UNIDAD 2

HTML. HOJAS DE ESTILOS

PARTE 3. HOJAS DE ESTILOS. CSS AVANZADO

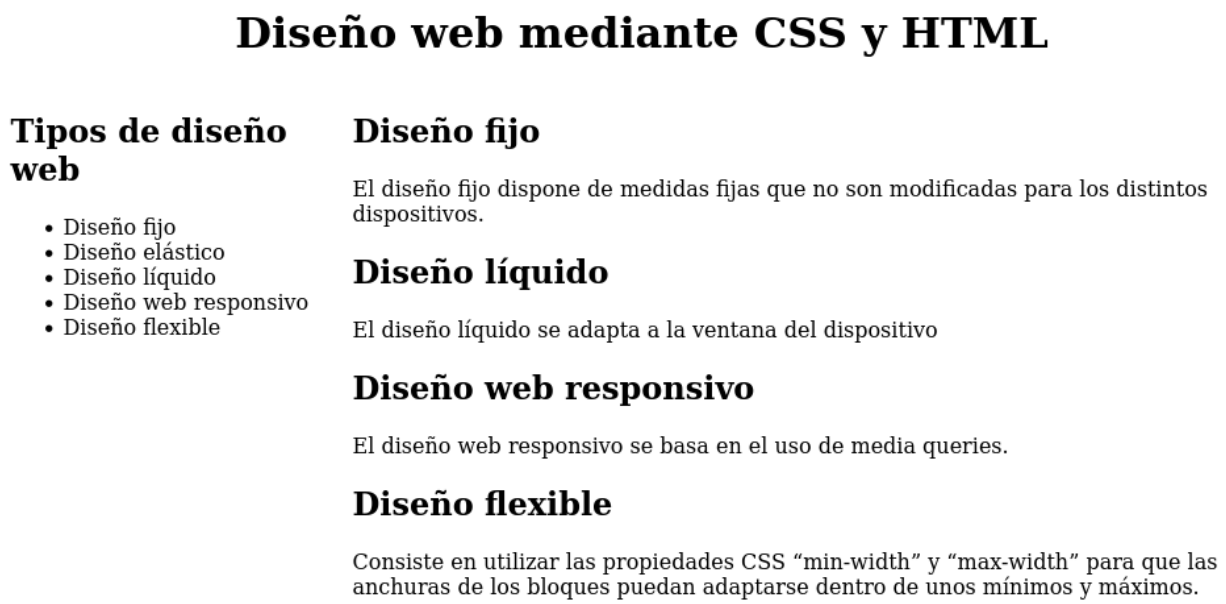
Índice

1.- Tipos de diseño web

A la hora de crear una interfaz web existen **diferentes tipos de diseño**: diseño con tamaño fijo, diseño con unidades «em», diseño mediante porcentajes, diseño que se adapta mediante puntos de ruptura o diseño definido dentro de unos márgenes predefinidos. Es importante mencionar que estos métodos **se pueden utilizar de forma combinada**. Veamos cada uno de estos métodos mediante un ejemplo.

Partamos de una web inicial para su estudio con los diferentes tipos de diseño. El contenido html es el siguiente:

```
<h1>Diseño web mediante CSS y HTML</h1>

<aside>
  <h2>Tipos de diseño web</h2>

  <ul>
  <li>Diseño fijo</li>
  <li>Diseño elástico</li>
  <li>Diseño líquido</li>
  <li>Diseño web responsivo</li>
  <li>Diseño flexible</li>
  </ul>
</aside>

<section>
  <h2>Diseño fijo</h2>
  <p>
  El diseño fijo dispone de medidas fijas que no son modificadas para los
distintos dispositivos.
  </p>

  <h2>Diseño líquido</h2>
  <p>
  El diseño líquido se adapta a la ventana del dispositivo
  </p>

    <h2>Diseño web responsivo</h2>
  <p>
  El diseño web responsivo se basa en el uso de media queries.

    <h2>Diseño flexible</h2>
  <p>
  Consiste en utilizar las propiedades CSS “min-width” y “max-width” para que
las anchuras de los bloques puedan adaptarse dentro de unos mínimos y máximos.
  </p>
</section>
```



Diseño web mediante CSS y HTML

Tipos de diseño web

- Diseño fijo
- Diseño elástico
- Diseño líquido
- Diseño web responsivo
- Diseño flexible

Diseño fijo

El diseño fijo dispone de medidas fijas que no son modificadas para los distintos dispositivos.

Diseño líquido

El diseño líquido se adapta a la ventana del dispositivo

Diseño web responsivo

El diseño web responsivo se basa en el uso de media queries.

Diseño flexible

Consiste en utilizar las propiedades CSS “min-width” y “max-width” para que las anchuras de los bloques puedan adaptarse dentro de unos mínimos y máximos.

1.1.- Diseño fijo

El **diseño fijo** dispone de **medidas fijas** que no son modificadas para los distintos dispositivos. Por lo tanto, se trata de un diseño que **no cumple las normas de un diseño web responsivo o *responsive web design***.

```
body {
  width: 960px;
  margin: 0 auto;
}

h1 {
  text-align: center;
}

aside, section {
  float: left;
  padding: 0;
  margin: 0;
}

aside{
 width: 260px;
}

section {
  width: 700px;
}
```

1.2.- Diseño elástico

En el diseño elástico se definen las **dimensiones** de la página **con unidades «em»**. De esta forma el diseño **se adaptará cuando el visitante del sitio cambie el tamaño del texto**, pero **no se adaptará en función de los cambios de tamaño de la ventana del navegador**.

Tal y como vimos en el apartado “Unidades de medida en CSS” de la unidad, el «em» es una unidad relativa y su valor depende del tamaño de fuente definido en el navegador.

El «em» es una unidad de tipografía que mide exactamente el ancho de la letra «M» mayúscula de una tipografía dada y a un tamaño dado. Por tanto, un «em» no mide siempre lo mismo. Actualmente, el tamaño de fuente por defecto en todos los navegadores suele ser de 16 píxeles.

```
body {
  width: 60em;
  margin: 0 auto;
}

h1 {
  text-align: center;
}

section, aside {
  float: left;
  padding: 0;
  margin: 0;
}

aside {
 width: 16em;
}

section {
  width: 44em;
}
```

1.3.- Diseño líquido o fluido

El diseño líquido se adapta a la ventana del dispositivo, normalmente definiendo los tamaños mediante **porcentajes**. Sin embargo, este método perjudica la experiencia de usuario y no permite controlar el diseño (el diseño varía constantemente según el tamaño del dispositivo y no se puede diseñar al píxel para todos los tamaños).

```
body {
```

```
  width: 80%;
  margin: 0 auto;
}

h1 {
  text-align: center;
}

aside, section {
  float: left;
  padding: 0;
  margin: 0;
}

aside {
 width: 25%;
}

section {
  width: 75%;
}
```

1.4.- Diseño web responsivo o adaptable

El diseño web r**esponsivo o *responsive web design*** (también llamado **adaptable o *adaptative***) se basa en el uso de ***media queries***. Gracias a esta técnica podemos **definir estilos condicionales con puntos de ruptura o puntos de interrupción, aplicables únicamente en determinadas situaciones.**

Supongamos que queremos aplicar un estilo diferente en pantallas de ancho superior a 1200px, inferior a 1200px, e inferior a 600px, tendríamos las siguientes media queries para definir los estilos del ejemplo que hemos estado viendo:

```
body {
  width: 80%;
  margin: 0 auto;
}

@media (min-width: 1201px) {
  h1 {
    text-align: center;
  }

  aside, section {
    float: left;
    padding: 0;
    margin: 0;
  }

  aside {
   width: 20%;
```

```
  }

  section {
   width: 80%;
  }
}

@media (min-width: 701px) and (max-width: 1200px) {
  h1 {
    text-align: center;
  }

  aside, section {
    float: left;
    padding: 0;
    margin: 0;
  }

  aside {
   width: 30%;
  }

  section {
    width: 70%;
  }
}

@media (max-width: 700px) {
  aside ul {
    padding-left: 0;
  }

  aside li {
    display: inline;
  }

  aside li:after {
    content: ", ";
  }
}
```

1.5.- Diseño flexible

Consiste en utilizar las propiedades CSS “**min-width**” y “**max-width**” para que las anchuras de los bloques puedan adaptarse dentro de unos mínimos y máximos.

```
body {
  max-width: 900px;
  margin: 0 auto;
}

h1 {
  text-align: center;
}
```

```
aside, section {
  float: left;
  padding: 0;
  margin: 0;
}

aside {
 width: 25%;
}

section {
  width: 75%;
}
```

1.6.- ¿Qué tipo de diseño debo utilizar?

Crear un sitio web adaptable exige, en muchos casos, **combinar algunas de las técnicas** anteriores. Lo más habitual es **declarar más de una *media query*** para adaptar el diseño a múltiples dispositivos. Pero el número de puntos de interrupción y cómo se implementan estas técnicas depende del diseño específico del sitio web.

Así pues, **lo ideal es utilizar un diseño web responsivo, preferentemente no elástico y con mezcla de diseño flexible y líquido según el contenedor**. De este modo ofreceremos la mejor experiencia de usuario y podremos considerar las distintas medidas de los dispositivos utilizando tan solo HTML y CSS.

1.7.- Tipos de diseño web según las propiedades CSS utilizadas

Veamos otro enfoque de los tipos de diseño web según las propiedades CSS que se utilicen:

1.7.1.- Diseño convencional con bloques y floats

Este es el enfoque tradicional de diseño en CSS, donde los elementos se colocan en el flujo normal del documento y se posicionan utilizando propiedades como **`float`**, **`position`**, **`display`**, entre otras. Aunque este enfoque todavía se utiliza, puede ser complicado y difícil de mantener para diseños más complejos y responsivos.

1.7.2.- Diseño con Flexbox

Flexbox es un modelo de diseño unidimensional que permite **organizar los elementos en una sola dirección (fila o columna)**. Con Flexbox, es más fácil alinear y distribuir

elementos de manera flexible, lo que facilita la creación de diseños responsivos y complejos.

1.7.3.- Diseño con CSS Grid

CSS Grid es un modelo de **diseño bidimensional que permite organizar los elementos en filas y columnas**, creando así diseños más complejos y estructurados. Con CSS Grid, se puede lograr un mayor control sobre la posición y el tamaño de los elementos en el diseño.

Cada enfoque tiene sus ventajas y desventajas, y el mejor enfoque dependerá de los requisitos específicos del diseño y de las preferencias del desarrollador. En muchos casos, una combinación de varios enfoques puede ser utilizada para obtener el mejor resultado en términos de flexibilidad y responsividad en el diseño web.

2.- Medios en CSS

2.1.- Estilos para diferentes medios en CSS

Las hojas de estilos CSS permiten **definir diferentes estilos para diferentes medios** o dispositivos: pantallas, impresoras, proyectores, televisores, etc. La tabla siguiente muestra el nombre que se utiliza en CSS para identificar cada medio:

Medio	Descripción
`all`	Todos los dispositivos
`print`	Para documentos paginados y mostrados en vista de impresión
`screen`	Pantallas de ordenador
`speech`	Sintetizadores para navegadores de voz utilizados por personas discapacitadas

Una de las ventajas de CSS es que nos permite modificar los estilos de una página en función del medio en el que se visualiza. Hay tres formas diferentes de indicar el medio en el que se deben aplicar los estilos CSS. Veamos cada una de ellas.

2.2.- Medios definidos con <link>

Se puede utilizar la etiqueta **`<link>`** en el código html **para enlazar los archivos CSS externos**. Para ello utilizaremos el atributo media para indicar el `medio` en el que se aplica el estilo de cada archivo mencionado en el atributo `href`.

```
<link rel="stylesheet" type="text/css" media="screen" href="style.css" />
<link rel="stylesheet" type="text/css" media="print"
href="especialStyle.css" />
```

2.3.- Medios definidos con @media

Mediante la **regla `@media`** podemos **indicar de forma directa el medio o medios en los que se aplicarán los estilos**. Para especificar el medio en el que se aplican los estilos, se incluye su nombre después de `@media`. Si los estilos se aplican a varios medios, se incluyen los nombres separados mediante comas.

En el siguiente ejemplo se define que el tamaño de letra de la página visualizada en una pantalla será de 12 píxeles. Sin embargo, cuando se impriman los contenidos de la página, su tamaño de letra será de 11 puntos. En cualquier caso el texto de la letra será de color azul.

```
@media print  { body { font-size: 11pt; } }
@media screen { body { font-size: 12px; } }
@media { body { color: blue; } } /* Se aplica a todo */
```

2.4.- Medios definidos con @import

Gracias a las reglas de tipo **`@import`** se pueden **enlazar archivos CSS externos para indicar los estilos que se aplican en cada medio**. Este método viene muy bien para no tener todos los estilos en una misma hoja.

En el ejemplo siguiente se carga el archivo “estilosPantalla.css” cuando la página se visualiza por pantalla. Cuando la página se imprime se carga el archivo “estilosImpresora.css”. El archivo estilos.css se cargará en ambos medios.

```
@import url("estilosPantalla.css") screen;
@import url("estilosImpresora.css") print;
@import url("estilos.css"); /* Se aplica a todo por defecto*/
```

2.5.- Hojas de estilo auditivas, CSS Speech Module

Las hojas de estilo auditivas proporcionan información para usuarios invidentes y de navegadores de voz de manera parecida a la que se proporciona visualmente.

> "Las propiedades CSS definidas en el módulo del habla (CSS Speech Module) permiten a los autores controlar de forma declarativa la presentación de un documento en la versión auditiva.
>
> El procesamiento auditivo de un documento combina la síntesis de voz (también conocido como "**TTS", el acrónimo de "Text to Speech**") y los iconos auditivos ("audio cues" en esta especificación).
>
> Las propiedades CSS del habla proporcionan la **capacidad de controlar el tono del habla y velocidad, el volumen, voces TTS utilizadas**, etc.
>
> Estas propiedades CSS **pueden ser utilizadas junto con las propiedades visuales** (medios mixtos), o **como una alternativa completa fonética a una presentación visual**."
>
> *Origen W3C*

3.- Media queries CSS

Las *media queries* se introdujeron en CSS3 para dar respuesta a las necesidades del diseño web *responsive*. Mediante ellas podemos **definir estilos condicionales, aplicables únicamente en determinadas situaciones.** El uso más extendido de las *media queries* es para establecer estilos diferentes para cada **ancho de pantalla**.

3.1.- Media queries en CSS para cada ancho de pantalla

Para **definir una media query en CSS** se utiliza la regla `@media` en la hoja de estilos. Se pueden definir todas las media queries que necesitemos según los diseños para cada ancho de pantalla.

¡Ojo! Como vimos en la unidad anterior, es necesario **enlazar la hoja de estilo externa en el documento HTML**.

Para definir los estilos para cada ancho de pantalla utilizaremos puntos de ruptura. Todos los estilos que no se incluyan dentro de una media query se verán en todos los tamaños de dispositivo.

Existen varias formas de definir el ancho del dispositivo dentro de la regla `@media`:

3.1.1.- Media query indicando el ancho mínimo de pantalla

Todos los estilos que se incluyan en el interior de la siguiente media query serán utilizados en pantallas que tengan un ancho mínimo de 768px (punto de ruptura de 768px).

```
@media (min-width: 768px) {
/* Estilos para pantallas con un ancho mínimo de 768px */
}
```

3.1.2.- Media query indicando el ancho máximo de pantalla

Todos los estilos que se incluyan en el interior de la siguiente media query serán utilizados en pantallas que tengan un ancho máximo de 767px (punto de ruptura de 767px).

```
@media (max-width: 767px) {
/* Estilos para pantallas con un ancho máximo de 768px */
}
```

3.1.3.- Media query combinando múltiples condiciones

Es posible combinar múltiples condiciones en una media query para aplicar estilos solo cuando se cumplan todas las condiciones. Todos los estilos que se incluyan en el interior de la siguiente media query serán utilizados en pantallas que tengan un ancho entre 768px y 1024px (puntos de ruptura de 768px y 1024px).

```
@media (min-width: 768px) and (max-width: 1024px) {
/* Estilos para pantallas con un ancho entre 768px y 1024px */
}
```

3.2.- Ejemplos de Media queries para definir el ancho del dispositivo

Ejemplo 1. Imaginemos que partimos del siguiente ejemplo de html:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <title>Ejemplo Media queries</title>
  </head>
    <body>
       <h3>Media queries</h3>
      <div class="click">

      </div>
    </body>
</html>
```

Supongamos que queremos aplicar un estilo diferente en pantallas de ancho superior a 1024px, inferior a 1024px e inferior a 480px, tendríamos las siguientes *media queries*:

```
.click:after {
    content:"En pantalla grande";
}
@media screen and (max-width: 1024px) {
    .click:after {
        content:"Tablet";
    }
}
@media screen and (max-width: 480px) {
    .click:after {
        content:"Movil";
    }
}
```

Ejemplo 2. La siguiente *media query* especifica una hoja de estilo para ser utilizada cuando la ventana tiene un ancho entre 450 y 800 píxeles:

```
.click:after {
    content:"En pantalla grande";
}
@media (min-width: 450px) and (max-width: 800px) {
       .click:after {
        content:"Tablet";
    }
}
```

3.3.- Otras propiedades o condiciones

A parte del ancho hay muchas otras propiedades que se pueden comprobar para aplicar o no los estilos. Las siguientes son algunas de ellas:

Nombre	Descripción
`width`	Anchura del *viewport*
`height`	Altura del *viewport*
`aspect-ratio`	Relación de aspecto anchura-altura del *viewport*
`orientation`	Orientación del *viewport*
`resolution`	Densidad de píxeles del dispositivo
`scan`	Proceso de escaneo del dispositivo
`grid`	Si el dispositivo es grid o bitmap
`update-frequency`	Velocidad de actualización del dispositivo para modificar la apariencia del contenido
`overflow-block`	Cómo maneja el dispositivo el contenido que excede los límites del *viewport* a lo largo del eje de bloque

Nombre	Descripción
`overflow-inline`	Cómo maneja el dispositivo el contenido que excede los límites del eje *inline*
`color`	Componente de número de bits por color del dispositivo, o cero si el dispositivo no es a color
`color-index`	Número de entradas en la tabla de búsqueda de color del dispositivo, o cero si el dispositivo no usa una tabla
`monochrome`	Bits por píxel en el buffer de marco monocromático del dispositivo, o 0 si el dispositivo no es monocromático
`hover`	Si se puede posicionar el puntero sobre los elementos

3.4.- Operadores lógicos en *media queries*

Las *media queries* nos permiten utilizar operadores lógicos para comprobar si se cumple una condición.

3.4.1.- Operador AND

El siguiente código aplicará estilos cuando la pantalla tenga un ancho mínimo de 700px y la orientación de la misma sea horizontal (*landscape*).

```
@media (min-width: 700px) and (orientation: landscape) { ... }
```

3.4.2.- Operador NOT

El siguiente código aplicará estilos cuando no se cumpla la condición especificada, es decir a cualquier elemento que no sea una pantalla convencional ni sea monocromo.

```
@media not screen and (monochrome) { ... }
```

4.- Metaetiqueta viewport

La declaración ***viewport*** nos **permite definir los parámetros de visualización de un página web en los diferentes dispositivos**. Consiste en una metaetiqueta mediante la que se establece si se puede hacer zoom en una página, el zoom inicial o la anchura de la pantalla del dispositivo.

4.1.- Configuración metaetiqueta viewport

Cuando trabajamos con una ***web responsive*** es necesario **definir un *viewport* adecuado**, de lo contrario es muy probable que la página no lea correctamente los *media queries* y que se vea en formato muy reducido, siendo necesario hacer zoom para ver el contenido.

La etiqueta *viewport* se define en el elemento <head> del documento HTML y puede contener diferentes atributos para ajustar el comportamiento de la página en dispositivos móviles.

