

# UD1\_ Aplicacións web e linguaxes de marcas: HTML e CSS

---

## ACTIVIDADE 5: Continuando con CSS

# Índice

---

<b>1.</b>	<b>Prefijos.....</b>	<b>3</b>
<b>2.</b>	<b>Más selectores.....</b>	<b>3</b>
2.1	Pseudo-clases con CSS3.....	3
2.1.1	Utilizar fórmulas.....	6
2.2	Pseudo-elementos.....	6
<b>3.</b>	<b>Diseño web responsive.....</b>	<b>8</b>
3.1	Media queries.....	8
<b>4.</b>	<b>Posicionamiento.....</b>	<b>10</b>
4.1	Posicionamiento flotante.....	11
4.2	Posicionamiento absoluto.....	12
4.3	Superposición de cajas.....	14
4.4	La propiedad display.....	14
4.5	La propiedad visibility.....	15
4.6	Contenido desbordado. Propiedad <code>overflow</code> .....	15
4.7	Flexbox. Cajas flexibles.....	16
<b>5.</b>	<b>Más propiedades CSS.....</b>	<b>22</b>
5.1	Propiedades de fuentes y texto.....	22
5.1.1	Sombreado de texto.....	23
5.1.2	Sombreado de cajas.....	23
5.2	Background en CSS3.....	24
5.2.1	Background con varias imágenes.....	26
5.3	Bordes redondeados.....	27
5.4	Trabajar con columnas en CSS3.....	28
5.5	Transform y transition.....	29
5.5.1	Transform.....	29
5.5.1.1	Transformaciones dinámicas.....	31
5.5.2	Transition.....	31

# 1. Prefijos

---

Antes de ver más propiedades vamos a ver que son los prefijos.

Los prefijos se usan con las propiedades css3 que están en estado experimental para conseguir un aspecto uniforme en todos los navegadores. Los más importantes son:

- moz- Para Firefox
- webkit- Para navegadores del grupo WebKit como Chrome o Safari.
- ms- Para navegadores Microsoft (IE)
- o- Para Presto, el antiguo motor de Opera.

Por ejemplo, para repartir el texto en dos columnas:

```
.columnas {  
    -webkit-column-count : 2;  
    -moz-column-count : 2;  
    column-count : 2; }
```

Los navegadores usan la primera línea de código que entienden e ignoran las demás. De este modo nos aseguramos que los navegadores antiguos entiendan la propiedad. Los navegadores nuevos, que ya no necesitan prefijo, se quedarán con la última línea de código (la propiedad sin prefijo).

## 2. Más selectores

---

### 2.1 Pseudo-clases con CSS3

Otro método de dar formato a una página web es a través de las pseudo-clases CSS. Su uso más común es con las etiquetas que identifican enlaces como se visualiza a continuación:

```
a:link {color:#FF0000;} /* enlace no visitado */  
a:visited {color:#00FF00;} /* enlace visitado */  
a:hover {color:#FF00FF;} /* ratón sobre el enlace */  
a:active {color:#0000FF;} /* enlace seleccionado */
```

Las pseudo-clases y clases CSS

Las pseudo-clases se pueden combinar con clases CSS:

```
a.red:visited {color:#FF0000;}  
  
<a class="red" href="mienlace.html">CSS3</a>
```

Si el enlace del ejemplo anterior se ha visitado, se mostrará en rojo.

:root

Selecciona el elemento HTML por ser la raíz del documento. Aplica los estilos definidos a *todo el documento*.

```
:root{  
    font-family:Arial, Helvetica, sans-serif;  
    color:#c00;  
}
```

### :first-child

El primer hijo de un elemento. En el siguiente ejemplo, el selector equivale a cualquier elemento `p` que es el primer hijo de cualquier elemento.

```
p:first-child
{
  color:blue;
}
```

Y el siguiente selector equivale a cualquier elemento `p` que es el primer hijo de un elemento `div`

```
div > p:first-child
{
  color:blue;
}
```

Para todos los elementos `em` que se encuentran en un `p` que es primer hijo, sería:

```
p:first-child em
{
  color:blue;
}
```

### :last-child

El último hijo de un elemento. Funciona igual que el anterior pero para el último hijo.

