Línea a través del texto.

```
text-decoration: overline;
```

text-transform

- Controla la capitalización del texto.
- Valores admitidos:
 - **none**: Sin cambios.
 - **capitalize**: Convierte la primera letra de cada palabra en mayúscula.
 - **uppercase**: Convierte todo a mayúsculas.
 - **lowercase**: Convierte todo a minúsculas.

```
text-transform: capitalize;
```

6.4. Enlaces

Para definir estilos en los enlaces, es importante conocer los diferentes estados en los que pueden encontrarse y sus respectivos selectores en CSS:

a:link

Enlace normal

- Aplica estilos a los enlaces que **aún no han sido visitados**.

```
a:link {  
  color: blue;  
  text-decoration: none;  
}
```

a:visited

Enlace visitado

- Se aplica a los enlaces que **ya han sido visitados** por el usuario.

```
a:visited {  
  color: purple;  
}
```

a:hover

Enlace con efecto hover

- Se aplica cuando el **cursor pasa sobre el enlace**, permitiendo efectos visuales.

```
a:hover {  
  color: orange;  
  text-decoration: underline;  
}
```

a:active

Enlace activo

- Se activa en el **momento exacto en que se hace clic en el enlace**.

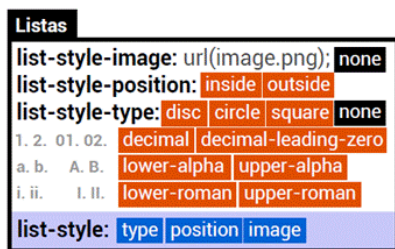
```
a:active {  
  color: red;  
}
```

El orden correcto en el CSS para evitar conflictos es:

```
a:link {  
    color: blue;  
    text-decoration: none;  
}  
a:visited {  
    color: purple;  
}  
a:hover {  
    color: orange;  
    text-decoration: underline;  
}  
a:active {  
    color: red;  
}
```


6.5. Listas

DWEC04.- Estructuras definidas por el usuario en Javascript.



list-style-type

Tipo de marcador de la lista

Define el **tipo de viñeta o numeración** de una lista o .

Valor	Ejemplo
disc (●)	● Elemento
circle (○)	○ Elemento
square (■)	■ Elemento
none (sin viñetas)	Elemento
decimal (1, 2, 3)	1. Elemento
decimal-leading-zero (01, 02, 03)	01. Elemento
lower-alpha (a, b, c)	a. Elemento
upper-alpha (A, B, C)	A. Elemento
lower-roman (i, ii, iii)	i. Elemento
upper-roman (I, II, III)	I. Elemento

```
ul {
  list-style-type: disc; /* Viñeta por defecto en listas no ordenadas*/
}

ol {
  list-style-type: decimal; /* Números en listas ordenadas */
}
```

list-style-position

Posición de la viñeta

Controla si las viñetas o números están **dentro** o **fuera** del área de la lista.

```
list-style-position: inside; /* La viñeta queda dentro del bloque */
```

- **outside** (por defecto): alineado normal
- **inside**: alineado con el texto

list-style-image

Usar una imagen como viñeta

Permite reemplazar la viñeta con una imagen personalizada. Si la imagen no carga, se mostrará el tipo de viñeta definido en list-style-type.

```
list-style-image: url('vineta.png'); /* Imagen en lugar de viñeta */
```

list-style

Atajo para definir todos los estilos

Permite combinar **type** + **position** + **image** en una sola línea.

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

6.6. Tablas

Tablas

border-collapse: `separate` `collapse`
border-spacing: `[size]`;
caption-side: `top` `bottom`
empty-cells: `show` `hide`
table-layout: `auto` `fixed`

border-collapse

Controla la separación entre los bordes de las celdas

Define si los bordes de las celdas deben estar **separados** o **colapsados** en una única línea.

- **separate:** (Por defecto) Las celdas tienen bordes separados.
- **collapse:** Los bordes de las celdas adyacentes se combinan en una sola línea.

```
border-collapse: collapse; /* Bordes unidos */
```

border-spacing

Espaciado entre celdas (si border-collapse: separate)

Controla la distancia entre los bordes de las celdas cuando border-collapse está en separate.

```
.tabla { /* tabla separada*/
  border-collapse: separate;
  border-spacing: 15px; /* Espaciado entre celdas */
}
```

caption-side

Posición del título de la tabla

Define si el <caption> (título de la tabla) aparece arriba (top) o abajo (bottom).