```
<meta name="viewport" content="...">
```

Disponemos de los siguientes parámetros en la metaetiqueta *viewport*.

- **Width**: anchura virtual (emulada) de la pantalla o anchura del *viewport*.
- **Height**: altura virtual de la pantalla o altura del *viewport*.
- **Initial-scale:** la escala inicial del documento.
- **Minimum-scale:** la escala mínima que se puede establecer en el documento.
- **Maximum-scale:** la escala máxima configurable en el documento.
- **User-scalable**: si se permite al usuario hacer zoom.

Un ejemplo de cómo se vería la metaetiqueta «viewport» en el código HTML es el siguiente:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Mi página web</title>
</head>
<body>
<!-- Contenido de la página -->
</body>
</html>
```

4.1.1.- Ejemplo de la etiqueta *viewport* que no permite escalar la página

Lo más habitual es definir la anchura con el valor “device-width”, que es una medida que hace referencia a la anchura de la pantalla del dispositivo, y no con una medida fija. Por lo tanto, con **width=device-width** conseguimos que el ***viewport* sea igual a la anchura real de la pantalla** del dispositivo, de modo que no se tratará de emular una pantalla mayor de lo que realmente es y veremos los píxeles reales.

Con **initial-scale=1** conseguimos **definir la escala inicial** del documento para evitar transformaciones.

```
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
```

Con **user-scalable=no** o con **maximum-scale con un valor igual al de initial-scale** conseguimos que el usuario **no pueda hacer zoom** en la página, con lo que siempre se mantendrán las medidas que nosotros hemos definido al construir la web.

```
<meta name="viewport" content="user-scalable=no, width=device-width, initial-scale=1">
```

4.1.2.- Ejemplo etiqueta *viewport* que permite escalar la página

Para no limitar al usuario el uso del zoom en nuestra página web y darle la posibilidad de agrandar y empequeñecer cualquier parte que necesite ver mejor lo ideal es **no definir ni el maximum-scale ni el user-scalable=no**.

Ejemplo:

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

Por lo tanto, es interesante que el usuario pueda hacer zoom en la página web (en un smartphone será con los dedos sobre la pantalla táctil). En este caso, el usuario no está cambiando los valores de *viewport*, sino la escala con la que se visualiza el documento.

5.- Sombras CSS

Gracias a las propiedades de CSS3 “**text-shado**w” y “**box-shadow**” podemos aplicar sombras en los textos y en los contenedores para así enriquecer nuestras interfaces.

5.1.- Propiedades text-shadow y box-shadow

El formato es prácticamente el mismo para las dos propiedades.

- **text-shadow:** posx posy desenfoque color | none;
- **box-shadow:** posx posy desenfoque tamaño color inset | none;

Significado de las propiedades:

- **posx**: sombra que se aplica en el eje horizontal.
- **posy**: sombra que se proyecta en el eje vertical.
- **desenfoque o *blur***: cantidad de sombra borrosa.
- **tamaño**: tamaño de la sombra hacia los lados.

- **color**: color de la sombra.
- **inset**: posiciona la sombra dentro del marco.

Importante:

- Para aplicar sombras arriba y a la izquierda se deben utilizar valores negativos.
- Para aplicar doble sombra se pueden poner dos valores separados con coma.
- Recuerda que para este tipo de propiedades debemos incluir los prefijos para navegadores.

Veamos por ejemplo qué significan los valores de la propiedad “text-shadow” del siguiente ejemplo:

```
text-shadow: 3px 4px 0px #fff;
box-shadow: 10px 10px 14px 2px rgba(0,0,0,0.47);
```

- 3px: es el tamaño de la sombra que se aplica a la derecha del texto.
- 4px: es la sombra que se proyecta hacia abajo.
- 0px: es la cantidad de sombra borrosa que se aplica.
- #fff: es el color de la sombra. Se puede aplicar también en RGBA y HSLA.

Para aplicar sombras arriba y a la izquierda se deben utilizar valores negativos.

Para aplicar varias sombras se puede añadir una lista de sombras separadas por comas. En el siguiente ejemplo se aplicarían dos sombras.

```
text-shadow: 3px 4px 0px #fff, 2px 2px 1px #ccc;
```

Recuerda que para este tipo de propiedades debemos incluir los prefijos para navegadores correspondientes.

5.1.1.- Ejemplo

Prueba las siguientes propiedades sobre diferentes párrafos y observa las diferencias entre los distintos valores.

```
<p class="shadow1">Texto con sombra</p>
<p class="shadow2">Texto en contenedor con sombra</p>
<p class="shadow3">Texto en contenedor con sombra interior</p>
<p class="shadow4">Texto con dos sombras</p>
```

```
.shadow1 { text-shadow: 3px 3px 2px rgba(152, 150, 207, 0.7);}
.shadow2 {box-shadow: 10px 10px 14px 2px rgba(0,0,0,0.47);}
.shadow3 {box-shadow: 5px 1px 4px 2px #45f875 inset;}
.shadow4 {text-shadow: 3px 4px 0px grey, 2px 2px 1px yellow;}
```

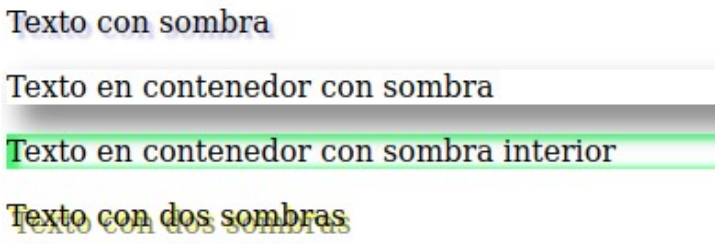


Figure 1: Ejemplo de sombreado

5.2.- Generadores de sombras online

Para facilitarnos la tarea de la creación de sombras podemos utilizar un generador de sombras tanto para los textos como para las cajas o contenedores. Algunas herramientas online útiles son las siguientes:

Generador de sombras en cajas o contenedores: cssmatic.com/box-shadow

6.- Gradientes CSS

Los gradientes nos permiten añadir efectos de colores degradados en nuestros diseños. Los gradientes están disponibles desde CSS3 y se configuran como fondos, por lo que tendremos que usar la propiedad “background“. Disponemos de dos tipos de gradientes: lineal y radial.

6.1.- Gradiente lineal

El gradiente lineal se define mediante la función **linear-gradient()** y permite crear fondos degradados en una dirección específica. El formato es el siguiente:

```
background: linear-gradient(dirección|ángulo, color, color, color...);
background: linear-gradient(dirección|ángulo, color tamaño, color tamaño, color tamaño...);
background: linear-gradient(color, color, color…);
```

Significado de las propiedades:

- **Dirección | ángulo:** indica la dirección o ángulo del gradiente. Puede ser especificado usando las palabras clave *to top, to bottom, to left* y *to right*. También puede ser indicado por un ángulo para declarar una dirección específica del gradiente.
- **Color:** permite crear una lista con varias paradas para ir cambiando el color del gradiente. Se pueden definir tantos colores como se necesiten. Se puede utilizar el nombre de color (en inglés), valor hexadecimal, color en RGB o HSL.

- **Tamaño:** altura a la que comienza a cambiar el color del gradiente.

Recuerda utilizar la extensión que te facilita la tarea de crear los **prefijos para navegadores**. En estas propiedades hay diferencias entre las funciones estándar y las especificadas por los navegadores.

6.1.1.- Ejemplo

Prueba las siguientes propiedades en diferentes contenedores y observa las diferencias entre los distintos valores.

```
background: linear-gradient(90deg, rgba(176,174,238,1) 0%, rgba(29,30,31,1));
background: linear-gradient(to right,
red,brown,orange,yellow,blue,green,violet,pink);
background: linear-gradient(to top, #000, #ccc);
background: linear-gradient(to right,red 300px,brown 400px,orange 700px);
```

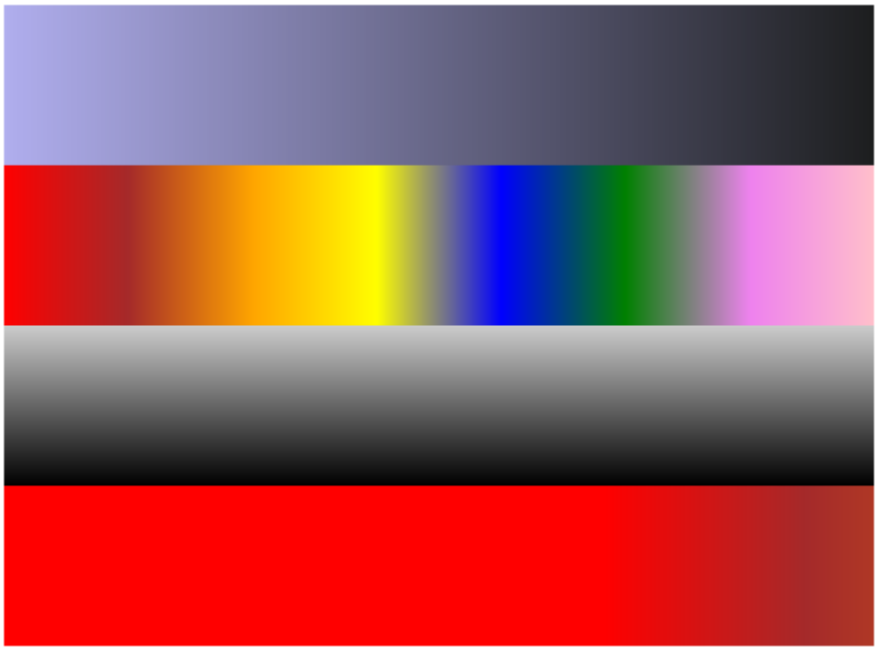


Figure 2: Ejemplo de gradiente lineal

6.2.- Gradiente radial

El gradiente radial nos permite crear degradados con formas circulares. El formato para los gradientes radiales es muy parecido al anterior. En este caso, debemos usar la función **radial-gradient()** y un nuevo atributo para la forma.

```
background: radial-gradient(forma, color, color, color...);
background: radial-gradient(forma tamaño, color, color, color...);
```

```
background: radial-gradient(forma, color tamaño, color tamaño, color
tamaño...);
background: radial-gradient(color, color, color...);
```

Significado de las propiedades:

- forma: existen dos tipos de forma: *circle* y *ellipse* (el valor por defecto será *ellipse*). De forma optativa se puede indicar el tamaño de la forma.
- color: permite crear una lista con varias paradas para ir cambiando el color del gradiente. Se pueden definir tantos colores como se necesiten.
- tamaño: para cada color se puede indicar la posición en la que comienzan las transiciones.

Ejemplo

Prueba las siguientes propiedades en diferentes contenedores y observa las diferencias entre los distintos valores.

```
background: radial-gradient(ellipse, rgba(176,174,238,1) 0%, rgba(29,30,31,1));
background: radial-gradient (circle,red,brown,orange,yellow,blue);
background: radial-gradient(circle,red 300px,brown 400px,orange 700px);
background: radial-gradient(#000, #ccc);
```

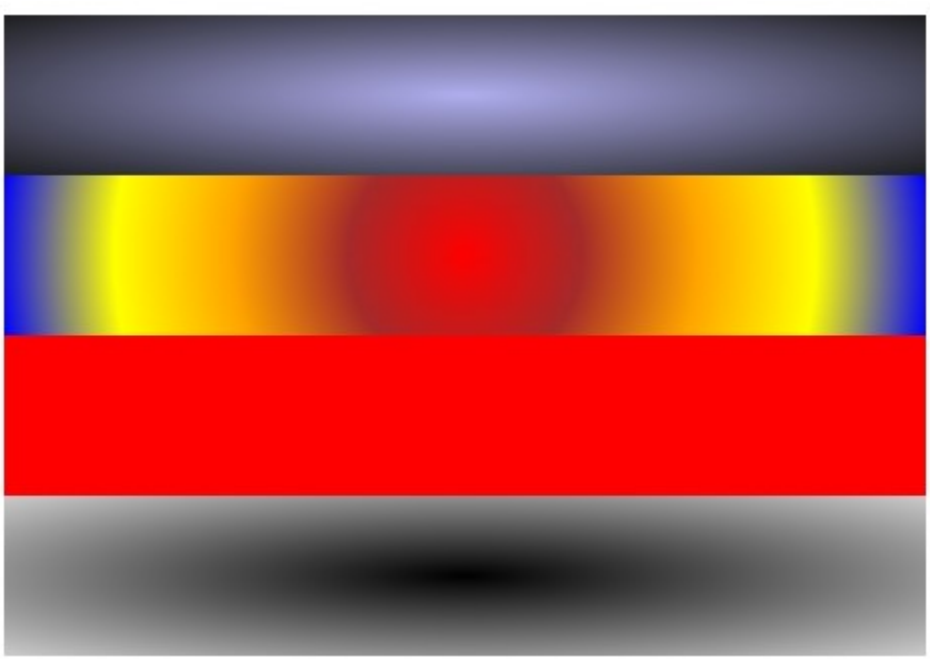


Figure 3: Ejemplo de gradiente radial

6.3.- Generadores de gradientes

Para facilitarnos la tarea de la **creación de gradientes** podemos utilizar un **generador de gradientes online** que nos proporcione el código CSS necesario para nuestro diseño. Algunas herramientas online útiles son las siguientes:

- cssgradient.io
- css3gen.com/gradient-generator

Ejemplo de gradiente otenido a partir de un generador:

```
background: rgb(176,174,238);
background: radial-gradient(circle, rgba(176,174,238,1) 0%, rgba(29,30,31,1)
100%);
```

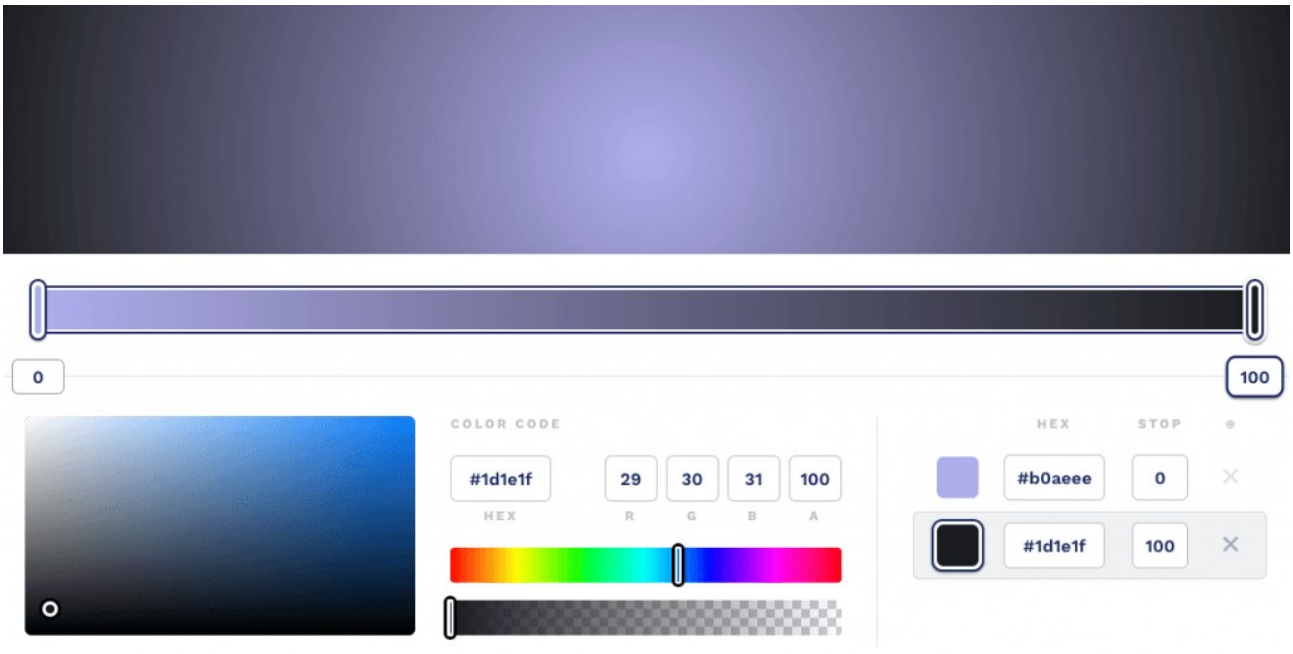


Figure 4: Ejemplo de generador de gradientes

6.4.- Patrones de gradientes

Podemos aplicar diseños muy trabajados creando **patrones** sin necesidad de cargar imágenes con mucho peso. Existen diversos artistas dedicados a hacer este tipo de diseños. Algunos ejemplos de diseños de este tipo se pueden obtener en la siguiente plataforma:

- leaverou.github.io/css3patterns

7.- Selector de atributos en CSS

Anteriormente vimos algunas de las pseudo-clases de CSS que nos permitían realizar una selección más específica de los elementos a los que queremos aplicar un cierto estilo. Por ejemplo, utilizábamos la jerarquía de los elementos para especificar la selección.

En esta sección vamos a ver los selectores de atributos que nos permitirán también realizar una selección más específica de los elementos a los que queremos aplicar un cierto estilo en función de los atributos o valores que tengan asignados.

7.1.1.1.- Selector de atributos en CSS

Los **selectores de atributos** permiten elegir elementos HTML en función de sus atributos y/o **valores** de esos atributos.

Los tipos de selectores de atributos son los siguientes:

- [**nombre_atributo**], selecciona los elementos que tienen establecido el atributo llamado nombre_atributo independientemente de su valor.
- [**nombre_atributo=valor**], selecciona los elementos que tienen establecido un atributo llamado nombre_atributo con un valor igual a valor.
- [**nombre_atributo~=valor**], selecciona los elementos que tienen establecido un atributo llamado nombre_atributo y al menos uno de los valores del atributo es valor.
- [**nombre_atributo|=valor**], selecciona los elementos que tienen establecido un atributo llamado nombre_atributo, cuyo valor es una lista de valores, donde alguno comienza por valor.
- [**nombre_atributo$=valor**] Selecciona aquellas etiquetas cuyo atributo acabe en ese valor.
- [**nombre_atributo^=valor**] Selecciona aquellas etiquetas cuyo atributo comience por ese valor.
- [**nombre_atributo*=valor**] El atributo nombre_atributo existe y contiene valor.

Ejemplo

A continuación se muestran algunos ejemplos de estos tipos de selectores:

Atributo	Descripción
`[href]`	El atributo `href` existe en la etiqueta.
`[href="#"]`	El atributo `href` existe y su valor es igual al texto #.
`[href*="eniun"]`	El atributo `href` existe y su valor contiene el texto `eniun`.
`[href^="https://"]`	El atributo `href` existe y su valor comienza por `https://`.
`[href$=".pdf"]`	El atributo `href` existe y su valor termina por `.pdf` (es un enlace a un PDF).
`[class~="eniun"]`	El atributo `class` contiene una lista de valores, que contiene `eniun`.
`[lang|="es"]`	El atributo `lang` contiene una lista de valores, donde alguno empieza por `es-`.