### :only-child

El único hijo de un padre. Para seleccionar los enlaces que son hijos únicos de su padre:

```
a:only-child{
  text-decoration:none;
  color:red;
}
```

### :nth-child(n)

El enésimo hijo de un padre. De este modo seleccionamos el hijo situado en segunda posición del `div` de clase ejemplo.

```
div.ejemplo:nth-child(2){
  color:blue;
}
```

### :nth-child (even)

Con este ejemplo seleccionaremos los elementos `p` que tengan una posición par, para esto empleamos la palabra `even`.

```
p:nth-child(even) {
  color:blue;
}
```

### :nth-child (odd)

Con este ejemplo seleccionaremos los elementos `p` que tengan una posición impar

```
p:nth-child(odd) {
  color:#9FF;
}
```

### :nth-last-child(n)

El enésimo hijo de su padre, contando del último al primero.

### :nth-of-type()

Con esta pseudo-clase podremos seleccionar mediante la posición, elementos hijos de una forma alterna pero de un determinado tipo. Por ejemplo: Si tenemos un `div` con un `h2` y tres párrafos, para seleccionar el segundo párrafo sería:

```
p:nth-of-type(2) {  
    color:blue;  
}
```

### :nth-last-of-type()

Con esta pseudo-clase podremos seleccionar mediante la posición elementos hijos de una forma alterna pero de un determinado tipo. Siendo la posición inicial el último elemento hijo.

### :first-of-type

Con esta pseudo-clase seleccionamos al primer elemento hijo de un determinado tipo.

### :only-of-type

Esta pseudo-clase selecciona cuando hay un único hijo de ese tipo en ese elemento de un mismo tipo.

### :empty

Selecciona un elemento que no tiene hijos. Se considera como hijo también al texto. Es decir, que un elemento que contenga un espacio en blanco no está vacío, por lo tanto no se aplicarán los estilos declarados.

### :disabled

Elemento desactivado

### :enabled

Elemento activado

### :checked

Elemento seleccionado

### :not()

Selecciona aquellos elementos que no están representados por el argumento. Por ejemplo:

```
p:not(.idea) {  
    color:blue;  
}
```

Seleccionaría todos los párrafos que no pertenecen a la clase `idea`.

## :lang

Selecciona elementos en función de su idioma. Los navegadores utilizan los atributos `lang`, las etiquetas `<meta>` y la información de la respuesta del servidor para determinar el idioma de cada elemento.

```
p:lang(es) { color: red; }
```

No podemos confundir la pseudo-clase `:lang(xx)` con el selector de atributos `[lang|=xx]`, tal y como se muestra con el siguiente ejemplo:

```
*[lang|=es] { ... } /* selector de atributo */
*:lang(es) { ... } /* pseudo-clase */

<body lang="es">
  <p>Por otra parte, los navegadores...</p>
</body>
```

El selector `*[lang|=es]` selecciona todos los elementos de la página que tengan un atributo llamado `lang` cuyo valor empiece por `es`. En el ejemplo anterior, solamente el elemento `body` cumple con la condición del selector.

Por otra parte, el selector `*:lang(es)` selecciona todos los elementos de la página cuyo idioma sea el español, sin tener en cuenta el método empleado por el navegador para averiguar el idioma de cada elemento. En este caso, tanto el elemento `body` como el elemento `p` cumplen esta condición.

### 2.1.1 Utilizar fórmulas

Con las pseudoclasas se pueden utilizar fórmulas. Por ejemplo, si queremos seleccionar los hijos que son múltiplos de 3 deberíamos poner:

```
p:nth-child(3n) {
  color:#9FF;
}
```

donde `n` es un contador que empieza en 0.

Así, si quisiésemos empezar en 1 y seleccionar de 3 en 3 deberíamos poner:

```
p:nth-child(3n+1) {
  color:#9FF;
}
```

Y seleccionaría el párrafo 1, 4, 7, ....