- **top:** (Por defecto) Arriba de la tabla.
- **bottom:** Debajo de la tabla.

```
caption-side: bottom; /* Mueve el título abajo */
```

empty-cells

Controla la visibilidad de celdas vacías

Permite mostrar u ocultar celdas que no tienen contenido. Solo funciona si border-collapse: separate.

- **show:** (Por defecto) Muestra las celdas vacías con bordes.
- **hide:** Oculta las celdas vacías sin afectar el diseño.

```
empty-cells: hide;
```

table-layout

Controla el ancho de la tabla

Define cómo el navegador distribuye el ancho de las columnas.

- **auto:** La tabla se ajusta automáticamente según el contenido.

- **fixed:** Todas las columnas mantienen un ancho proporcional sin importar el contenido (basado en el width de la tabla).

```
table-layout: fixed;  
width: 500px;
```

6.7. Visibilidad

Tipos de elementos

display:	inline	block	inline-block	none	list-item
	table	table-cell	table-row		
visibility:	visible	hidden	collapse		

display

Define cómo se muestra un elemento

La propiedad display controla la forma en que un elemento HTML se representa en la página.

Principales valores de display:

Valor	Descripción
inline	El elemento no inicia en una nueva línea y solo ocupa el ancho necesario. No permite modificar width ni height.
block	El elemento ocupa todo el ancho disponible y comienza en una nueva línea. Se puede modificar width y height.
inline-block	Similar a inline, pero permite modificar width y height.
none	Oculto el elemento completamente, no ocupa espacio en la página.
list-item	Se usa en elementos , mostrándolos como ítems de lista con viñetas o numeración.
table	Hace que un elemento se comporte como una tabla (<table>).
table-cell	Hace que un elemento se comporte como una celda de tabla (<td>).
table-row	Hace que un elemento se comporte como una fila de tabla (<tr>).

```
display: inline;
```

visibility

Controla la visibilidad sin modificar el espacio

La propiedad visibility permite ocultar elementos sin eliminar su espacio en la página.

- **visible:** El elemento es visible (por defecto).
- **hidden:** Oculta el elemento pero sigue ocupando espacio.
- **collapse:** Oculta elementos de una tabla (tr o td), similar a hidden, pero con un comportamiento especial en tablas.

```
visibility: hidden; /* Oculta el elemento pero sigue ocupando espacio.*/
display: none; /* Oculta el elemento y no ocupa espacio */
```

7. Grid Layout

Se trata un sistema de **diseño basado en cuadrículas** que se utiliza para estructurar y organizar elementos en una página web o aplicación. Su propósito es definir tanto el número de **filas** como de **columnas**, y controlar el posicionamiento y alineación de los elementos dentro de la cuadrícula. De esta manera se crea una distribución visualmente armoniosa y coherente, mejorando la legibilidad y la experiencia del usuario.

Para utilizar CSS Grid, el contenedor principal debe tener la propiedad:

display: grid

```
display: grid;
```

o

```
display: inline-grid;
```

```
display: inline-grid;
```

Inline-grid ajusta su ancho automáticamente al tamaño de su contenido.

Esto convierte al elemento en una **rejilla** en la que podemos definir el número de filas y columnas.

7.1. Propiedades del contenedor Grid

El contenedor controla la estructura general de la cuadrícula mediante las siguientes propiedades:

- **Número de columnas y filas:** Estas propiedades permiten establecer la cantidad y tamaño de filas y columnas dentro del grid.

grid-template-columns

Define el número de **columnas** y su tamaño.

```
grid-template-columns: 100px 200px auto;
```

Este ejemplo define tres columnas en el contenedor grid.

100px → Primera columna de 100 píxeles de ancho.

200px → Segunda columna de 200 píxeles de ancho.

auto → Tercera columna que se ajustará automáticamente al contenido o puede expandirse si hay espacio disponible.