Selecciona aquellas imágenes con extensión png:

```
img[src$=png]
```

Se muestran de color azul todos los enlaces que tengan un atributo “class”, independientemente de su valor:

```
a[class] { color: blue; }
```

Se muestran de color azul todos los enlaces que tengan un atributo “class” con el valor “externo”:

```
a[class="externo"] { color: blue; }
```

Se muestran de color azul todos los enlaces que apunten al sitio “http://www.ejemplo.com”.

```
a[href="http://www.ejemplo.com"] { color: blue; }
```

Selecciona todos los elementos de la página cuyo atributo “lang” sea igual a “en”, es decir, todos los elementos en inglés

```
*[lang=en] { ... }
```

Selecciona todos los elementos de la página cuyo atributo “lang” empiece por “es”, es decir, “es”, “es-ES”, “es-AR”, etc.

```
*[lang|="es"] { color : red }
```

Ejemplo

Utilizando selectores de atributos realiza los siguientes ejercicios:

- Selecciona aquellas imágenes con extensión png y añade un borde de 5 píxeles de color rojo.
- Muestra de color verde todos los enlaces que tengan un atributo “class”, independientemente de su valor.
- Muestra de color rosa todos los enlaces que apunten al sitio “https://www.eniun.com”.
- Muestra de color lila todos los enlaces que tengan un atributo “class” con el valor “externo”.

```
img[src$=png]{border: 5px solid red;}
a[class] { color: green; }
a[href="https://www.google.com"] { color: pink; }
a[class~="externo"] { color: purple; }
```

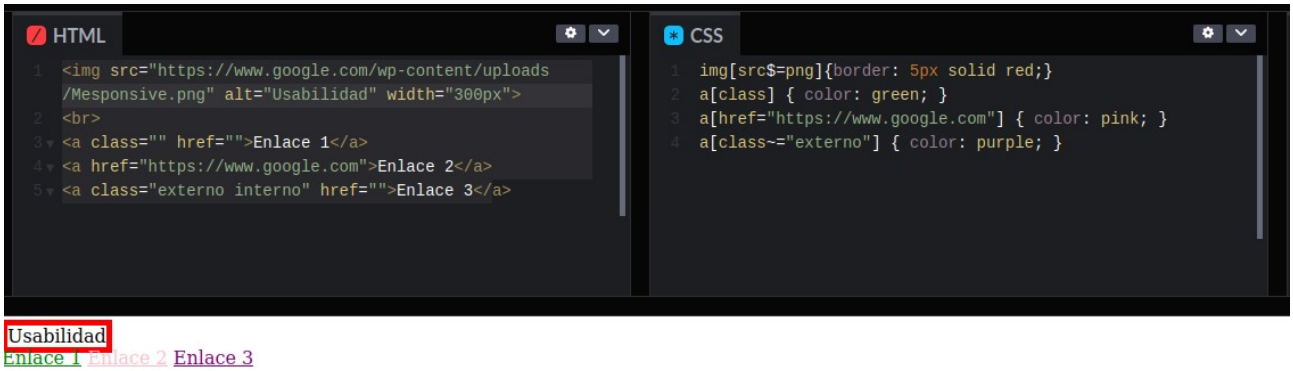


Figure 5: Ejemplo de selector de atributos

8.- Transiciones en CSS

Las transiciones en CSS nos permiten aplicar un **cambio de estilo gradual a los elementos** del documento HTML. Además, nos ofrecen la ventaja de poder **especificar el tiempo** para que se produzca la transformación entre estilos, de esta forma podríamos utilizarlas para dar un **efecto de animación**.

Todos los parámetros para aplicar las transiciones se pueden establecer en una sola línea y también mediante propiedades por separado. Veamos cómo se implementa en una sola línea.

Formato:

```
transition:[propiedad a modificar] [Duración] [Tipo de animación] [Retardo];
```

Ejemplo:

```
.caja1{
   background-color: #C0392B;
   transition: background-color 1s linear;
}
.caja1:hover{
   background-color: #3F51B5;
}
```

Significado de las propiedades:

[Propiedades a modificar]: algunas de las propiedades que podemos modificar utilizando transiciones son las que se muestran en la siguiente lista:

```
all
background-color
border
border-radius
color
top
bottom
left
right
box-shadow
width
height
line-height
margin
opacity
word-spacing
letter-spacing
fill
padding
stroke
text-shadow
vertical-align
visibility
z-index
```

En caso de no indicar ninguna propiedad, la transición se aplicará a todos los cambios que se produzcan

[Duración en segundos]: se debe especificar el número de segundos que va a durar la animación. Por ejemplo: 3s (3 segundos).

[Tipo de animación]: esta propiedad es opcional y sirve para indicar la velocidad de la animación. Algunas de las posibilidades son las siguientes:

- linear: la velocidad de la animación es uniforme en todo el recorrido.
- ease: la velocidad se acelera al inicio, luego se retarda un poco y se acelera al final de nuevo.
- ease-in: la animación empieza lentamente y va aumentando progresivamente.

- ease-out: la animación empieza muy rápida y va descendiendo progresivamente.
- ease-in-out: la animación empieza y acaba lentamente, y es en la parte central del recorrido donde la velocidad es más rápida.

[Retardo]: tiempo (en segundos) que el navegador esperará antes de poner en marcha la animación. Se especifica el número de segundos a esperar seguido de la “s” (ejemplo: 1s).

8.1.1.1.- Ejemplo

Crea un contenedor de 180 píxeles de ancho y de alto, incluye un texto en su interior y aplícale un color de fondo. Realiza las siguientes transiciones cuando el usuario pase por encima del contenedor el puntero del ratón:

- Cambia el color de fondo del contenedor.
- Cambia el tamaño de las letras del contenedor.
- Cambia el color de las letras del contenedor.
- Cambia el valor del interlineado.
- Cambia el valor de la propiedad border-radius del contenedor.
- Cambia el valor de la sombra del contenedor.
- Cambia la opacidad del contenedor.
- Cambia tres propiedades más a tu elección.

```
<div class="container">
  <div class="caja caja1">Background</div>
  <div class="caja caja2">Font-size</div>
  <div class="caja caja3">Color</div>
  <div class="caja caja4">Line-height</div>
  <div class="caja caja5">Border-radius</div>
  <div class="caja caja6">Box-shadow</div>
  <div class="caja caja7">Opacity</div>
  <div class="caja caja8">Border</div>
  <div class="caja caja9">Margin</div>
  <div class="caja caja10">Padding</div>
</div>
```

```
.container{
    position: relative;
}
.caja{
  width: 180px;
  height: 180px;
  background-color: #30B037;
  color: #FFF;
  text-align: center;
  line-height: 180px;
  margin: 15px;
  float: left;
  font-size: 18px;
```

```
  font-family: Arial;
}
.caja1{
  background-color: #C0392B;
  -webkit-transition: background-color 1s linear;
  transition: background-color 1s linear;
}
.caja1:hover{
  background-color: #3F51B5;
}
.caja2{
  background-color: #9B59B6;
  -webkit-transition: font-size 1s ease;
  transition: font-size 1s ease;
}
.caja2:hover{
  font-size: 28px;
}
.caja3{
  background-color: #2980B9;
  -webkit-transition: color 1s linear;
  transition: color 1s linear;
}
.caja3:hover{
  color: yellow;
}
.caja4{
  background-color: #3498DB;
  -webkit-transition: 1s linear;
  transition: 1s linear;
}
.caja4:hover{
  line-height: 300px;
}
.caja5{
  border-radius: 0px;
  background-color: #17A589;
  -webkit-transition: 1s linear;
  transition: 1s linear
}
.caja5:hover{
  border-radius: 20px;
}
.caja6{
  -webkit-transition: 1s linear;
  transition: 1s linear;
  background-color: #F1C40F;
}
.caja6:hover{
  -webkit-box-shadow: 10px 10px 14px 2px rgba(0,0,0,0.47);
  box-shadow: 10px 10px 14px 2px rgba(0,0,0,0.47);
}
.caja7{
  -webkit-transition: 1s linear;
  transition: 1s linear;
  background-color: #E67E22;
```

```
}
.caja7:hover{
  opacity: 0.3;
}
.caja8{
  -webkit-transition: 1s linear;
  transition: 1s linear;
  background-color: #95A5A6;
}
.caja8:hover{
  border-style: solid;
  border-color: #95A5A6;
  border-color: #000;
}
.caja9{
  -webkit-transition: 1s linear;
  transition: 1s linear;
  background-color: #2C3E50;
}
.caja9:hover{
  margin: 80px;
}
.caja10{
  -webkit-transition: 1s linear;
  transition: 1s linear;
  background-color: black;
}
.caja10:hover{
  padding: 30px;
}
```

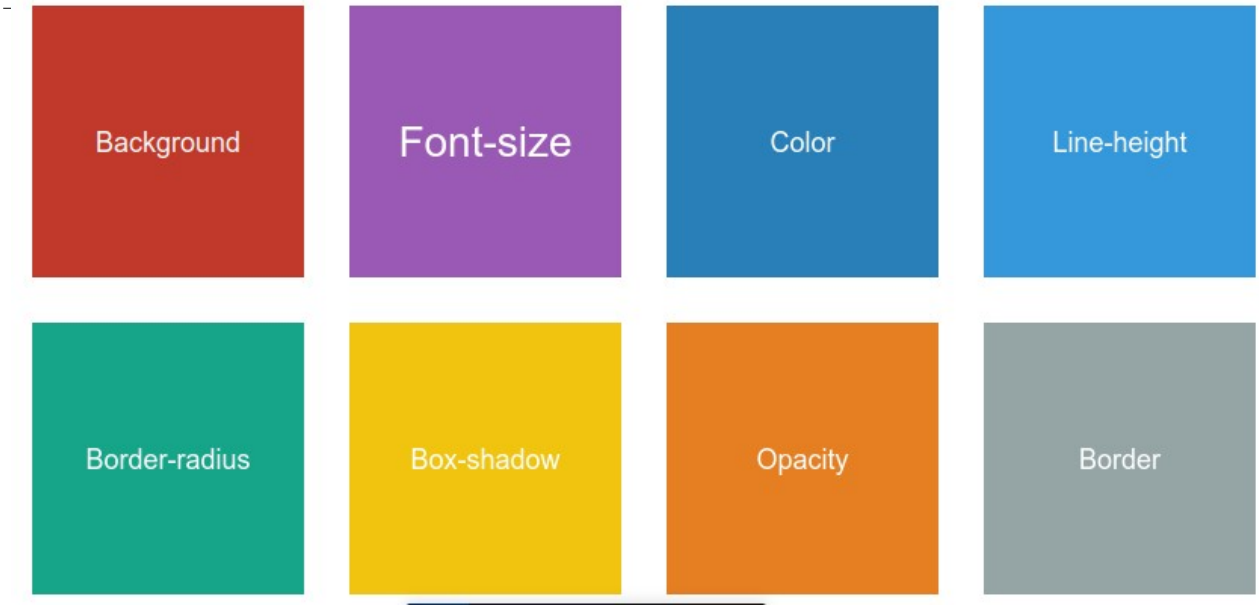


Figure 6: Ejemplo de transiciones

Menú con transiciones:

```
<ul>
<li><a class="menu servicios" href="#" data-title="Servicios"><span>
    <span>Servicios</span>
  </span></a></li>
<li><a class="menu proyectos" href="#" data-title="Proyectos">
  <span>
    <span>Proyectos</span>
  </span>
  </a></li>
  <li><a class="menu recursos" href="#" data-title="Recursos">
  <span>
    <span>Recursos</span>
  </span>
  </a></li>
    <li><a class="menu contacto" href="#" data-title="Contacto">
  <span>
    <span>Contacto</span>
  </span>
  </a></li>
</ul>
```

```
/* Menú horizontal con transiciones */

/* Formato general de cada item */

.menu {
  display: block;
  position: relative;
  height: 76px;
  width: 120px;
  margin: 0;
  overflow: hidden;
}

/* Bordes superiores */

.menu.servicios {
  border-top: #ffc000 0.3em solid;
}

.menu.recursos {
  border-top: #b81800 0.3em solid;
}

.menu.proyectos {
  border-top: #00a53c 0.3em solid;
}

.menu.contacto {
  border-top: #9600b4 0.3em solid;
}

/* Atributos y transición */
```

```
.menu > span {
  display: block;
  position: absolute;
  overflow: hidden;
  left: 0;
  top: 0;
  width: 100%;
  height: 0%;
  transition: 0.5s ease-in-out;
  -webkit-transition: 0.5s ease-in-out;
}

.menu:after, .menu > span > span {
  display: block;
  text-align: center;
  border-radius: 0em;
  padding: 2em 0 1.5em;
}

.menu:after {
  content: attr(data-title);
  width: 100%;
  background: #000;
  color: #fff;
}

.menu > span > span {
  width: 120px;
  color: #fff;
}

.menu.servicios > span > span {
  background: #ffc000;
}

.menu.proyectos > span > span {
  background: #00a53c;
}

.menu.recursos > span > span {
  background: #b81800;
}

.menu.contacto > span > span {
    background: #9600b4;
}

/* Lo que pasa con hover */

.menu:hover > span {
    height: 100%;
}

/* Dando formato a lista */
```

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}

li a {
  display: inline;
  float: left;
  text-decoration: none;
  font-family: sans;
}
```

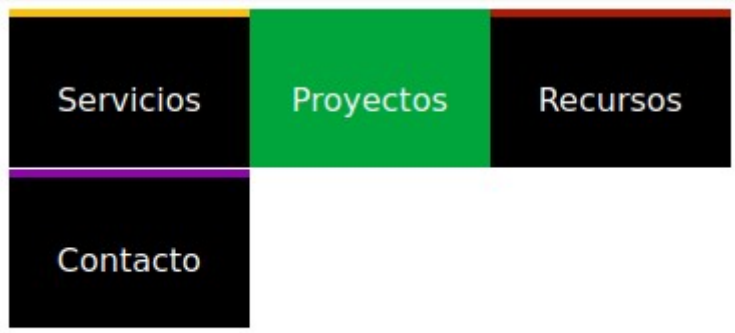


Figure 7: Ejemplo de menú con transiciones

9.- Transformaciones en CSS

Las **transformaciones CSS** nos permiten **rotar, torcer, escalar o desplazar** los elementos de nuestra página web. Este tipo de propiedades de CSS3 son muy interesantes para convertir el lenguaje de hojas de estilo en un sistema capaz de realizar todo tipo de **efectos visuales**. Las dos propiedades que nos sirven para definir las transformaciones son **`transform`** y **`transform-origin`**.

- `Transform-origin` La posición de origen que se utilizará para la transformación es por defecto el lado superior izquierdo del elemento.
- `Transform` La posición de origen para realizar la transformación es el eje central del elemento.

9.1.- Tipos de transformaciones

Las transformaciones que podemos aplicar son las siguientes:

- **Scale**: modifica el tamaño de los elementos. La función scale () se establece con uno o dos valores, que representan la **cantidad de escala que se aplica en cada dirección**: **scale (x) o scale(x,y)**. Se define mediante un **valor numérico** de manera que cuando un valor de coordenadas está fuera del rango [-1, 1], el elemento crece a lo largo de esa dimensión. Cuando está dentro del rango el elemento se encoge.

```
transform: scale(0.5); /* Escala el elemento a la mitad */
```

- **Translate**: cambia la posición del elemento hacia la izquierda, derecha, arriba o abajo. La función translate() se establece con uno o dos valores: **translate(x) o translate(x,y)**. Los valores x e y son los vectores de translación en las coordenadas x e y. Sus valores pueden estar definidos en píxeles, porcentajes,…

```
transform: translate(10px); /* Traslada el elemento 10px hacia la derecha */
```

- **Rotate**: gira o rota los elementos en grados: **rotate(v)**.

```
transform: rotate(45deg); /* Rota el elemento 45 grados */
```

- **Skew**: distorsiona los elementos según el **ángulo en grados**. La función skew() se establece con uno o dos valores: **skew(x) o skew(x,y).**

```
transform: skew(45deg); /* Distorsiona el elemento 45 grados en el eje x */
```

- **Matrix**: mueve o transforma los elementos con precisión de píxel. La función matrix() se establece con seis valores numéricos: matrix(a,b,c,d,x,y). Los dos últimos valores representan la translación y los primeros la transformación lineal.

```
transform: matrix(0.5, 0.1, 0.5, 1, 10, -2);
```

La propiedad transform se usa junto con la propiedad transition vista en la sección anterior para que la transformación pueda tener una transición espaciada en el tiempo:

Los **prefijos** de navegador o prefijos comerciales (vendor prefixes) a un prefijo que se antepone a una regla CSS destinado a que dicha regla sea leída y aplicada exclusivamente por un navegador concreto, por ejemplo el prefijo **-webkit**

```
.caja1{
   background-color: #C0392B;
   -webkit-transition: 1s linear;
   transition: 1s linear;
 }
 .caja1:hover{
   -webkit-transform: scale(.5);
   transform: scale(.5);
 }
```

Ejemplo

Crea un contenedor de 180 píxeles de ancho y de alto, incluye un texto en su interior y aplícale un color de fondo. Realiza las siguientes transformaciones cuando el usuario pase por encima del contenedor el puntero del ratón:

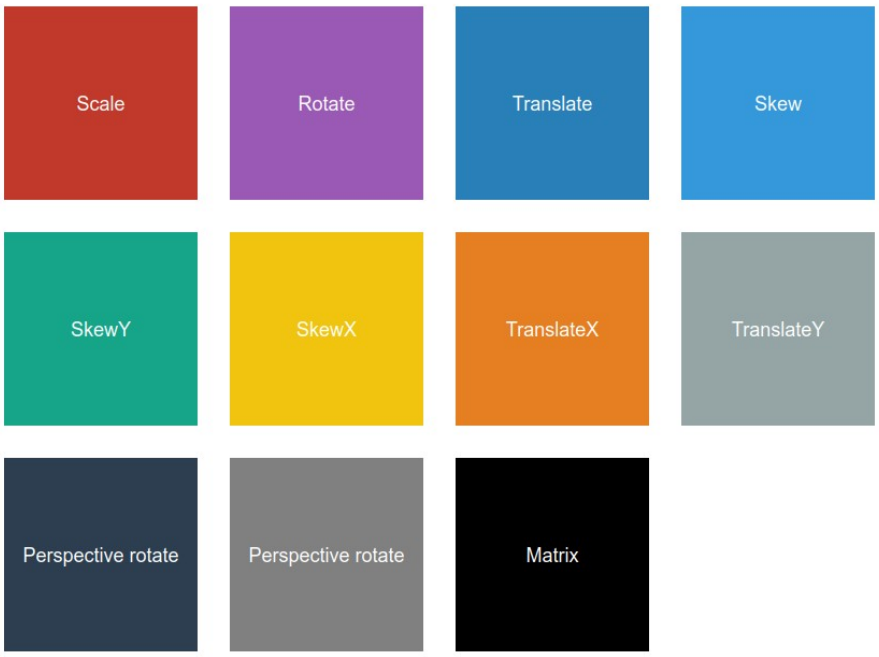
- Modifica la escala del contenedor.
- Modifica la rotación del contenedor.
- Modifica la posición del elemento mediante “translate”.
- Distorsiona el contenedor mediante “skew”.