## 2.2 Pseudo-elementos

Los pseudo-elementos que permiten controlar aspectos particulares de un elemento; como por ejemplo, el estilo de la primera letra de una palabra. En CSS2 ya había cuatro pseudo-elementos: `::first-letter`, `::first-line`, `::before` y `::after`

Para distinguirlos de las pseudo-clases en CSS3 empiezan con doble `::`, aunque se permite usar un único `:`

### ::first-letter

Se utiliza para darle un estilo especial a la primera letra de un texto.

```
p::first-letter
{
  color:red;
  font-size:xx-large;
}
```

## ::first-line

Se utiliza para darle un estilo especial a la primera línea del texto.

## ::before

El pseudo-elemento `before` puede ser utilizado para insertar algún contenido antes de que el contenido de un elemento. El siguiente ejemplo se inserta una imagen antes de cada elemento `h1`

```
h1:before
{
    content:url(logo.gif);
}
```

## ::after

El pseudo-elemento `after` puede ser utilizado para insertar algún contenido después de que el contenido de un elemento.

Los pseudo-elementos `::before` y `::after` se utilizan en combinación con la propiedad `content` de CSS para añadir contenidos antes o después del contenido original de un elemento. En el siguiente ejemplo se añade el texto Capítulo - delante de cada encabezado `h1` y un punto detrás de cada párrafo de la página:

```
h1:before {
    content: "Capítulo - ";
}
p:after {
    content: ".";
}
```

## ::selection

Permite dar formato al texto seleccionado en la página. Inicialmente el pseudo-elemento `::selection` fue pensado como parte del CSS3 pero fue suprimido antes de llegar a ser una recomendación. Actualmente aparece en el "Working Draft" ( borrador ) de CSS4.

Con este pseudo-elemento los dobles dos puntos son obligatorios y además debemos incluir los prefijos.

De momento sólo permite modificar las propiedades `color`, `background-color` y `text-shadow` del texto seleccionado.

```
p::-moz-selection{color:green;}
p::selection{color:green;}
```

### Texto no seleccionable

Podemos hacer que el texto no se pueda seleccionar estableciendo la propiedad `user-select` a `none`.

```
p:first-of-type{
    -webkit-user-select: none;
    -moz-user-select: none;
    -ms-user-select: none;
    user-select: none;
}
```

Así impedimos que se seleccione el primer párrafo.



Realiza la tarea 1 de la actividad 5.

## 3. Diseño web responsive

Es importante que una web se visualice correctamente independientemente del dispositivo que use el usuario. A esto responden el diseño web responsive y el diseño web adaptativo.

El diseño web responsive y el diseño web adaptativo son conceptos muy parecidos, pero lo cierto es que no son lo mismo. Ambos son métodos de programación flexible que adaptan la web en función del dispositivo para asegurar su visualización.

La principal diferencia es que el diseño responsive se adapta literalmente. En lo que respecta al diseño adaptativo, se hace un diseño para cada dispositivo (varias resoluciones preestablecidas como ordenador de escritorio, tableta y móvil y todas sus casuísticas).

La base principal del diseño web responsive son los media queries.

### 3.1 Media queries

Los media queries son un módulo de CSS que nos permite identificar el medio en el cual se está mostrando nuestra web y bajo qué condiciones para, en base a esto, se apliquen estilos específicos. Hay dos formas de usar media queries:

#### Atributo media

Con CSS2 ya podíamos tener distintas hojas de estilos en función de los tipos de medios:

```
<link href="estilos.css" rel="stylesheet" media="screen">
```

Con CSS3 podemos usar este mismo atributo para referirnos a tamaños específicos de pantalla

```
<link rel="stylesheet" href="estilos.css" media="only screen and (max-width:640px) ">
```

Con lo que esta hoja se aplicará cuando la página se ven en una ventana que no supere los 640 píxeles de anchura.