Otros formatos:

```
grid-template-columns: repeat(3, 1fr); /* 3 columnas iguales de 1 fr*/
```

El contenedor grid tendrá 3 columnas, todas con el mismo tamaño, utilizando la unidad fr (fracción del espacio disponible).

Esto equivale a escribir:

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

Es decir, 3 columnas de igual tamaño que se reparten equitativamente el ancho del contenedor.

```
grid-template-columns: [inicio] 100px [medio] auto [final]; /* Definiendo nombres */
```

Esta línea define un grid con dos columnas y usa nombres personalizados (inicio, medio y final) para referirse a ciertos puntos dentro de la cuadrícula.

[inicio] → Nombre que marca el inicio de la cuadrícula.

100px → Primera columna de 100 píxeles de ancho.

[medio] → Nombre que marca el inicio de la segunda columna.

auto → Segunda columna que se ajusta automáticamente al contenido (y puede expandirse si hay más espacio disponible).

[final] → Nombre que marca el final de la cuadrícula.

grid-template-rows

Define el número de **filas** y su tamaño.

Lógicamente para que se creen las filas (con sus respectivos tamaños) tendrá que haber suficientes elementos para ocupar todas las filas.

```
grid-template-rows: 50px 100px auto; /* 3 filas */
```

```
grid-template-rows: 50px 100px auto; /* 3 filas */
```

También soporta repeat():

```
grid-template-rows: repeat(2, 45px); /* 2 filas de 45px */
```

· Separación entre Filas y Columnas

row-gap

column-gap

Define el espacio entre filas y columnas.

```
row-gap: 10px; /* 10px entre filas */
column-gap: 20px; /* 20px entre columnas */
```

También se puede usar gap para ambas:

```
gap: 10px 20px; /* 10px entre filas y 20px entre columnas */
```

· Distribución del Espacio Disponible

Si el grid no ocupa todo el contenedor, puedes distribuirlo con:

justify-content

(alineación horizontal del grid dentro del contenedor)

Controla la posición del grid dentro del contenedor en eje X.

justify-content: start | center | end | space-between | space-around | space-evenly;

```
justify-content: center; /* Centra el grid horizontalmente */
```

align-content

(alineación vertical del grid dentro del contenedor)

Controla la posición del grid dentro del contenedor en eje Y.

align-content: start | center | end | space-between | space-around | space-evenly;

```
align-content: space-between; /* Espacia las filas uniformemente */
```

```
align-content: space-between; /* Espacia las filas uniformemente */
```

Para que align-content funcione, el grid debe tener una altura mayor que la suma de sus filas.

· Alineación de los Elementos Dentro del Grid

Estas propiedades definen cómo se alinean los elementos dentro de cada celda.

justify-items

(alineación horizontal de los elementos dentro de celdas)

justify-items: start | center | end | stretch;

```
justify-items: stretch; /* Hace que los elementos ocupen todo el ancho de la celda */
```

align-items

(alineación vertical de los elementos dentro de celdas)

align-items: start | center | end | stretch;

```
align-items: center; /* Centra verticalmente el contenido de cada celda*/
```

place-items

(combinación de align-items y justify-items)

place-items: (align-items) (justify-items)

```
place-items: start end; /* Centra verticalmente y alinea a la izquierda*/
```

Plantillas de Áreas

grid-template-areas

Permite asignar nombres a secciones del grid.

grid-template-areas:

```
"nombre1 nombre2 nombre3"
```

```
"nombre4 nombre5 nombre6"
```

```
"nombre7 nombre8 nombre9";
```

Cada línea representa una fila, y cada palabra representa una columna.

Reglas:

- Las áreas deben formar rectángulos continuos (no pueden tener formas irregulares).
- El número de columnas debe coincidir en todas las filas.
- Se pueden dejar espacios vacíos usando un . (punto).

```
/* Definición de las áreas */
grid-template-areas:
  "A A D"
  "B B D"
  "C C C";
```

Ubicar un elemento en un área específica.

Los elementos dentro de un display: grid pueden controlarse con:

```
grid-column-start: 1;
grid-column-end: 3;
grid-row-start: 2;
grid-row-end: 4;
```

O en una sola línea:

```
grid-column: 1 / 3;
grid-row: 2 / 4;
```