```
<div class="container">
  <div class="caja caja1">Scale</div>
  <div class="caja caja2">Rotate</div>
  <div class="caja caja3">Translate</div>
  <div class="caja caja4">Skew</div>
  <div class="caja caja5">SkewY</div>
  <div class="caja caja6">SkewX</div>
  <div class="caja caja7">TranslateX</div>
  <div class="caja caja8">TranslateY</div>
  <div class="caja caja9">Perspective rotate</div>
  <div class="caja caja10">Perspective rotate</div>
  <div class="caja caja11">Matrix</div>
</div>
```

```
.container{
  position: relative;
}
.caja{
  width: 180px;
  height: 180px;
  color: #FFF;
  text-align: center;
  line-height: 180px;
  margin: 15px;
  float: left;
  font-size: 18px;
  font-family: Arial;
```

```
}
.caja1{
  background-color: #C0392B;
  -webkit-transition: 1s linear;
  transition: 1s linear;
}
.caja1:hover{
  -webkit-transform: scale(.5);
  transform: scale(.5);
}
.caja2{
  background-color: #9B59B6;
  -webkit-transition: 1s linear;
  transition: 1s linear;
}
.caja2:hover{
  -webkit-transform: rotate(360deg);
  transform: rotate(360deg);
}
.caja3{
  -webkit-transition: 1s linear;
  transition: 1s linear;
  background-color: #2980B9;
}
.caja3:hover{
  -webkit-transform: translate(10%);
  transform: translate(10%);
}
.caja4{
  background-color: #3498DB;
  -webkit-transition: 1s linear;
  transition: 1s linear;
}
.caja4:hover{
  -webkit-transform: skew(10deg);
  transform: skew(10deg);
}
.caja5{
  background-color: #17A589;
  -webkit-transition: 1s linear;
  transition: 1s linear;
}
.caja5:hover{
  -webkit-transform: skewY(10deg);
  transform: skewY(10deg);
}
.caja6{
  background-color: #F1C40F;
  -webkit-transition: 1s linear;
  transition: 1s linear;
}
.caja6:hover{
  -webkit-transform: skewX(-5deg);
  transform: skewX(-5deg);
}
.caja7{
```

```
  -webkit-transition: 1s linear;
  transition: 1s linear;
  background-color: #E67E22;
}
.caja7:hover{
  -webkit-transform: translateX(20px);
  transform: translateX(20px);
}
.caja8{

  -webkit-transition: 1s linear;

  transition: 1s linear;
  background-color: #95A5A6;
}
.caja8:hover{
  -webkit-transform: translateY(20px);
  transform: translateY(20px);
}
.caja9{
  -webkit-transition: 1s linear;
  transition: 1s linear;
  background-color: #2C3E50;
}
.caja9:hover{
  -webkit-transform: perspective(150px) rotateX(45deg);
  transform: perspective(150px) rotateX(45deg);
}
.caja10{
  -webkit-transition: 1s linear;
  transition: 1s linear;
  background-color: grey;
}
.caja10:hover{
  -webkit-transform: perspective(150px) rotateY(45deg);
  transform: perspective(150px) rotateY(45deg);
}
.caja11{
  -webkit-transition: 1s linear;
  transition: 1s linear;
  background-color: black;
}
.caja11:hover{
  -webkit-transform: matrix(0.5, 0.1, 0.5,1,10, -2);
  transform: matrix(0.5, 0.1, 0.5,1,10, -2);
}
```



10.- Propiedad overflow

La propiedad ***overflow*** (desbordamiento o excedente) nos permite controlar el **comportamiento del contenido que se encuentra en una caja o contenedor**. Por lo tanto, mediante esta propiedad podremos especificar si queremos recortar un contenido, mostrar barras de desplazamiento o mostrar el contenido que excede de un elemento a nivel de bloque.

10.1.- Valores de *overflow*

La propiedad *overflow* solo funciona sobre elementos de tipo bloque con una altura definida. Sus valores son los siguientes:

```
overflow: visible|hidden|scroll|auto;
```

- **Overflow: visible (default).** Por defecto la propiedad *overflow* es visible y lo que hace es que los contenidos se salgan del elemento y sean completamente visibles.
- **Overflow: hidden.** Los contenidos que se salen del contenedor padre se ocultan y no se muestran barras de *scroll*. De esta forma se puede controlar el tamaño del elemento y su contenido.

- **Overflow: scroll**. Se muestra una barra de *scroll* (horizontal y vertical) cuando los contenidos no caben en el contenedor o caja.
- **Overflow: auto**. El navegador es el que decide si se muestran las barras de *scroll* o si se extiende el contenedor. En cualquier caso, gracias a este valor nunca se permite que el contenido desborde al contenedor. Este valor es muy interesante ya que si el contenido se sale por un lado (horizontal o vertical), sólo se muestra la barra de *scroll* de ese lado.

Puede interesarte conocer también las propiedades overflow-x y overflow-y.

Ejemplo

Crea un contenedor con un ancho y alto fijo. Incluye en su interior un texto que exceda el espacio definido en el contenedor. Aplica los diferentes valores de la propiedad *overflow* y comenta sus diferencias.

```
.scroll {
  background-color: lightgreen;
  width: 150px;
  height: 150px;
  overflow: scroll;
}

.hidden {
  background-color: lightblue;
  width: 150px;
  height: 150px;
  overflow: hidden;
}

.auto {
  background-color: lightgrey;
  width: 150px;
  height: 150px;
  overflow: auto;
}
.visible {
  background-color: lightyellow;
  width: 150px;
  height: 150px;
  overflow: visible;
}
.auto-img {
  background-color: lightgrey;
  overflow: hidden;
}
.auto-img img{
 float: left;
 max-height: 300px;
}
```

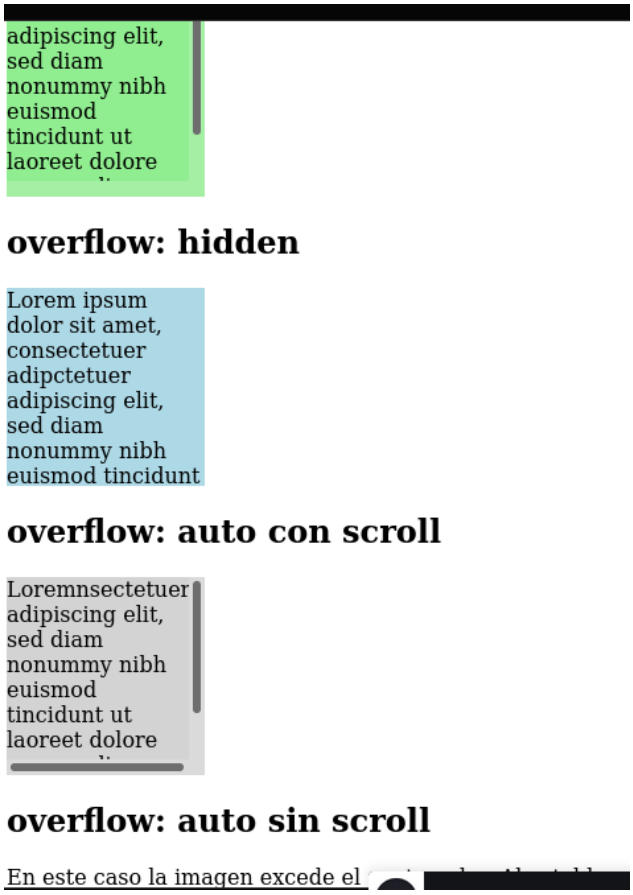


Figure 8: Ejemplo de overflow

11.- Filtros en CSS

Los filtros CSS permiten aplicar efectos visuales a las imágenes. Para aplicar un filtro CSS a una imagen se utiliza la propiedad ***filter***. Aquí tienes un ejemplo de cómo aplicar un filtro de desenfoque a una imagen:

HTML:

```
<img src="imagen.jpg" alt="Imagen" class="filtro-desenfoque">
```

CSS:

```
.filtro-desenfoque {
filter: blur(5px);
}
```

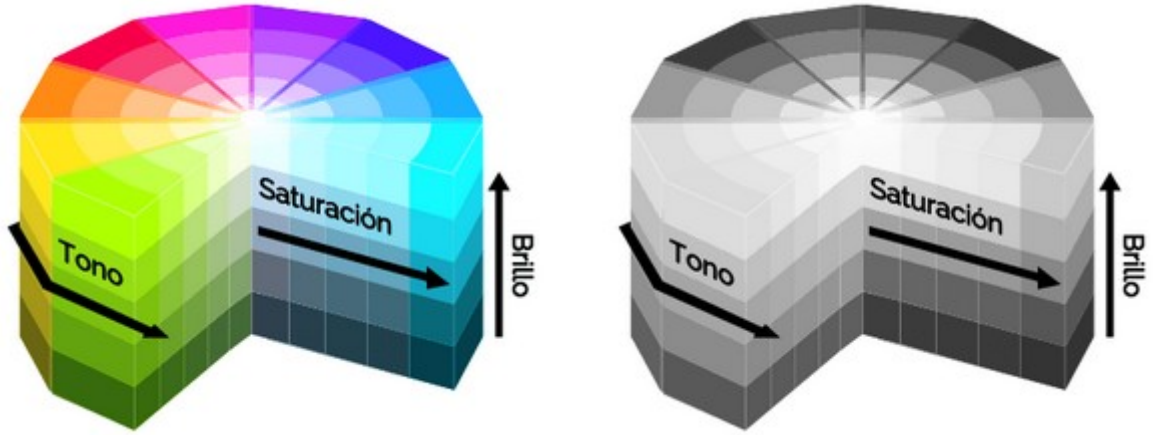
En este ejemplo, la clase filtro-desenfoque se aplica a la etiqueta `<img>` para seleccionar la imagen a la que se le aplicará el filtro. La propiedad *filter* se utiliza con el valor *blur(5px)* para aplicar un desenfoque de 5 píxeles a la imagen.

Puedes ajustar el filtro y su valor según el efecto que desees lograr. También puedes combinar múltiples filtros separándolos por espacios en la propiedad *filter*, por ejemplo: `filter: blur(5px) grayscale(100%);` aplicaría tanto un desenfoque como una escala de grises a la imagen.

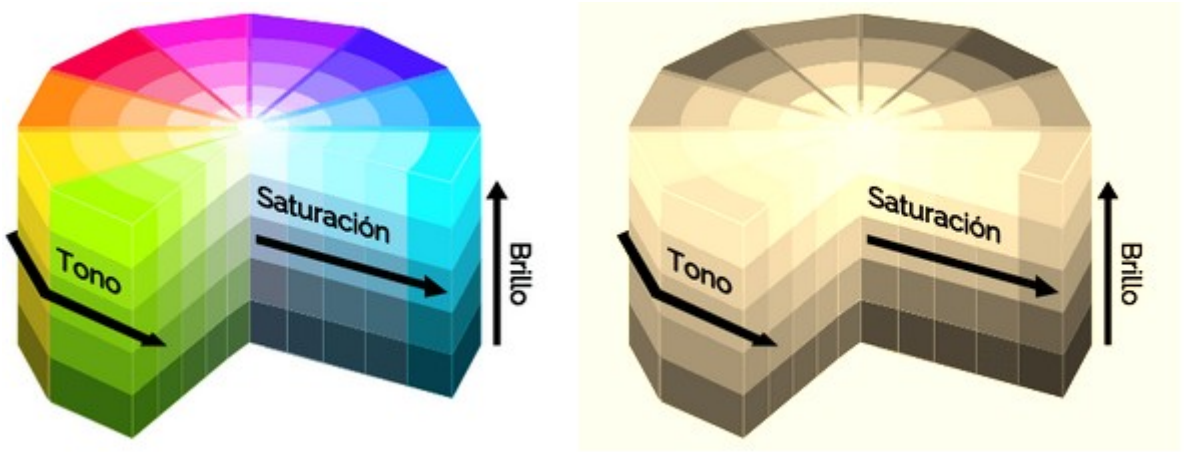
Es importante tener en cuenta que la compatibilidad de los filtros CSS puede variar entre los navegadores.

A continuación se presentan algunos ejemplos de filtros CSS con una breve explicación:

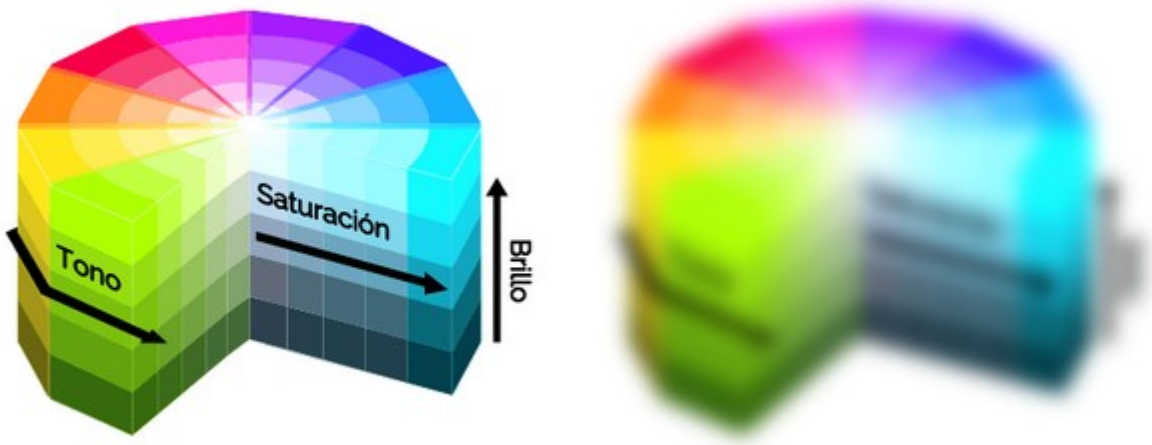
`filter: grayscale(100%);`: Convierte la imagen a escala de grises.



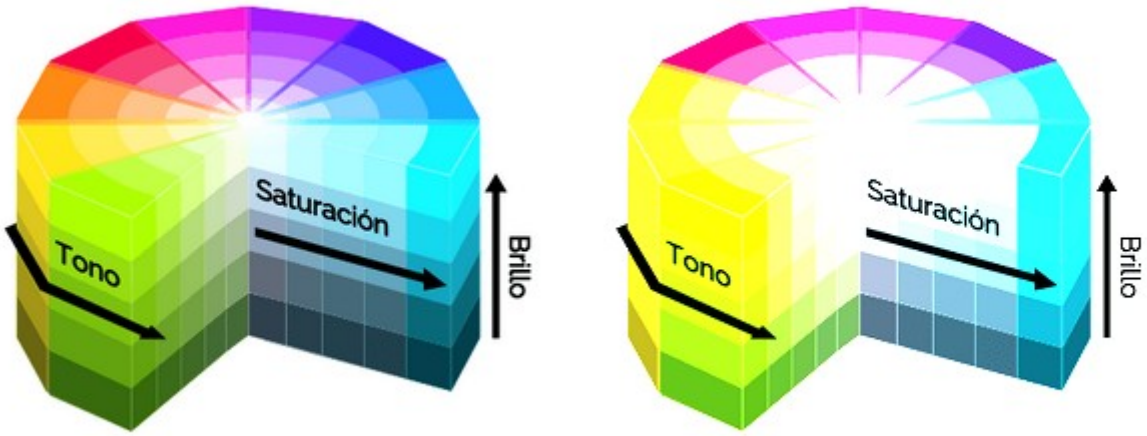
`filter: sepia(100%);`:Aplica un tono sepia a la imagen.



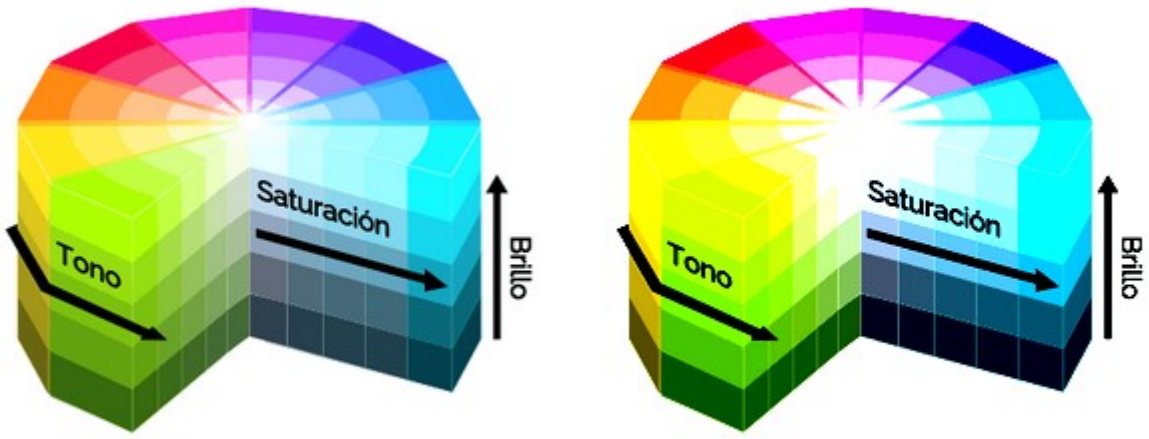
`filter: blur(5px);`: Aplica un efecto de desenfoque a la imagen.



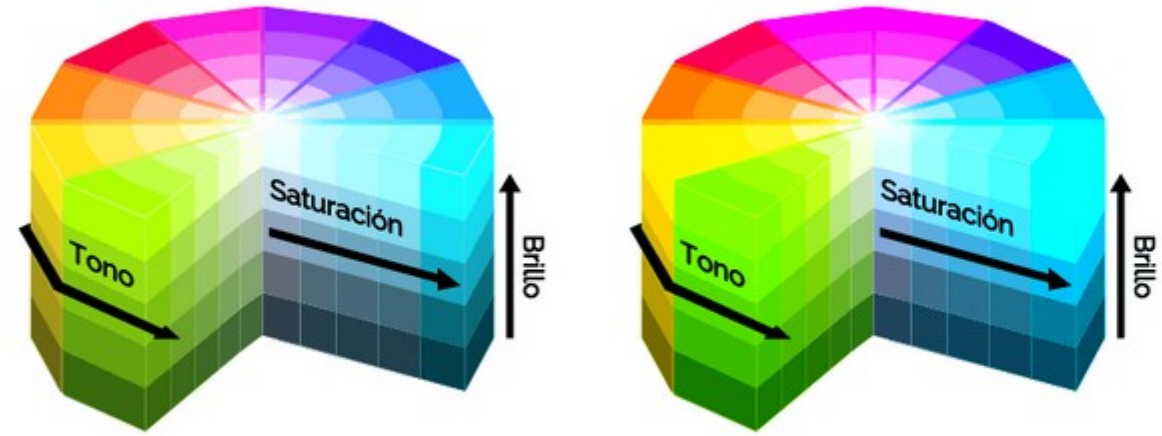
`filter: brightness(190%);`: Ajusta el brillo de la imagen.



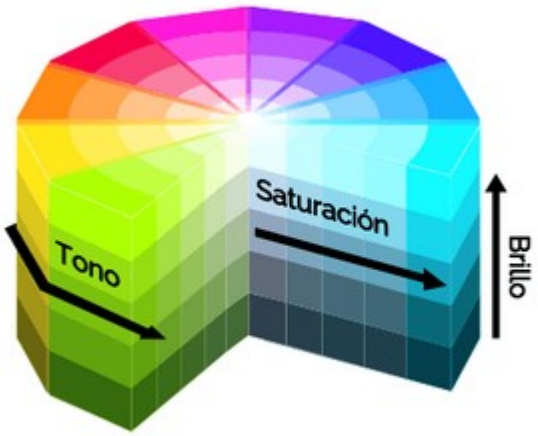
`filter: contrast(200%);`: Ajusta el contraste de la imagen.

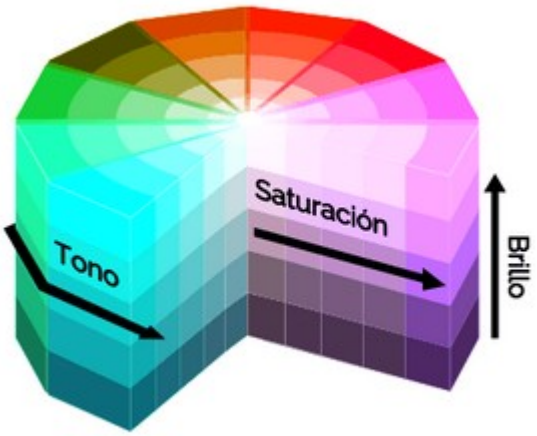


`filter: saturate(200%);`: Aumenta o disminuye la saturación de la imagen.

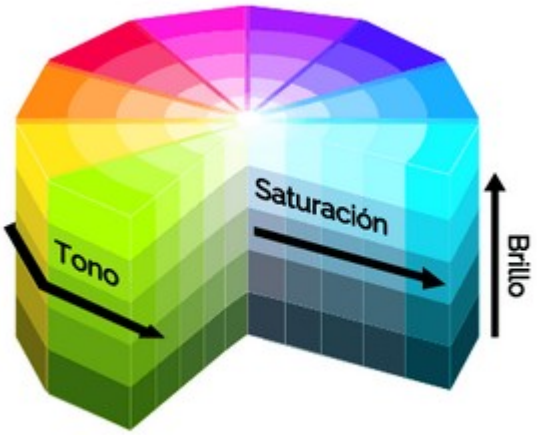


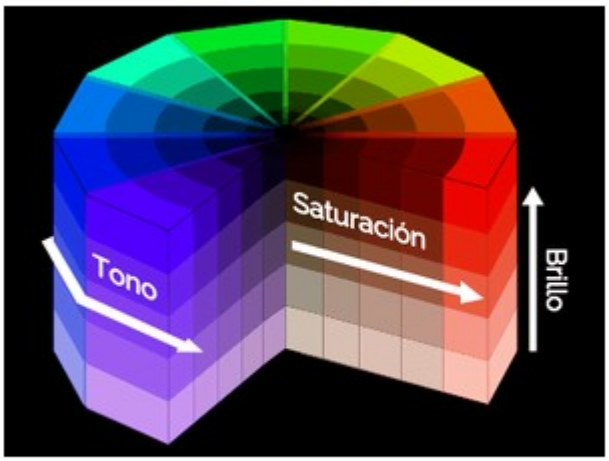
`filter: hue-rotate(90deg);`: Rota el matiz (tono) de la imagen.



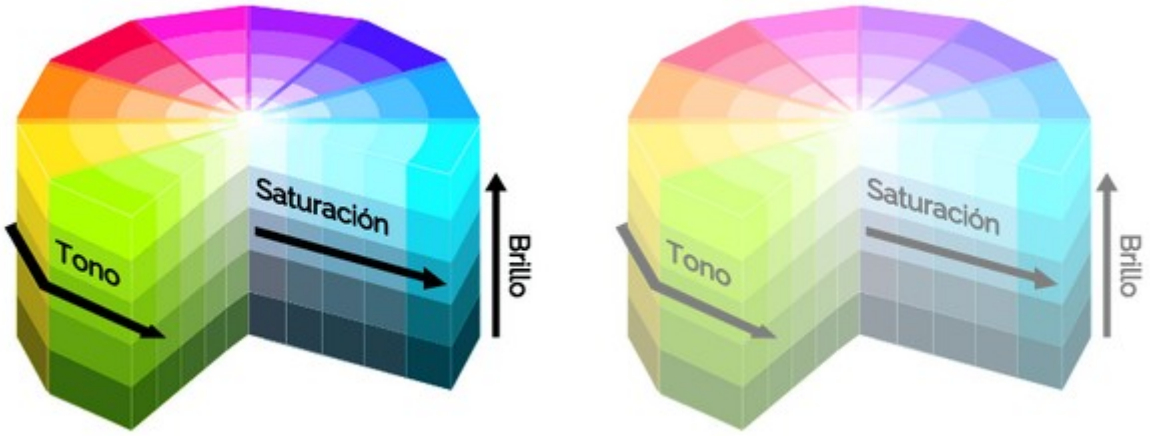


`filter: invert(100%);`: Invierte los colores de la imagen.





`filter: opacity(50%);`: Ajusta la opacidad de la imagen.



12.- Tooltips en CSS

Un es una pequeña ventana emergente que se muestra cuando el usuario coloca el cursor sobre un elemento. Proporciona información adicional o aclaratoria sobre el elemento al que está asociado, sin ocupar mucho espacio en la página. El tooltip suele aparecer cerca del elemento objetivo y desaparece cuando el cursor se mueve fuera de él.

El contenido de un tooltip puede ser texto, imágenes u otros elementos HTML. Se utiliza comúnmente para proporcionar descripciones breves, etiquetas o indicaciones sobre la función o el propósito de un elemento en particular, como enlaces, botones o imágenes. Los tooltips son una forma efectiva de mejorar la usabilidad y la experiencia del usuario al proporcionar información adicional de manera contextual y discreta. Ejemplo:

12.1.- Pasos para crear tooltips en en CSS

Puedes crear un *tooltip* utilizando CSS y HTML siguiendo estos pasos:

1. Crea la estructura HTML para el elemento que tendrá el *tooltip*. Por ejemplo, puedes usar un elemento `<span>` dentro de un elemento con una clase específica.