#### @media

Hay un modo mejor de hacer esto usando la directiva @media dentro de nuestro código CSS. Incluso podemos tener más de una directiva de este tipo.

Para ello existen unos puntos donde la estructura de nuestra página web va a cambiar que se conocen como breakpoints. La siguiente tabla puede ser usada como una referencia a la hora de usar esta directiva:

	Vertical	Apaisado
Teléfono móvil	320px	480px
táblet pequeño	600px	800px
táblet	768px	1024px
sobremesa	>1224px	
pantallas grandes	>1824px	

Aunque a menudo se usa un único punto de discontinuidad a 768px.

Los medios posibles son:

- all. Valor por defecto. Indica que se aplica el CSS a todos los medios.



- `print`. Para dispositivos de impresión.
- `screen`. Para dispositivos de pantalla.
- `speech`. Para dispositivos sintetizadores de voz.
- `braille`. Dispositivos táctiles de lenguaje braille.
- `embossed`. Impresoras de lenguaje braille.

Se abre la posibilidad de que aparezcan nuevos dispositivos (por ejemplo 3d-glasses).

Dentro de un media query podemos utilizar las siguientes propiedades:

propiedad	Valor
<code>height</code>	Altura de la ventana que muestra la página. Ejemplo: <code>@media screen and (height:800px){</code> /* se ejecuta si la ventana mide 800 píxeles exactos de altura */
<code>width</code>	Anchura de la ventana que muestra la página
<code>device-height</code>	Altura del dispositivo
<code>device-width</code>	Anchura del dispositivo
<code>max-height</code>	Altura máxima de la ventana que muestra el dispositivo
<code>max-width</code>	Anchura máxima de la ventana que muestra el dispositivo
<code>min-height</code>	Anchura mínima de la ventana que muestra el dispositivo
<code>min-width</code>	Anchura mínima de la ventana en la que se muestra la página
<code>max-device-height</code>	Altura máxima del dispositivo
<code>max-device-width</code>	Anchura máxima del dispositivo
<code>min-device-height</code>	Altura mínima del dispositivo
<code>min-device-width</code>	Anchura mínima del dispositivo
<code>orientation</code>	Indica la orientación del dispositivo. Se le puede dar el valor <code>portrait</code> (aspecto tipo retrato, proporciones verticales del dispositivo) o <code>landscape</code> (apaisado).
<code>aspect-ratio</code>	Relación de aspecto que debe de cumplir la ventana que muestra la página. Por ejemplo 16/9
<code>device-aspect-ratio</code>	Relación de aspecto que debe de cumplir la pantalla. Por ejemplo 16/9
<code>max-aspect-ratio</code>	Relación de aspecto máxima que debe de cumplir la ventana que muestra la página. Por ejemplo si ponemos 16/9 un ratio mayor (como 20/9) hace que el código no se ejecute.
<code>max-device-aspect-ratio</code>	Máxima relación de aspecto, pero referida al dispositivo y no a la ventana
<code>min-aspect-ratio</code>	Relación de aspecto mínima que debe de cumplir la ventana que muestra la página
<code>min-device-aspect-ratio</code>	Relación de aspecto mínima que debe de cumplir el dispositivo que muestra la página
<code>resolution</code>	Resolución del dispositivo.
<code>max-resolution</code>	Resolución máxima del dispositivo (por ejemplo 300dpi)
<code>min-resolution</code>	Resolución mínima del dispositivo
<code>color</code>	Indica si el dispositivo tiene capacidad para mostrar colores: <code>@media print and (color){</code> /* código que se ejecuta si se imprime a color */ También permite indicar el número de bits de profundidad de colores del dispositivo:

	@media print and (color:3){ /* se ejecuta si se imprime con 3 bits de profundidad de color*/}
monochrome	Indica si el dispositivo es monocromo y asignando un número, se entenderá que se refiere al número de bits disponibles para mostrar las gamas de colores monocromos.
max-color	Máximo número de bits de profundidad para los colores del dispositivo de salida
min-color	Mínimo número de bits de profundidad para los colores del dispositivo de salida
color-index	En dispositivos con colores indexados, indica el número de colores en la tabla de índices que tiene que tener el dispositivo para que el código CSS en esta directiva funcione.
max-color-index	Máximo número de entradas de color
min-color-index	Mínimo número de entradas de color
max-monochrome	Máxima profundidad en bits de colores en una pantalla monocromo
min-monochrome	Profundidad en bits de colores siendo monocromo el dispositivo.
scan	Modo de escaneo de imágenes en el dispositivo, puede valer progressive (modo progresivo) e interlace (entrelazado)
grid	Indica si el dispositivo es táctil. Ejemplo: @media handheld and (grid)

Pueden usarse los operadores lógicos (only | not).

- not.\_ Niega todo lo que sigue.
- only (solamente). Los navegadores antiguos, que no son capaces de procesar los @media bloques, no entienden esta palabra e ignoran todo el bloque de código, sin perder el tiempo tratando de procesarlo.

Por ejemplo, si queremos dos breakpoints a 480 y 768 podríamos escribir las siguientes directivas:

```
@media only screen and (max-width:480px) {...}

@media only screen and (max-width:768px) and (min-width:481px) {...}
```

El primer bloque para los teléfonos y el segundo bloque para las tablets, donde usamos min-width a 481 para no solaparnos con el bloque anterior.

Otro ejemplo:

```
@media screen and (device-max-width:640px) and (orientation:landscape){...}
```

Haría referencia a las pantallas con menos de 640 píxeles y que están en formato horizontal.

## 4. Posicionamiento

Los elementos de una página web están contenidos en una caja rectangular, de acuerdo con el modelo de caja. La manera en que los diferentes elementos de una página web se distribuyen en la pantalla dependen del esquema de posicionamiento elegido.

Existen tres esquemas de posicionamiento: normal, flotante y absoluto.

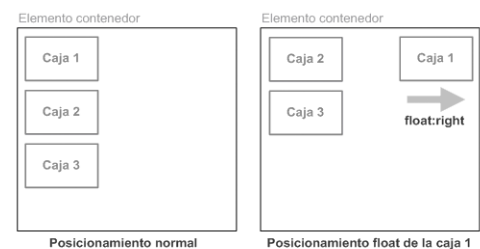
- El esquema **normal** es el que se aplica por omisión. En el esquema de posicionamiento normal, los elementos se muestran en la pantalla en el mismo orden en que se encuentran en el código fuente de la página. Los elementos pueden ser elementos de bloque o elementos en línea. Los elementos en línea están contenidos dentro de elementos de bloque y los elementos de bloque ocupan todo el ancho de la página y la altura necesaria para mostrar todo su contenido.

- El esquema **flotante** es el que permite que un elemento no ocupe todo el ancho de la pantalla y se muestre a la izquierda o a la derecha de la página con otros elementos a su lado. En el esquema de posicionamiento flotante, los elementos se muestran en la pantalla en el mismo orden en que se encuentran en el código fuente, pero pueden estar desplazados a derecha o izquierda. Para asignar el esquema de posicionamiento flotante a un elemento se utiliza la propiedad `float`.
- El esquema **absoluto** permite asignar cualquier posición a un elemento en la página (no solamente a izquierda o derecha como en el esquema flotante). En el esquema de posicionamiento absoluto, los elementos pueden mostrarse en la pantalla en un orden diferente al que se tienen en el código fuente, e incluso superponerse. Para asignar el esquema de posicionamiento absoluto a un elemento se utiliza la propiedad `position`.

## 4.1 Posicionamiento flotante

La propiedad `float` es muy útil para contenidos multimedia como imágenes o tablas. Cuando existe un elemento flotante, los elementos que se encuentran a continuación del elemento flotante fluyen a lo largo de él, salvo que haya un elemento que tenga establecido la propiedad `clear`.