```
<div class="tooltip-container">
<span class="tooltip-text">Este es un tooltip</span>
Mi elemento
</div>
```

2. Estiliza el contenedor del *tooltip* y el texto del *tooltip* en tu archivo CSS. Puedes ajustar los estilos según tus preferencias.

```
.tooltip-container {
position: relative;
display: inline-block;
}
.tooltip-text {
visibility: hidden;
width: 120px;
background-color: #000;
color: #fff;
text-align: center;
border-radius: 6px;
padding: 5px;
position: absolute;
z-index: 1;
bottom: 125%; /* Posiciona el tooltip encima del elemento */
left: 50%;
transform: translateX(-50%);
opacity: 0;
transition: opacity 0.3s;
}
```

3. Agrega una regla CSS para mostrar el *tooltip* cuando se pasa el cursor sobre el elemento. Puedes utilizar el selector `:hover` para lograrlo.

```
.tooltip-container:hover .tooltip-text {
visibility: visible;
opacity: 1;
}
```

Con estos pasos, has creado un *tooltip* básico con CSS. Puedes personalizar los estilos y añadir efectos adicionales según tus necesidades. Además, también puedes utilizar JavaScript para crear *tooltips* más avanzados con funcionalidades adicionales.

12.2.- Crear un *tooltip* en diferentes posiciones

Para crear un ***tooltip*** que se muestre en diferentes posiciones, como encima, a la derecha, abajo y a la izquierda, puedes ajustar las propiedades de posicionamiento en CSS. Aquí tienes los pasos para lograrlo:

1. Asegúrate de tener la estructura HTML básica del *tooltip*, como se explicó anteriormente.
2. Agrega las siguientes reglas CSS para cada posición del *tooltip*:

- Para el *tooltip* encima del elemento:

```
.tooltip-text {
bottom: 100%; /* Cambia el valor según sea necesario */
left: 50%;
transform: translateX(-50%);
}
```

- Para el *tooltip* a la derecha del elemento:

```
.tooltip-text {
top: 50%;
left: 100%; /* Cambia el valor según sea necesario */
transform: translateY(-50%);
}
```

- Para el *tooltip* abajo del elemento:

```
.tooltip-text {
top: 100%; /* Cambia el valor según sea necesario */
left: 50%;
transform: translateX(-50%);
}
```

- Para el *tooltip* a la izquierda del elemento:

```
.tooltip-text {
top: 50%;
right: 100%; /* Cambia el valor según sea necesario */
transform: translateY(-50%);
}
```

3. Asegúrate de ajustar las propiedades `width`, `background-color`, `color`, `border-radius`, `padding` y cualquier otro estilo que desees para que coincidan con el diseño de tu *tooltip*.

Con estos ajustes, el *tooltip* se posicionará correctamente en la dirección deseada según las reglas CSS proporcionadas.

13.- Flexbox, modelo de caja flexible

En CSS, inicialmente se utilizaba el posicionamiento (static, relative, absolute...), los elementos en línea o en bloque (y derivados) o la propiedad float para realizar maquetaciones, lo que a grandes rasgos no dejaba de ser un sistema de creación de diseños bastante tosco que no encajaba con los retos que tenemos en la actualidad: sistemas de escritorio, dispositivos móviles, múltiples resoluciones, etc...

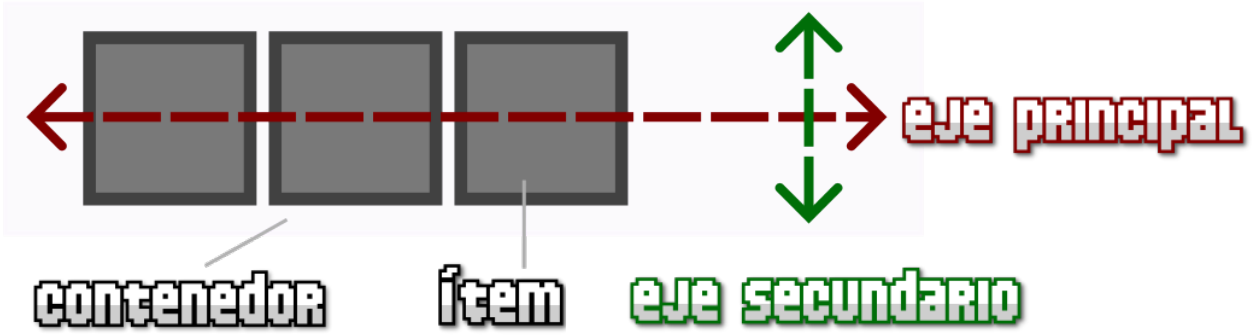
Flex (también llamado flexbox) es un sistema de elementos flexibles que llega con la idea de olvidar estos mecanismos y acostumbrarnos a una mecánica más potente, limpia y personalizable, en la que los elementos HTML se adaptan y colocan automáticamente y es más fácil personalizar los diseños de una página web.

Flex está especialmente diseñado para crear, mediante CSS, estructuras de **una sóla dimensión**.

13.1.- Introducción

13.1.1.- Conceptos

Para empezar a utilizar **flex** lo primero que debemos hacer es conocer algunos de los elementos básicos de este nuevo esquema, que son los siguientes:



- **Contenedor**: Es el elemento padre que tendrá en su interior cada uno de los ítems flexibles. Observa que al contrario que muchas otras estructuras CSS, por norma general, en Flex establecemos las propiedades al elemento padre.
  - **Eje principal**: Los contenedores flexibles tendrán una orientación principal específica. Por defecto, el eje principal del contenedor flex es en **horizontal** (*en fila*).
  - **Eje secundario**: De la misma forma, los contenedores flexibles tendrán una orientación secundaria, perpendicular a la principal. Si la principal es en horizontal, la secundaria será en **vertical** (*y viceversa*).
- **Ítem**: Cada uno de los **hijos** que tendrá el contenedor en su interior.

13.1.2.- Modalidades de flex

Una vez tenemos claro esto, imaginemos el siguiente escenario:

```
<div class="container"> <!-- Flex container -->
  <div class="item item-1">1</div> <!-- Flex items -->
  <div class="item item-2">2</div>
  <div class="item item-3">3</div>
</div>
```

Para activar el modo **flex**, utilizaremos sobre el elemento contenedor la propiedad display, y especificaremos el valor `flex` o `inline-flex` (*dependiendo de como queramos que se comporte el contenedor*):

Tipo de elemento	Descripción
`inline-flex`	Establece un contenedor en línea, similar a `inline-block` (ocupa solo el contenido).
`flex`	Establece un contenedor en bloque, similar a `block` (ocupa todo el ancho del padre).

Por defecto, y sólo con esto, observaremos que los elementos se disponen todos sobre una misma línea. Esto ocurre porque estamos utilizando el modo **flex** y estaremos trabajando con ítems flexibles básicos, garantizando que no se desbordarán ni mostrarán los problemas que, por ejemplo, tienen los porcentajes sobre elementos que no utilizan flex.

13.1.3.- Dirección de los ejes

Existen dos propiedades principales para manipular la dirección y comportamiento de los ítems a lo largo del eje principal del contenedor. Son las que veremos a continuación:

Propiedad	Valor	Significado
`flex-direction`	**row** | row-reverse | column | column-reverse	Cambia la orientación del eje principal.

Mediante la propiedad `flex-direction` podemos modificar la dirección del **eje principal** del contenedor para que se oriente en horizontal (*valor por defecto*) o en vertical. Además, también podemos incluir el sufijo `-reverse` para indicar que coloque los ítems en orden inverso.

Valor	Descripción
`row`	Establece la dirección del eje principal en **horizontal**.
`row-reverse`	Establece la dirección del eje principal en **horizontal invertido**.
`column`	Establece la dirección del eje principal en **vertical**.
`column-reverse`	Establece la dirección del eje principal en **vertical invertido**.

Esto nos permite tener un control muy alto sobre el orden de los elementos en una página. Veamos la aplicación de estas propiedades sobre el ejemplo anterior, para modificar el flujo del eje principal del contenedor:

```
.container {
```

```
  display: flex;
  flex-direction: column;
  background: steelblue;
}

.item {
  background: grey;
}
```

flex-direction: row;

flex-direction: row-reverse;

flex-direction: column;

flex-direction: column-reverse;

13.1.4.- Contenedor flex multilínea

En general, **flex** se suele utilizar para estructuras de una sola dimensión, es decir, contenedores que sólo van en una dirección. Sin embargo, existe una propiedad denominada `flex-wrap` con la que podemos especificar un comportamiento especial del contenedor.

Por defecto, si un elemento no cabe dentro de nuestro contenedor flex, los elementos se harán más pequeños (*son flexibles*) para ajustarlos al contenedor. Este es el comportamiento por defecto de un contenedor **flex**. Sin embargo, con la propiedad `flex-wrap` podemos cambiar este comportamiento y permitir que nuestro contenedor flex se desborde, convirtiéndose en un **contenedor flex multilínea**.

Propiedad	Valor	Significado
`flex-wrap`	**nowrap** | wrap | wrap-reverse	Evita o permite el desbordamiento (multilinea).

Los valores que puede tomar esta propiedad, son las siguientes:

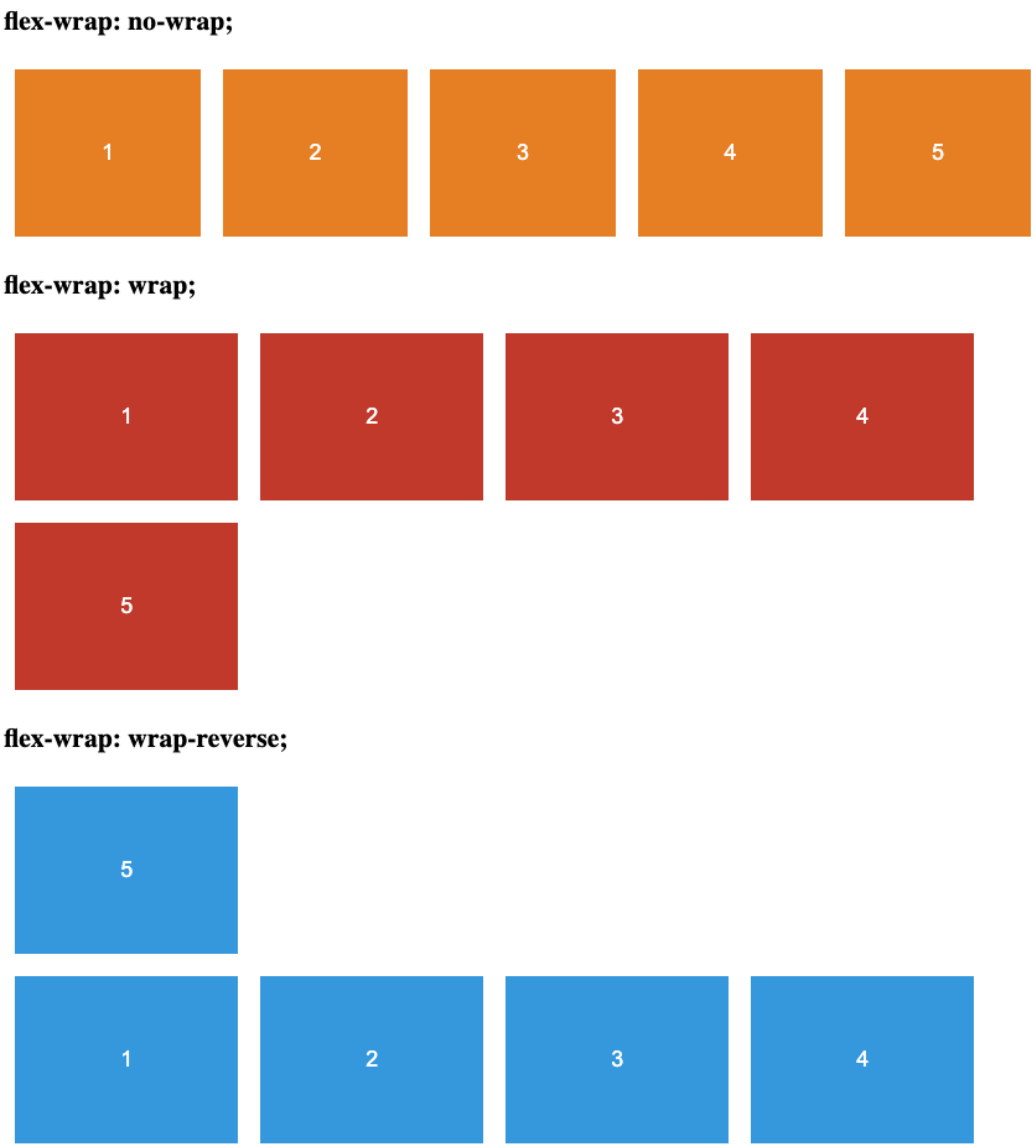
Valor	Descripción
nowrap	Los ítems se ajustan para ocupar el tamaño del contenedor (no permite desbordamiento en múltiples líneas). Los elementos no pasan a la siguiente fila y se reduce su anchura para mostrarlos. Este es el valor por defecto.
wrap	Establece los ítems en modo multilínea (permite que se desborde el contenedor). Los elementos pasan a la siguiente fila y conservan su anchura
wrap-reverse	Establece los ítems en modo multilínea, pero en dirección inversa. los elementos pasan a la siguiente fila, pero en sentido inverso al de su declaración

Teniendo en cuenta estos valores de la propiedad `flex-wrap`, podemos conseguir cosas como la siguiente:

```
.container {
  display: flex;
  flex-wrap: wrap;
  background: steelblue;
  width: 200px;
}

.item {
  background: grey;
  width: 50%;
}
```

En el caso de especificar **nowrap** (*u omitir la propiedad* `flex-wrap`) en el contenedor, los 3 ítems se mostrarían en una misma línea del contenedor (*que es el comportamiento por defecto*). En ese caso, si tenemos 3 ítems, cada uno debería tener aproximadamente `66px` de ancho. Un tamaño de `100px` por ítem, sumaría un total de `300px`, que no cabrían en el contenedor de `200px`, por lo que **flex** reajusta los ítems flexibles para que quepan todos en la misma línea, manteniendo las mismas proporciones.

Sin embargo, si especificamos `wrap` en la propiedad `flex-wrap`, lo que permitimos es que el contenedor se pueda desbordar, pasando a ser un contenedor **multilínea**, que mostraría los **ítems** que quepan en la primera linea y el resto en las líneas siguientes. El valor `wrap-reverse` haría exactamente lo mismo, pero con el orden inverso.



13.1.5.- Atajo: Dirección de los ejes

Recuerda que existe una propiedad de atajo (short-hand) llamada `flex-flow`, con la que podemos resumir los valores de las propiedades `flex-direction` y `flex-wrap`, especificándolas en una sola propiedad y ahorrándonos utilizar las propiedades concretas:

```
.container {
  /* flex-flow: <flex-direction> <flex-wrap>; */
  flex-flow: row wrap;
}
```

13.1.6.- Huecos (gaps)

Existen dos propiedades de flexbox que han surgido recientemente: `row-gap` y `column-gap`. Dichas propiedades, permiten establecer el tamaño de un «hueco» entre ítems desde el elemento padre contenedor, y que eliminan la necesidad de estar utilizando `padding` o `margin` en los elementos hijos, con las complicaciones que ello suele conllevar:

Propiedad	Valor	Descripción
`row-gap`	**normal** |	Espacio entre filas (sólo funciona con `flex-direction: column`)
`column-gap`	**normal** |	Espacio entre columnas (sólo funciona con `flex-direction: row`)

Ten en cuenta que, como **flex** es un sistema para diseños de una sola dimensión, sólo una de las dos propiedades tendrá efecto. Si la propiedad `flex-direction` está establecida en `column`, podrás utilizar `row-gap`, y en el caso de que la propiedad `flex-direction` se encuentre en `row`, podrás utilizar el `column-gap`.

Eso sí, es posible usar ambas si tenemos la propiedad `flex-wrap` definida a `wrap` y, por lo tanto, disponemos de multicolumnas flexbox, ya que en este caso si podemos separar elementos por filas y por columnas.

> Ten en cuenta que los huecos **sólo se aplican entre elementos**, y no entre un elemento hijo y su contenedor padre.

13.1.7.- Atajo: Huecos

En **Flex CSS** existe una propiedad de atajo para los huecos, denominada `gap`. Con esta propiedad podemos indicar de una sola vez valores para las propiedades `row-gap` y `column-gap`, de forma que escribimos menos y es más cómodo en ciertas situaciones:

Propiedad	Valor	Descripción
gap	**0**	Aplica el tamaño indicado para el hueco en ambos ejes.
gap	**0 0**	Aplica los tamaños indicados para el hueco del eje X y el eje Y.

Como se puede ver, por defecto, el tamaño de los huecos es de **0**, sin embargo, podemos utilizar tanto las propiedades individuales como la propiedad de atajo `gap` para modificar estos tamaños.

A continuación, podemos ver un ejemplo de su utilización en este fragmento de código:

```
.container {
  /* 2 parámetros: <row> <column> */
  gap: 4px 8px;
  /* Equivalente a */
  row-gap: 4px;
  column-gap: 8px;

  /* 1 parámetro: usa el mismo para ambos */
  gap: 4px;
  /* Equivalente a */
  row-gap: 4px;
  column-gap: 4px;
}
```

Recuerda que esta característica es una de las últimas en implementarse en navegadores, por lo que en algunos navegadores muy antiguos podría darte problemas. Sin embargo, el soporte actual es bastante bueno

13.2.- Alinear elementos

Cuestiones habituales en el mundo de CSS suelen ser **«Cómo centrar con Flex»**, **«cómo alinear verticalmente»** o **«cómo alinear horizontalmente»**. A continuación, vamos a dar un repaso a las propiedades de alineación de elementos, para que no se te vuelva a resistir como distribuir los elementos de un contenedor **Flex**.

13.2.1.- Propiedades de alineación

Ahora, tras el tema de Introducción a Flex, tenemos un control básico de un contenedor con ítems flexibles. Pero para alinear correctamente, necesitamos conocer las propiedades existentes dentro de flex para disponer los ítems dependiendo de nuestro objetivo.

Vamos a echar un vistazo a las siguientes propiedades, donde algunas actuan en el **eje principal** (*recordemos que por defecto es el horizontal*), mientras que otras actuan en el **eje secundario** (*por defecto, el eje vertical*):

Propiedad	Valor	Actúa en eje
`justify-content`	**start** | end | center | space-between | space-around | space-evenly	
`align-items`	start | end | center | **stretch** | baseline	
`align-content`	start | end | center | space-between | space-around | space-evenly | **stretch**	

De esta pequeña lista, hay que centrarse en la primera y la segunda propiedad, ya que son las principales. La última propiedad, `align-content` solo tiene efecto si tenemos un contenedor **flex multilinea**.

> Es posible que a veces hayas encontrado valores como `flex-start` o `flex-end`, en lugar de `start` o `end`. Antiguamente, las palabras claves de estas propiedades tenían el prefijo `flex-`, pero aunque funcionen, se recomienda usar la versión sin el prefijo.

Antes, un pequeño resumen:

- `justify-content`: Se utiliza para alinear los ítems del **eje principal** (*por defecto, el horizontal*).
- `align-items`: Usada para alinear los ítems del **eje secundario** (*por defecto, el vertical*).
- `align-content`: Se utiliza para alinear el contenido del **eje secundario** entre líneas (*sólo en contenedor multilinea*).

13.2.2.- Alineación de elementos

13.2.2.1.- La propiedad `justify-content`

La primera propiedad, `justify-content`, sirve para colocar los ítems de un contenedor mediante una disposición concreta a lo largo del **eje principal** (*por defecto, en horizontal*). Los valores que puede tomar esta propiedad son los siguientes:

Valor	Descripción
start	Agrupa los ítems al **inicio** del eje principal.
`end`	Agrupa los ítems al **final** del eje principal.
`center`	Agrupa los ítems al **centro** del eje principal.
`space-between`	Distribuye los ítems dejando espacio **entre ellos**.
`space-around`	Distribuye los ítems dejando espacio **alrededor de ellos**.
`space-evenly`	Distribuye como `space-around`, pero con un **espacio exactamente**

Valor	Descripción
	igual alrededor de ellos.

Con cada uno de estos valores, modificaremos la disposición de los ítems del contenedor donde se aplica, pasando a colocarse como se ve en el ejemplo interactivo siguiente

justify-content: flex-start;

justify-content: flex-end;

justify-content: center;

justify-content: space-between;

justify-content: space-around;

(*nótense los números para observar el orden de cada ítem*):

13.2.2.2.- La propiedad `align-items`

Existe otra propiedad importante denominada `align-items`. Se encarga de alinear los ítems en el **eje secundario** del contenedor. Hay que tener cuidado de no confundir `align-items` con `align-content`, puesto que el segundo actúa sobre cada una de las líneas de un contenedor multilinea (*no tiene efecto si no usamos `flex-wrap`*), mientras que `align-items` lo hace sobre la única línea que tiene un contenedor flex sin wrap.

Los valores que puede tomar `align-items` son los siguientes:

Valor	Descripción
`start`	Alinea los ítems al **inicio** del eje secundario.
`end`	Alinea los ítems al **final** del eje secundario.
`center`	Alinea los ítems al **centro** del eje secundario.
stretch	Alinea los ítems **estirándolos** de modo que cubran desde el inicio hasta el final del contenedor.
`baseline`	Alinea los ítems en el contenedor según la **base** del contenido de los ítems del contenedor.

align-items: stretch;

align-items: flex-start;

align-items: flex-end;

align-items: center;

align-items: baseline;

13.2.3.- Alineación multilínea

13.2.3.1.- La propiedad `align-content`

Una vez entendidos los casos anteriores, debemos atender a la propiedad `align-content`, que es un caso particular de `align-items`. Nos servirá cuando estemos tratando con un **contenedor flex multilinea** creado mediante `flex-wrap`. Los contenedores multilinea son un tipo de contenedor en el que, cuando los ítems no caben en el ancho disponible, el eje principal se divide en múltiples líneas.

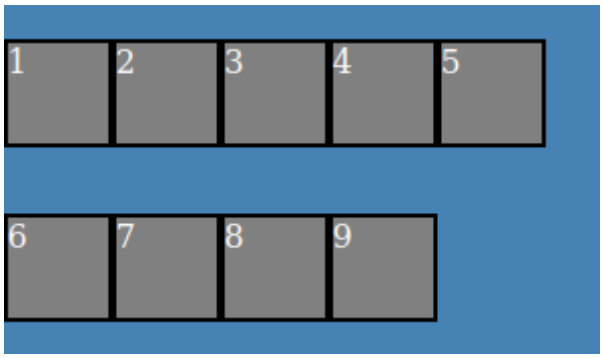
De esta forma, `align-content` servirá para alinear cada una de las líneas del contenedor multilinea. Los valores que puede tomar la propiedad `align-content` son los siguientes:

Valor	Descripción
`start`	Agrupa los ítems al **inicio** del eje principal.
`end`	Agrupa los ítems al **final** del eje principal.
`center`	Agrupa los ítems al **centro** del eje principal.
`space-between`	Distribuye los ítems desde el **inicio hasta el final**.
`space-around`	Distribuye los ítems dejando el mismo espacio **a los lados** de cada uno.
stretch	**Estira** los ítems para ocupar de forma equitativa todo el espacio.

Recuerda que estaremos modificando la disposición en vertical, salvo que estemos usando `flex-direction: column`, en cuyo caso, el eje principal sería horizontal.

En el ejemplo siguiente, veremos que al indicar un contenedor de **200 píxels de alto** con ítems de **50px** de alto y un **flex-wrap** establecido para tener contenedores multilinea, podemos utilizar la propiedad `align-content` para alinear los ítems de forma vertical de modo que se queden en la zona inferior del contenedor:

```
.container {
  background: #CCC;
  display: flex;
  width: 200px;
  height: 200px;

  flex-wrap: wrap;
  align-content: end;
}

.item {
  background: #777;
  width: 50%;
  height: 50px;
}
```



13.2.4.- Alineaciones específicas

13.2.4.1.- La propiedad `align-self`

Por otro lado, la propiedad `align-self` actúa exactamente igual que `align-items`, sin embargo es la primera propiedad de flex que vemos que se utiliza sobre un **ítem hijo específico** y no sobre el elemento padre contenedor. Salvo por este detalle, funciona exactamente igual que `align-items`.

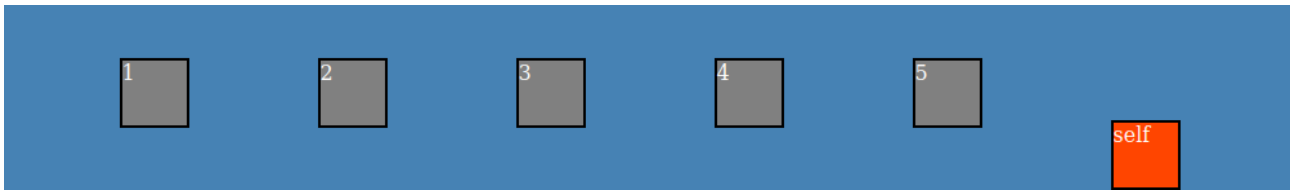
Propiedad	Valores	Actúa en eje
`align-self`	**auto** | start | end | center | stretch | baseline	

Gracias a ese detalle, `align-self` nos permite cambiar el comportamiento de `align-items` y sobreescribirlo con **comportamientos específicos** para ítems concretos que no queremos que se comporten igual que el resto.

La propiedad `align-self` puede tomar los siguientes valores:

Valor	Descripción
`start`	Alinea los ítems al **inicio** del contenedor.
`end`	Alinea los ítems al **final** del contenedor.
`center`	Alinea los ítems al **centro** del contenedor.
`stretch`	Alinea los ítems **estirándolos** al tamaño del contenedor.
`baseline`	Alinea los ítems en el contenedor según la **base** de los ítems.
auto	Hereda el valor de **align-items** del padre (si no se ha definido, es **stretch**).

Como vemos, se comporta igual que la propiedad `align-items`, salvo con la adición del valor **auto**, que si se especifica, el navegador le asignará el valor de la propiedad `align-items` del contenedor padre. En caso de no existir, el valor por defecto será **stretch**.



13.2.5.- Atajo: Alineaciones

Existe una propiedad de atajo con la que se pueden establecer los valores de las propiedades `align-content` y `justify-content` de una sola vez. Dicha propiedad es `place-content` y funciona de la siguiente forma:

```
.container {
  display: flex;

  /* 2 parámetros */
  place-content: start end;
  /* Equivalente a... */
  align-content: start;
  justify-content: end;

  /* 1 parámetro */
  place-content: start;
  /* Equivalente a... */
  align-content: start;
  justify-content: start;
}
```

En la siguiente charla, retransmitida por mi canal de Twitch, explicamos como crear una baraja de cartas de Poker, utilizando exclusivamente sistemas de maquetación de CSS, como **Flex**, Grid, posicionamiento u otros detalles.

13.2.6.- Orden de los elementos

Por último, y quizás una de las propiedades más interesantes, es `order`. Se trata de una propiedad mediante la cual podemos modificar y establecer un orden de los elementos mediante números:

Propiedad	Valor	Descripción
`order`	**0** |	Número (peso) que indica el orden de aparición de los ítems.

Por defecto, todos los elementos hijos de un contenedor flex tienen establecido un `order` por defecto al valor `0`, aunque no se especifique de forma explícita. Si indicamos una propiedad `order` con un valor numérico diferente, irá recolocando los ítems según dicho

número, colocando antes los elementos con un número `order` más pequeño (*incluso valores negativos*) y después los elementos con números más altos.

Esta característica se vuelve muy potente en combinación con las estrategias de `flex-direction` o incluso Media Queries, ya que podríamos recolocar fácilmente elementos que son contenedores de secciones de la web, de modo que una zona concreta, por ejemplo, un submenú, aparezca arriba en escritorio y abajo en móvil.

13.3.- Otras propiedades

Las propiedades de *flexbox* que hemos visto son las más comunes y utilizadas para controlar el comportamiento de los elementos flex. Sin embargo, hay más propiedades disponibles que te permiten tener un mayor control sobre el diseño de tus elementos flex. Algunas de estas propiedades adicionales incluyen `order`, `flex-grow`, `flex-shrink`, y `flex-basis`, entre otras. Veamos un resumen de algunas de las propiedades más utilizadas.

Propiedad	Descripción	Valores Ejemplo
display	Define el contenedor como un flex container	`flex`
flex-wrap	Controla si los elementos flex deben envolverse o no	`nowrap` (por defecto), `wrap`, `wrap-reverse`
justify-content	Alinea los elementos a lo largo del eje principal	`flex-start` (por defecto), `flex-end`, `center`, `space-between`, `space-around`, `space-evenly`
align-items	Alinea los elementos a lo largo del eje secundario	`stretch` (por defecto), `flex-start`, `flex-end`, `center`, `baseline`
flex-direction	Especifica la dirección principal de los elementos	`row` (por defecto), `row-reverse`, `column`, `column-reverse`
flex-grow	Controla cómo los elementos flex se expanden	Número (valor relativo)

Propiedad	Descripción	Valores Ejemplo
flex-shrink	Controla cómo los elementos flex se encogen	Número (valor relativo)
flex-basis	Establece el tamaño inicial de los elementos flex	Valor, `auto` (por defecto)
order	Controla el orden de los elementos flex dentro del contenedor	Número (entero)
align-self	Alinea un elemento individual a lo largo del eje secundario	`auto` (por defecto), `flex-start`, `flex-end`, `center`, `baseline`, `stretch`

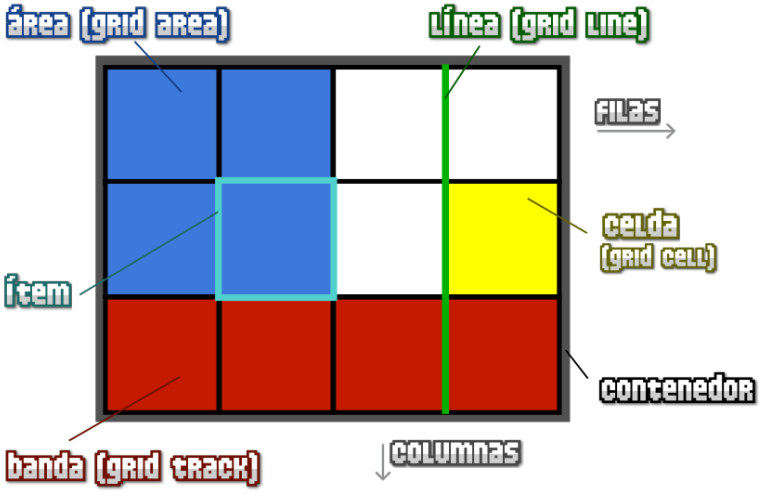
14.- CSS Grid

CSS Grid *nos permite maquetar contenido ajustándolo a cuadrículas o rejillas (grids) totalmente configurables mediante estilos CSS.*

Mediante CSS Grid podemos dividir la página en una rejilla a partir de la cual se pueden posicionar los diferentes elementos de manera muy sencilla. Gracias a los sistemas de maquetación de CSS Grid y CSS Flexbox se pueden crear estructuras con menos código y de una forma más fácil que con los métodos tradicionales.

La diferencia básica entre CSS Grid y CSS Flexbox es que Flexbox se creó para diseños de una dimensión, en una fila o una columna. En cambio CSS Grid Layout se pensó para el diseño bidimensional, en varias filas y columnas al mismo tiempo.

Para crear diseños basados en Grid CSS necesitaremos tener en cuenta una serie de conceptos que utilizaremos a partir de ahora y que definiremos a continuación:



- **Contenedor**: El elemento padre contenedor que definirá la cuadrícula o rejilla.
- **Ítem**: Cada uno de los hijos que contiene la cuadrícula (elemento contenedor).
- **Celda** (grid cell): Cada uno de los cuadritos (unidad mínima) de la cuadrícula.
- **Área** (grid area): Región o conjunto de celdas de la cuadrícula.
- **Banda** (grid track): Banda horizontal o vertical de celdas de la cuadrícula.
- **Línea** (grid line): Separador horizontal o vertical de las celdas de la cuadrícula.

Para utilizar cuadriculas **Grid CSS**, trabajaremos bajo el siguiente escenario:

```
<div class="grid"> <!-- contenedor -->
  <div class="item item-1">Item 1</div> <!-- cada uno de los ítems del grid -->
  <div class="item item-2">Item 2</div>
  <div class="item item-3">Item 3</div>
  <div class="item item-4">Item 4</div>
</div>
```

Para utilizar **CSS Grid** definiremos un contenedor padre y en su interior los ítems que se necesiten crear. Además, se precisan definir una serie de propiedades, tanto para el contenedor padre como para las rejillas que se definen en su interior. Veamos en detalle las propiedades más importantes.

14.1.- Propiedad `display`

Para crear la cuadrícula **grid** hay que definir sobre el elemento contenedor la propiedad **`display`** y especificar el valor **grid** o **inline-grid**.

```
<!DOCTYPE html>
<html>
<head>
<style>
.contenedor{
display: grid;
}
</style>
</head>
<body>
<div class="contenedor"> <!-- contenedor padre-->
<div class="rejilla">Item 1</div> <!-- Ítems del grid -->
<div class="rejilla">Item 2</div>
<div class="rejilla">Item 3</div>
<div class="rejilla">Item 4</div>
</div>
</body>
</html>
```

Los valores **inline-grid** y **grid** indican cómo se comporta el contenedor con respecto al contenido exterior. Con el valor inline-grid el contenedor aparece en línea con respecto al contenido exterior. Con el valor grid, el contenedor aparece en bloque con respecto al contenido exterior.

Valor	Descripción
inline-grid	Cuadrícula en línea con respecto al contenido exterior. Es decir, permite que la cuadrícula aparezca encima/debajo del contenido exterior (en bloque)
grid	Cuadrícula en bloque con respecto al contenido exterior. Es decir, permite que la cuadrícula aparezca a la izquierda/derecha (*en línea*) del contenido exterior (*ojo, la cuadrícula entera, no cada uno de sus ítems*)

Una vez elegido uno de estos dos valores, y establecida la propiedad `display` al elemento contenedor, hay varias formas de configurar nuestra cuadrícula grid. Al igual que con **Flex**, muchas de las propiedades se aplican al contenedor padre, sin embargo, existen algunas que se aplican sobre los elementos hijos. Las iremos viendo todas detalladamente.

14.2.- Definir filas y columnas.

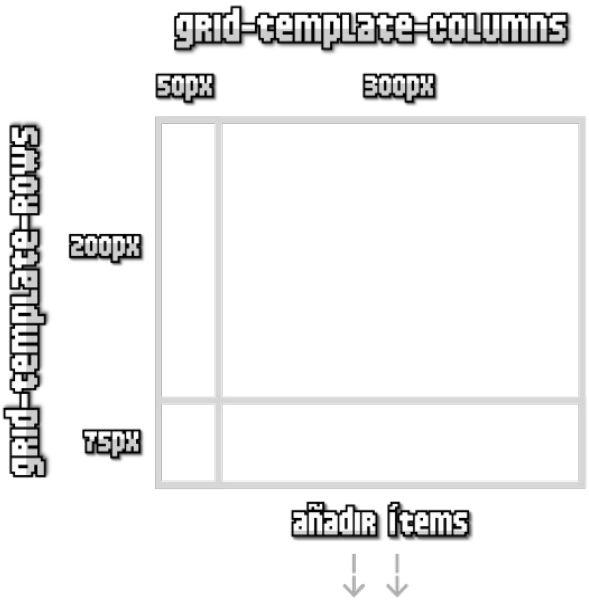
En **Grid CSS**, la forma principal de definir una cuadrícula es indicar el tamaño de sus filas y sus columnas de forma explícita. Para ello, sólo tenemos que usar las propiedades CSS `grid-template-columns` y `grid-template-rows`:

Propiedad	Valor	Descripción
`grid-template-columns`	[*col1*] [*col2*] …	Tamaño de cada columna
`grid-template-rows`	[*fila1*] [*fila2*] …	Tamaño de cada fila

Como vemos a continuación, si definimos los siguientes valores crearemos una cuadrícula de 2 filas por 3 columnas.

```
.contenedor{
  display: grid;
  grid-template-columns: 50px 300px;
  grid-template-rows: 200px 75px;
}
```

Con la propiedad `display: grid` definimos que queremos crear un grid, y mediante las propiedades `grid-template-columns` y `grid-template-rows` definimos los tamaños de las columnas y las filas del mismo. Esto significa que, a priori, tendríamos una cuadricula o grid de **4 celdas** en total:



En el ejemplo anterior he utilizado **píxels** como unidades de las celdas de la cuadrícula, sin embargo, también podemos utilizar otras unidades (*o incluso combinarlas*): porcentajes, la palabra clave `auto` (*que obtiene el tamaño restante*) o la unidad especial de grid `fr` (*fracción restante*), que explicaremos a continuación.

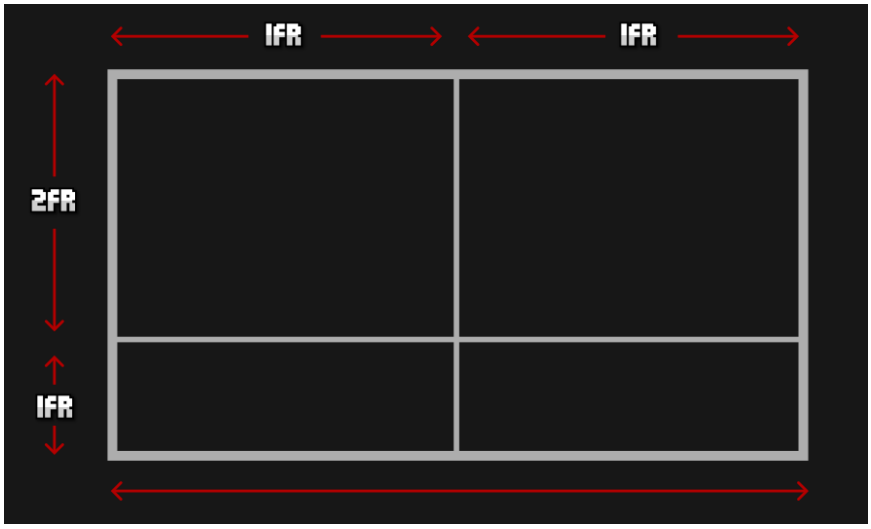
Supongamos el siguiente fragmento de código, donde utilizamos las unidades `fr`:

```
.grid {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: 2fr 1fr;
}
```

Este nuevo ejemplo, también crea una cuadrícula de **2x2**, donde el tamaño de la cuadrícula se divide en:

- **Dos columnas**: Mismo tamaño de ancho para cada una.
- **Dos filas**: La primera fila ocupará el doble (*`2fr`*) que la segunda fila (*`1fr`*).

```
.grid {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: 2fr 1fr;
}
```



De esta forma, es muy fácil predecir el espacio que va a ocupar la cuadrícula, ya que sólo tenemos que sumar todas las unidades para saber el tamaño total, y comparar con cada columna o fila para saber como de grande o pequeña es respecto al total. Así tendremos un mejor control del espacio restante de la cuadrícula, y resultará más intuitivo calcularlo.

Se pueden combinar varias **unidades diferentes**, como por ejemplo píxeles (*px*), fracciones restantes (*fr*), porcentajes (%) y otras combinaciones similares.

14.2.1.- Filas y columnas repetitivas

En algunos casos, en las propiedades grid-template-columns y grid-template-rows podemos necesitar indicar las mismas cantidades un número alto de veces, resultando repetitivo y molesto escribirlas varias veces. Se puede utilizar la función repeat() para indicar repetición de valores, señalando el número de veces que se repiten y el tamaño en cuestión.
La expresión a utilizar sería la siguiente: `repeat(número de veces, tamaño)`:

```
.grid {
  display: grid;

  grid-template-columns: 100px repeat(4, 50px) 200px;
  grid-template-rows: repeat(2, 1fr 2fr);

  /* Equivalente a... */
  grid-template-columns: 100px 50px 50px 50px 50px 200px;
  grid-template-rows: 1fr 2fr 1fr 2fr;
}
```

Asumiendo que tuvieramos un contenedor grid con 24 ítems hijos en el HTML, el ejemplo anterior crearía una cuadrícula con **6 columnas** y **4 filas**. Recuerda que en el caso de tener más ítems hijos, el patrón se seguiría repitiendo.

14.2.2.- Función minmax()

La función `minmax()` se puede utilizar como valor para definir rangos flexibles de celda. Funciona de la siguiente forma:

`minmax(min, max)` Define un rango entre `min` y `max`.

Si establecemos un rango, por ejemplo, grid-template-column: minmax(200px, 500px), estaremos indicando que la celda de columna indicada, tendrá un tamaño de 500px, salvo que redimensionemos la ventana del navegador y la hagamos más pequeña, en cuyo caso, el tamaño de la celda podría ir disminuyendo hasta 200px, medida en la cuál se quedaría como mínimo.

Prueba con este ejemplo, y prueba a redimensionar la ventana del navegador:

```
<div class="container">
  <div class="item item-1">Item 1</div>
  <div class="item item-2">Item 2</div>
  <div class="item item-3">Item 3</div>
  <div class="item item-4">Item 4</div>
</div>

<style>
.container {
  display: grid;
  grid-template-columns: repeat(2, minmax(400px, 600px));
  grid-template-rows: repeat(2, 1fr);
  gap: 5px;
}

.item {
  background: black;
  color: white;
  padding: 1em;
}
</style>
```

Comprobarás que las celdas se hacen más pequeñas hasta un punto en el que se alcanza el mínimo.

14.2.3.- Auto-fill y Auto-fit

En la función `repeat()` es posible utilizar las palabras claves `auto-fill` o `auto-fit` para indicar al navegador que queremos que **rellene** o **ajuste** el contenedor grid con múltiples elementos hijos dependiendo del tamaño del **viewport** (*región visible del navegador*).

Es decir, si utilizamos `repeat(auto-fill, minmax(300px, 1fr)`, el navegador se va a encargar de que los elementos hijos con el tamaño mínimo quepan en la primera fila, y los que no quepan, se desplacen a las siguientes filas del grid, de modo que se aproveche lo mejor posible el contenedor, y consigamos un efecto similar al que se consigue con **media queries**, pero de una forma más directa y con menos código.

Imagina el siguiente ejemplo, con un grid con 10 ítems:

```
.grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
  background: grey;
  gap: 10px;
}

.item {
  background: blue;
  color: #fff;
  font-size: 2rem;
}
```

Si cambiamos el ejemplo anterior a `auto-fit` no veremos ninguna diferencia. Sin embargo, si por ejemplo cambiamos el valor mínimo de `300px` a `50px` (*de modo que no llegue a cubrir la primera fila completamente*), comprobaremos que mientras `auto-fill` va **rellenando** la fila del grid y deja el resto del espacio libre, `auto-fit` **ajusta** el tamaño de los ítems para que cubran el tamaño máximo de la fila.

Ejemplo de auto-fill

Ejemplo de auto-fit

14.2.4.- Atajo: La propiedad grid-template

Si acostumbras a utilizar estas propiedades frecuentemente, puedes utilizar la propiedad `grid-template`, que sirve de atajo para muchas cosas, y una de ellas, resumir en una sola propiedad los valores que tenemos en `grid-template-columns` y en `grid-template-rows`:

Propiedad	Valores	Descripción
`grid-template`	**none** | *grid-template-rows* / *grid-template-columns*	Atajo para definir dimensiones del grid.

Esta propiedad convierte en un proceso bastante cómodo el crear grids de unas dimensiones concretas de forma resumida. En el caso de utilizar el valor `none`, las propiedades `grid-template-rows`, `grid-template-columns` y la propiedad `grid-template-areas`, que veremos más adelante en el tema de Grid por áreas, se establecen a sus valores por defecto, desactivando su funcionamiento.

En el caso de utilizar unos valores definidos, la propiedad `grid-template-areas` se establecerá a `none`.

14.3.- Huecos en grid

Por defecto, la cuadrícula tiene todas sus celdas pegadas a sus celdas contiguas. Aunque sería posible darle un `margin` a las celdas dentro del contenedor, existe una forma más apropiada: los huecos (*gutters*).

Para especificar los **huecos** (*espacio entre celdas*) podemos utilizar las propiedades `column-gap` y/o `row-gap`. En ellas indicaremos el tamaño de dichos huecos:

Propiedad	Descripción
`column-gap`	Espaciado entre celda y celda que se encuentre en columna
`row-gap`	Espaciado entre celda y celda que se encuentre en fila

En el siguiente ejemplo, hemos definido una cuadrícula con tres columnas de igual tamaño utilizando la propiedad `grid-template-columns`. Luego, hemos utilizado la propiedad `column-gap` para establecer un espacio de 20 píxeles entre las columnas y la propiedad row-gap para establecer un espacio de 10 píxeles entre las filas. También hemos agregado algunos elementos a la cuadrícula y les hemos dado un color de fondo y un relleno para que puedas ver el efecto del espacio entre las columnas y las filas

```
.grid-container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  column-gap: 20px;
  row-gap: 10px;
}
.grid-item {
  background-color: lightblue;
  padding: 10px;
}
```

14.3.1.- Atajo: Propiedad `grid-gap`

La propiedad `grid-gap` de CSS Grid es una propiedad abreviada para establecer tanto `grid-column-gap` como `grid-row-gap` en una sola declaración. Esta propiedad acepta uno o dos valores, donde el primer valor especifica el tamaño del espacio entre las filas y el segundo valor especifica el tamaño del espacio entre las columnas. Si solo se proporciona un valor, se aplica a ambas propiedades. Ejemplo:

```
.grid-container {
display: grid;
grid-template-columns: repeat(3, 1fr);
grid-gap: 10px 20px;
}
.grid-item {
  background-color: lightblue;
  padding: 10px;
}
```

1	2	3
4	5	6

En este ejemplo, hemos definido una cuadrícula con tres columnas de igual tamaño utilizando la propiedad grid-template-columns. Luego, hemos utilizado la propiedad column-gap para establecer un espacio de 20 píxeles entre las columnas y la propiedad row-gap para establecer un espacio de 10 píxeles entre las filas. También hemos agregado algunos elementos a la cuadrícula y les hemos dado un color de fondo y un relleno para que puedas ver el efecto del espacio entre las columnas y las filas.

14.4.- Alinear elementos

Al igual que con la maquetación **Flex CSS, Grid** incorpora un sistema para alinear elementos que se basa en **Flex** y es incluso más potente, ya que permite la alineación de elementos en **dos dimensiones**, así como centrar o colocar elementos hijos del contenedor **Grid**.

14.4.1.- Propiedades de alineación

Existen una serie de propiedades que se pueden utilizar para colocar y ajustar nuestra cuadrícula **grid** o ajustar los ítems a lo largo de ella, de forma sencilla y cómoda. Algunas de estas propiedades probablemente ya las conocerás del módulo CSS **flex**, sin embargo, en **grid** pueden tener un comportamiento diferente.

Dichas propiedades pueden afectar al **eje principal** o al **eje secundario** del grid y son las siguientes:

Propiedad	Valores	Afecta a...
`justify-items`	start | end | center | stretch	Elementos
`align-items`	start | end | center | stretch	Elementos
`justify-content`	start | end | center | stretch | space-around | space-between | space-evenly	Contenido
`align-content`	start | end | center | stretch | space-around | space-between | space-evenly	Contenido

Estas propiedades se aplican sobre el elemento contenedor padre, sin embargo, afectan tanto al contenedor como al comportamiento de los elementos hijos.

Antes de continuar, un pequeño resumen:

- `justify-items`: Alinea los elementos (*hijos*) en **horizontal** (*eje principal*) dentro de cada celda.
- `align-items`: Alinea los elementos (*hijos*) en **vertical** (*eje principal*) dentro de cada celda.
- `justify-content`: Alinea el contenido (*la cuadricula*) en **horizontal** (*eje primario*) en el contenedor padre.

- `align-content`: Alinea el contenido (*la cuadricula*) en **vertical** (*eje secundario*) en el contenedor padre.

Es importante tener en cuenta que el **grid** se define mediante las propiedades vistas anteriormente. Con estas cuatro propiedades veremos como colocar los ítems hijos dentro de cada celda de la cuadrícula, o la colocación de la cuadrícula en el elemento padre contenedor.

14.4.2.- Propiedades de alineación

14.4.2.1.- Alineación de elementos. La propiedad `justify-items`

La primera propiedad, `justify-items` sirve para colocar los ítems de un contenedor **grid** a lo largo de sus celdas correspondientes, siempre en el **eje principal** (*por defecto, en horizontal*). Los valores que puede tomar esta propiedad son los siguientes:

Valor	Descripción
`start`	Coloca cada ítem al **inicio** de su celda en el eje principal.
`end`	Coloca cada ítem al **final** de su celda en el eje principal.
`center`	Coloca cada ítem en el **centro** de su celda en el eje principal.
stretch	Hace que cada ítem **se estire** y ocupe todo el espacio disponible de su celda en el eje principal.

14.4.2.2.- Alineación de elementos. La propiedad `align-items`

De forma análoga, la propiedad `align-items` sirve para colocar los ítems de un contenedor **grid** a lo largo de sus celdas correspondientes, pero en lugar de el eje principal, las coloca en el **eje secundario** (*por defecto, en vertical*). Los valores que puede tomar son los mismos que la propiedad anterior.

14.4.2.3.- Alineación de contenido. La propiedad `justify-content`

La propiedad `justify-content` permite modificar la distribución del contenido de la cuadrícula en su contenedor padre, a lo largo de su eje principal (*por defecto, el horizontal*). Los valores que puede tomar son los siguientes:

Valor	Descripción
`start`	Coloca la cuadrícula en su conjunto al **inicio** del contenedor padre en su eje principal (*horizontal*).
`end`	Coloca la cuadrícula en su conjunto al **final** del contenedor padre en su eje principal (*horizontal*).
`center`	Coloca la cuadrícula en su conjunto al **centro** del contenedor padre en su eje principal (*horizontal*).

Valor	Descripción
stretch	**Estira** la cuadrícula ocupando todo el espacio disponible del contenedor padre en su eje principal (*horizontal*).
`space-between`	Establece espacios sólo entre las celdas, en su eje principal (*horizontal*).
`space-around`	Establece espacios alrededor de las celdas, en su eje principal (*horizontal*).
`space-evenly`	Idem al anterior, pero solapando los espacios, de modo que sean todos de tamaño equivalente.

14.4.2.4.- Alineación de contenido. La propiedad `align-content`

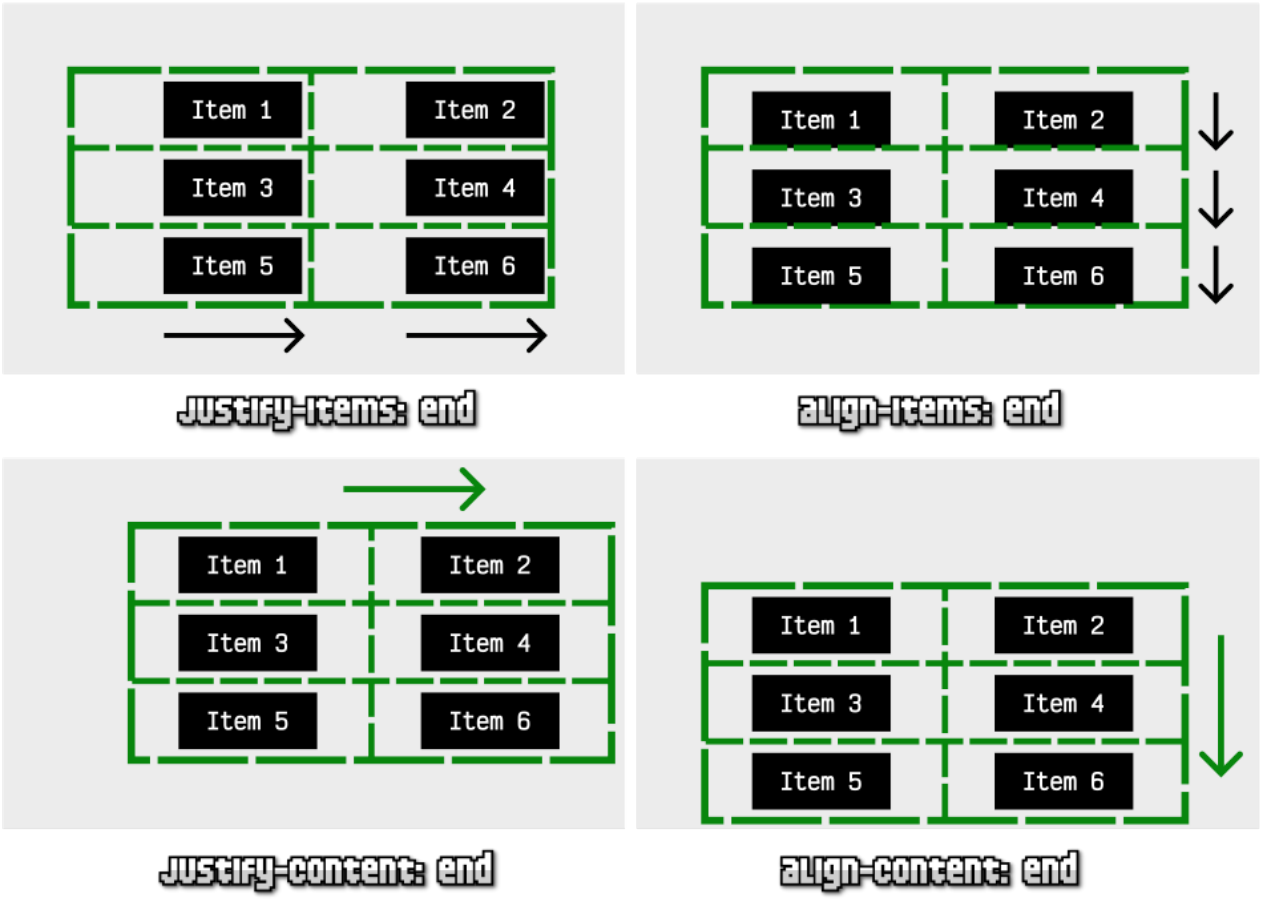
De forma análoga, la propiedad `align-content` sirve para colocar el contenido de la cuadrícula en su contenedor padre, pero a lo largo de su contenedor secundario (*por defecto, el vertical*). Los valores que puede tomar son exactamente los mismos que la propiedad anterior.

Vamos a partir de un escenario con el siguiente código HTML y CSS, donde planteamos unas bases y profundizaremos en las propiedades anteriormente mencionadas:

```
.container {
  display: grid;
  grid-template-columns: repeat(2, 250px);
  grid-template-rows: repeat(3, 50px);
  gap: 10px;
  background: grey;
  height: 300px;

  justify-items: center;
  align-items: center;
  justify-content: center;
  align-content: center;
}

.item {
  padding: 10px;
  background: black;
  color: white;
}

<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
  <div class="item">Item 4</div>
  <div class="item">Item 5</div>
  <div class="item">Item 6</div>
</div>
```

Observa que tenemos **6 elementos hijos**, dentro de un contenedor donde hemos creado un grid de **2 columnas x 3 filas** (*en rojo, realmente es invisible*). Observa hemos aplicado las cuatro propiedades indicadas en la tabla de arriba con el valor `center`. A continuación, vamos a variar sólo una de ellas cada vez, colocando el valor `end` para observar su repercusión.

Este sería el resultado al aplicar cada una de las propiedades individuales anteriores:



14.4.3.- Propiedades de alineación

En el caso de que queramos que uno de los ítems hijos tenga una distribución diferente al resto, podemos aplicar en el elemento hijo la propiedad `justify-self` (*eje principal*) o `align-self` (*eje secundario*) sobreescribiendo su distribución su general, y aplicando una específica.

Propiedad	Descripción
`justify-self`	Altera la alineación del ítem hijo en el eje horizontal y la sobreescribe con la indicada.
`align-self`	Altera la alineación del ítem hijo en el eje vertical y la sobreescribe con la indicada.

Recuerda que estas propiedades funcionan exactamente igual que sus análogas `justify-items` o `align-items` y tienen los mismos valores, sólo que en lugar de indicarse en el elemento padre contenedor, se hace sobre un elemento hijo y repercute en dicho elemento hijo específicamente.

14.4.4.- Atajo: Alineaciones Grid

Si vamos a crear estructuras grid donde utilicemos los pares de propiedades `justify-items` y `align-items` por un lado, `justify-content` y `align-content` por otro, e incluso `justify-self` y `align-self` por otro, podemos utilizar las siguientes propiedades de atajo:

Propiedad	Valores	Descripción
`place-items`	[*align-items*] [*justify-items*]	Propiedad de atajo para ***-items**
`place-content`	[*align-content*] [*justify-content*]	Propiedad de atajo para ***-content**
`place-self`	[*align-self*] [*justify-self*]	Propiedad de atajo para ***-self**

Con ellas conseguiremos que nuestro código sea más simple, menos texto y más directo:

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(2, 1fr);

  place-items: center;
  /* Equivale a... */
  justify-items: center;
  align-items: center;

  place-content: center;
  /* Equivale a... */
  justify-content: center;
  align-content: center;
}

.item:first-child {
  place-self: end;
  /* Equivale a... */
  justify-self: end;
  align-self: end;
}
```

14.4.5.- Orden de los elementos

Por último, tenemos la propiedad `order`. Funciona exactamente igual que como funciona en **flex**. Es una propiedad mediante la cual podemos modificar y establecer un orden de los elementos mediante números que actuarán como «peso» del elemento:

Propiedad	Valor	Descripción
`order`	**0** |	Número (peso) que indica el orden de aparición de los ítems.

Por defecto, todos los elementos hijos de un contenedor flex tienen establecido un `order` por defecto al valor `0`. Si indicamos una propiedad `order` con un valor numérico diferente, recolocará los ítems según dicho número, colocando antes los elementos con un número

`order` más pequeño (*incluso números negativos*) y los elementos con números más altos después.

14.5.- Grid por áreas

Mediante los **Grids por área** es posible indicar el nombre y posición concreta de cada área de una cuadrícula. Para ello utilizaremos la propiedad `grid-template-areas` en nuestro contenedor padre, donde debemos especificar el orden de las áreas en la cuadrícula. Posteriormente, en cada ítem hijo, utilizamos la propiedad `grid-area` para indicar el nombre del área del que se trata y que el navegador pueda identificarlas:

Propiedad	Descripción
`grid-template-areas`	Indica la disposición de las áreas en el grid. Cada texto entre comillas simboliza una fila.
`grid-area`	Indica el nombre del área. Se usa sobre ítems hijos del grid.

De esta forma, es muy sencillo crear una cuadrícula altamente personalizada en apenas unas cuantas líneas de CSS, con mucha flexibilidad en la disposición y posición de cada área. Veamos un ejemplo:

css
html

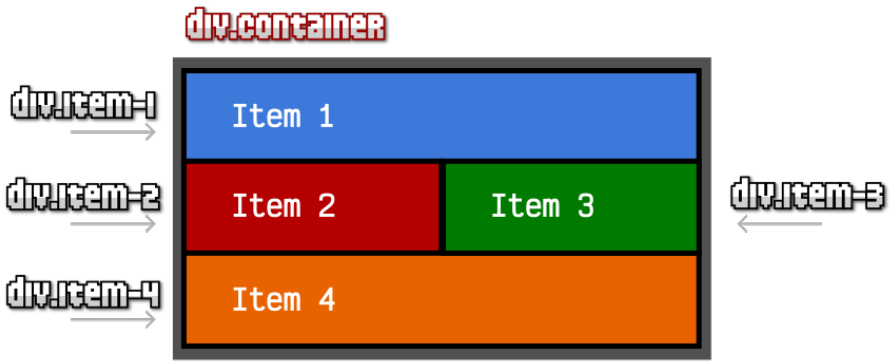
```
.container {
  display: grid;
  grid-template-areas: "head head"
                       "menu main"
                       "foot foot";

  grid-template-columns: 1fr 1fr;
  grid-template-rows: 100px 200px 100px;
}

.item-1 { grid-area: head; background: blue; }
.item-2 { grid-area: menu; background: red; }
.item-3 { grid-area: main; background: green; }
.item-4 { grid-area: foot; background: orange; }

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

> Recuerda que **área** y **nombre de clase** son cosas independientes y diferentes. Es muy importante no confundirlas.

Aplicando este código, conseguiríamos una cuadrícula donde:

- **Item 1** es la cabecera (*head*), que ocupará la primera fila (*toda la parte superior*).

- **Item 2** es el menú lateral (*menu*), que ocupará el área izquierda del grid (*debajo de la cabecera*).
- **Item 3** es el contenido (*main*), que ocupará el área derecha del grid (*debajo de la cabecera*).
- **Item 4** es el pie de cuadrícula (*foot*), que ocupará la última fila (*área inferior del grid*).



OJO: Ten en cuenta añadir contenido de texto en cada celda del grid, ya que si no existe contenido y no has definido un tamaño de fila/columna, el grid se adaptará a su contenido (*que no lo hay*) y parecerá que no existe. También ten en cuenta que puedes combinar con propiedades como `grid-template-columns` y/o `grid-template-rows` para añadirle tamaño o dimensiones.

14.5.1.- La propiedad `grid-template-areas`

La propiedad `grid-template-areas` es la propiedad principal de este sistema, y debe utilizarse en el contenedor padre **grid**.

Propiedad	Valores	Descripción
`grid-template-areas`	**none** | *fila1*, *fila2*, ...	Define cada fila del grid, indicando el nombre del área a colocar.

Cada una de estas **filas** se definen como un donde indicaremos el nombre de un **área** que posteriormente definiremos en nuestro código CSS. Cada fila puede tener ninguna o varias áreas que habría que separar por espacio. A continuación veremos algunos ejemplos de los valores que podemos indicar en esta propiedad y su significado:

Valores	Descripción
none	Indica que no se creará ninguna plantilla de áreas.
`"head"`	Indica que se creará una fila de una columna con el área head.

Valores	Descripción
`"head menu"`	Indica que se creará una fila de 2 columnas con el área head en una y el área menu en otra.
`"head head"`	Indica que se creará una fila de 2 columnas con el área head ocupando ambas.
`"."`	Indica que se colocará una celda sin nombre (*nula*) en esta posición.

Recuerda que las áreas deben existir y estar definidas con la propiedad `grid-rea`, de lo contrario, se anulará la propiedad.

14.5.2.- La propiedad `grid-area`

Por otro lado, al utilizar la propiedad `grid-template-areas` y nombrar varias áreas en su valores, es necesario que dichas áreas estén definidas mediante la propiedad `grid-area` en sus elementos hijos. Recuerda no confundir nombre de área, con nombre de clase, puesto que no es lo mismo.

Propiedad	Valores	Descripción
`grid-area`	**auto** | *nombre*	Da un nombre de área al elemento indicado.

Esta propiedad permite nombrar un elemento del HTML con un **nombre de área**. Mucho cuidado, ya que este nombre no es un string, y por lo tanto no debe definirse entre comillas `"`. Estos nombres se utilizarán en la propiedad `grid-template-areas` para definir donde irán ubicados.

Los valores que puede tomar la propiedad `grid-area` son los siguientes:

Valores	Descripción
auto	Coloca la celda en la próxima área vacía que se encuentre disponible.
nombre	Le da un nombre de área al elemento en cuestión.

14.5.3.- Celdas irregulares

Hasta ahora, salvo algunas excepciones como `justify-self`, `align-self` o `grid-area`, hemos visto propiedades CSS que se aplican solamente al contenedor padre de una cuadrícula. Las siguientes propiedades se aplican a los ítems hijos de una cuadrícula, para alterar o cambiar el comportamiento específico de dicho elemento, que no se comportará como el resto.

Las propiedades para crear esa distribución irregular de una celda del grid son las siguientes:

Propiedad	Descripción
`grid-column-start`	Indica en que columna empezará el ítem de la cuadrícula.
`grid-column-end`	Indica en que columna terminará el ítem de la cuadrícula.
`grid-row-start`	Indica en que fila empezará el ítem de la cuadrícula.
`grid-row-end`	Indica en que fila terminará el ítem de la cuadrícula.

Observa que tenemos dos pares de propiedades, una con el prefijo `grid-column-` y otro par con el prefijo `grid-row-`. Luego, dentro de ambas, tenemos un sufijo `-start` para indicar donde empieza y otra con un sufijo `-end` para indicar donde termina. Con dichas propiedades, estableceremos **casos particulares** donde, a un elemento hijo, le asignaremos la parte de la cuadrícula donde debe empezar y donde terminar, creando un caso particular sobre el resto.

14.5.3.1.- Valores posibles

En las propiedades anteriores podemos establecer diferentes valores para indicar donde comienza o termina una **celda irregular** en nuestro grid de CSS:

Valor	Descripción	Más info
auto	No se indica ningún comportamiento particular. Se queda igual. **Valor por defecto**.	
línea	Indica la línea específica, es decir, la separación de columnas o filas.	
`span` *línea*	Indica hasta cuántas líneas debe llegar.	
nombre	Idem al anterior, pero con líneas nombradas.	Ver líneas nombradas
`span` *nombre*	Idem al anterior, indicando nombre de línea hasta donde llegar.	Ver líneas nombradas
nombre número	Idem al anterior, pero busca el *nombre* que aparece por *número* vez.	Ver líneas nombradas

Observa que mientras que el valor `auto` (*valor por omisión*) no cambia en nada nuestra celda, indicando un *número* haremos referencia a la linea que divide las celdas del grid. Por ejemplo, si tenemos una fila con **4 columnas**, ten en cuenta que tendríamos **5 líneas**.

De esta forma, indicando un valor numérico haríamos referencia a la línea en cuestión, pero si indicamos la palabra clave `span` antes, haremos referencia a cuántas líneas deberemos alargar la celda.

```
.item-6 {
  grid-column-start: 2;
  grid-column-end: 4;
}

.item-6 {
```

```
  grid-column-start: 2;
  grid-column-end: span 2;
}
```

Los dos ejemplos anteriores son equivalentes. El primero define que se empieza en la línea 2 hasta la 4, por lo que abarca dos celdas: 2-3 y 3-4. En el segundo caso, definimos que se empieza en la línea 2 y la celda se alargará 2 celdas más: hasta la 3 y hasta la 4.

15.- Centrar horizontal y verticalmente con CSS un elemento

Existen **diversas formas de centrar horizontal y verticalmente un elemento en un contenedor mediante CSS**. Cada método es adecuado para una situación concreta y, por ello, es necesario saber cómo funcionan todas las propiedades para reconocer fácilmente qué técnica utilizar en cada caso.

A continuación se describe cómo se puede realizar el centrado de algunos elementos mediante **cajas flexibles**, con la propiedad "**transform**", con las propiedades "**line-height**" y "**text-align**" y con la propiedad **vertical-align**.

15.1.- Centrar vertical y horizontalmente elementos con flexbox

Este es el **método más útil** porque funciona para todos los elementos que se posicionen dentro de un contenedor. Por tanto, es **válido para imágenes, textos, vídeos**, etc. Sin embargo, para comprender cómo funciona es necesario tener conocimientos sobre el modelo de caja flexible o *flexbox*.

Para utilizar las propiedades de ***flexbox*** es necesario crear un contenedor y declararle la propiedad "**display: flex**". Para centrar horizontal y verticalmente el contenido utilizaremos las propiedades **"justify-content: center"** y "**align-items:center**".

```
div{
    height: 200px;
    background-color: #EEE;
    display:flex;
    justify-content: center;
    align-items: center;
}
```

15.1.1.1.- Ejemplo

Figure 9: Ejemplo de centrado con flexbox

15.2.- Centrar vertical y horizontalmente un texto con line-height y text-align

Mediante las propiedades **line-height** y **text-align** podemos **alinear únicamente textos**. Así que este método sólo va a ser **válido para el centrado de párrafos, encabezados**, etc.

```
div {
    height: 200px;
    background-color: #EEE;
    text-align: center;
}
div h1 {
    line-height: 200px;
}
```

15.2.1.1.- Ejemplo

Figure 10: Ejemplo de centrado de texto

15.3.- Centrar vertical y horizontalmente una imagen o contenedor con propiedad transform y posición absoluta

Mediante este método se **puede alinear tanto vertical como horizontalmente imágenes y contenedores**. **No es válido para la alineación de textos** ya que no tiene en cuenta el tamaño de la letra. El centrado de elementos se realiza mediante posición absoluta y la propiedad transform.

```
div {
    height: 300px;
    background-color: #EEE;
    position: relative;
}
div img{
    position: absolute;
    left: 50%;
    top: 50%;
    transform: translate(-50%, -50%);
    -webkit-transform: translate(-50%, -50%);
}
```

Ejemplo

Figure 11: Ejemplo de centrar vertical y horizontalmente una imagen

15.4.- Centrar un elemento vertical y horizontalmente utilizando posición absoluta y margen negativo

Centrado absoluto con posición absoluta y margen negativo:

```
.container {
  position: relative;
}

.centered-element {
  position: absolute;
  top: 50%;
  left: 50%;
  margin-top: -50px; /* Ajusta el valor según la mitad de la altura del
elemento */
  margin-left: -50px; /* Ajusta el valor según la mitad del ancho del elemento
*/
}
```

15.5.- Centrar un elemento con grid

Para centrar un elemento con grid se debe utilizar `display: grid` y `place-items: center` en el contenedor. De esta forma, el elemento se centrará tanto vertical como horizontalmente dentro del contenedor.

```
.container {
  display: grid;
  place-items: center;
}
```

En el siguiente ejemplo, el contenedor tiene una altura fija de 300px y un borde para que puedas visualizarlo. El elemento que deseas centrar está dentro del contenedor y tiene un fondo gris claro y un relleno de 20px.

15.6.- Alinear un texto verticalmente con una imagen con vertical-align

Con esta técnica podemos **alinear un texto a una imagen de forma vertical**. Esta propiedad admite los valores ***top***, ***middle*** y ***bottom***, entre otros valores.

```
img{
    vertical-align: middle;
}
```

Ejemplo

Figure 12: Ejemplo de alinear un texto verticalmente con una imagen

15.7.- Alinear verticalmente un texto en un div con vertical-align

Vamos a alinear un texto en un div con la propiedad vertical-align. Hay que tener en cuenta que esta propiedad no se comporta de la misma manera con todos los elementos. En este caso, haremos uso de las reglas de estilo “display: inline-table” y “display: table: cell”.

```
div{
   height: 200px; width: 200px;
   background-color: #EEE;
   display: inline-table;
 }
 p{
   display: table-cell;
   vertical-align: middle;
 }
```

Ejemplo

Figure 13: Ejemplo de alinear verticalmente un texto en un div

15.8.- Alinear horizontalmente una imagen con text-align

Las imágenes se pueden alinear a la izquierda, derecha o centro insertándolas dentro de un div e indicando la propiedad text-align correspondiente.

```
.a{text-align: left;}
 .b{text-align: center;}
 .c{text-align: right;}
```

Ejemplo

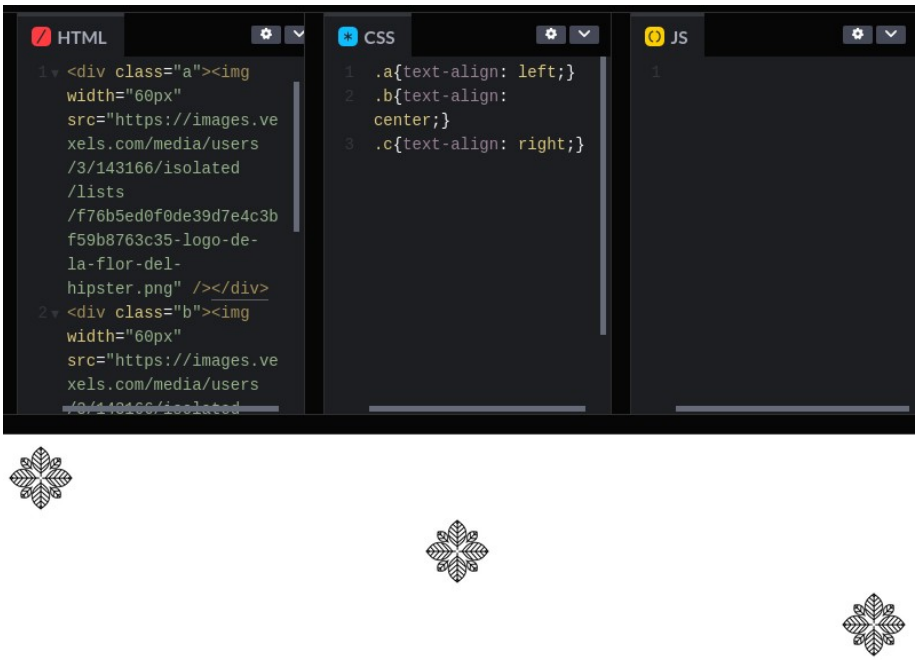


Figure 14: Ejemplo de alinear horizontalmente una imagen con text-align

15.9.- Alinear verticalmente un checkbox con un label

Cuando el tamaño del texto no coincide con el tamaño del *checkbox*, estos dos elementos no quedan alineados verticalmente. Para conseguir esta alineación utilizaremos "**vertical-align:middle**" y **definiremos el tamaño del texto a las dos etiquetas**, tal y como se muestra a continuación.

```
label, input{
    font-size: 28px;
    vertical-align: middle;
}
```

Figure 15: Ejemplo de alinear verticalmente un checkbox con un label

16.- Preprocesadores Less y Sass

La **simplicidad de CSS** hace que sea muy limitado. Por ejemplo, en el caso de necesitar cambiar el valor de una propiedad para que sea igual en muchos documentos se hace muy costoso. Además, tampoco se pueden utilizar condicionales, variables o estructuras anidadas. Para facilitar estas tareas, y muchas otras, nacen los **preprocesadores CSS**.

Mediante los preprocesadores CSS se desarrollan **estilos más mantenibles y con una sintaxis más rica**. Los ficheros de los preprocesadores son parecidos a los de CSS y

deben compilarse para traducirse a hojas de estilo CSS reconocibles por los diferentes navegadores.

16.1.- Preprocesadores Less y Sass

Los **preprocesadores más utilizados son Less y Sass**. Less es muy sencillo de instalar en caso de trabajar con Node.js y Saas requiere soporte para lenguaje Ruby.

16.1.1.- Ejemplos Sass:

Variables:

```
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
     font:  $font-stack;
     color: $primary-color;
}
```

Nesting:

```
nav {
    ul {
        margin: 0;
        padding: 0;
        list-style: none;
    }
    li {
        display: inline-block;
    }
}
```

Operators:

```
article {
    float: left;
    width: 600px / 960px * 100%;
}
aside {
    float: right;
    width: 300px / 960px * 100%;
}
```

Mixins:

```
@mixin transform($property) {
    -webkit-transform: $property;
    -ms-transform: $property;
    transform: $property;
}
.box {
```

```
    @include transform(rotate(30deg));
}
```

17.- Referencias

17.1.- Bibliográficas

- Libro: Curso de Desarrollo Web: HTML, CSS y JavaScript. Edición 2018
- Libro: SEO Avanzado. Casi todo lo que sé de posicionamiento web (Social Media) – Maciá Domene, Fernando

17.2.- Web

- lenguajecss.com
- developer.mozilla.org/es/docs/Web/CSS
- w3schools.com/css/default.asp
- desarrolloweb.dlsi.ua.es/libros/html-css/ejercicios
- w3c.es
- desarrolloweb.dlsi.ua.es/libros/html-css/ejercicios
- en.wikipedia.org/wiki/Comparison_of_browser_engines_(CSS_support)
- escss.blogspot.com/2012/06/css-speech-module-css-hablado.html
- websitesetup.org/wp-content/uploads/2019/11/wsu-css-cheat-sheet-gdocs.pdf

17.3.- Cursos recomendados

- es.khanacademy.org/computing/computer-programming/html-css/intro-to-css/pt/css-basics

Aclaraciones:

Párrafo resaltado de advertencia

Párrafo resaltado de información

Párrafo resaltado de advertencia negativa

Párrafo resaltado positivo