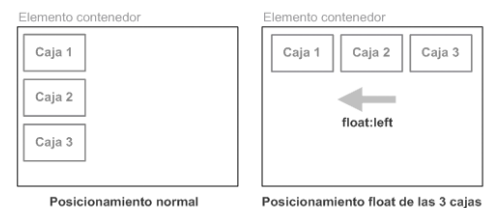
Cuando una caja se posiciona con el modelo de posicionamiento flotante, automáticamente se convierte en una caja flotante, lo que significa que se desplaza a la zona más a la izquierda o más a la derecha de la posición en la que originalmente se encontraba. En la imagen vemos el resultado de posicionar de forma flotante la caja 1 a la derecha.



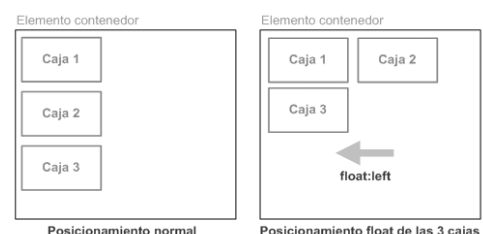
Las propiedades `float` y `clear` se pueden aplicar a cualquier elemento de una página web (para que los párrafos floten uno al lado del otro en vez de ocupar de lado a lado, es necesario asignarles tamaño con las propiedades `width` y `height`).

### Múltiples cajas flotantes

Si existen otras cajas flotantes, al posicionar otra caja flotante se tendrán en cuenta, y se respetará el sitio disponible. En el ejemplo se posicionan de forma flotante a la izquierda las tres cajas.



Las cajas no se superponen entre sí porque las cajas flotantes tienen en cuenta las otras cajas flotantes existentes. Como la caja 1 ya estaba posicionada a la izquierda, la caja 2 se colocará pegada al borde derecho de la caja 1, que es el sitio más a la izquierda posible respecto de la zona en la que se encontraba.



Si no existe sitio en la línea actual, la caja flotante se situará en la línea inferior.

### float

Esta propiedad permite posicionar de forma flotante un caja. Su valor por defecto es `none` y además puede tomar los valores `left` y `right`.

## clear

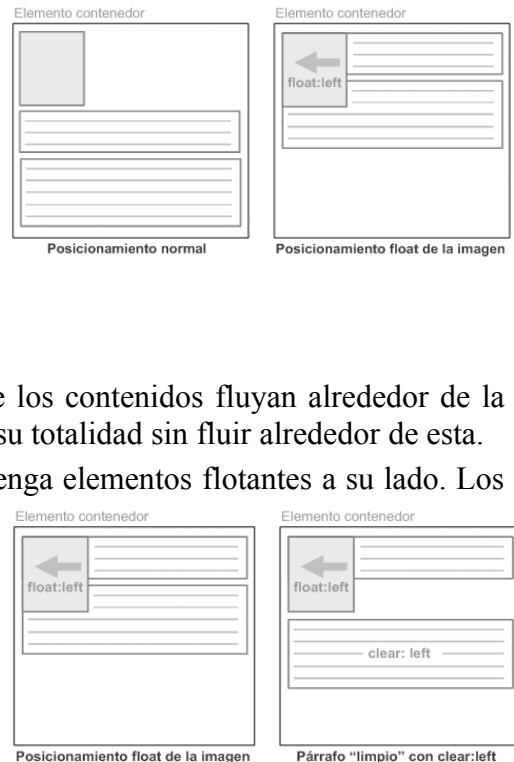
Como ya hemos dicho, los elementos que se encuentran alrededor de una caja flotante adaptan sus contenidos para fluir alrededor del elemento posicionado. Uno de los principales motivos para usar posicionamiento `float` fue la posibilidad de colocar imágenes alrededor de las cuales fluye el texto. El código asociado a la imagen anterior sería:

```
img {  
  float: left;  
}
```

En muchas ocasiones queremos que algunos de los contenidos fluyan alrededor de la imagen pero el resto queremos que se muestren en su totalidad sin fluir alrededor de esta.

La propiedad `clear` hace que un elemento no tenga elementos flotantes a su lado. Los posibles valores de `clear` son:

- `left`, que hace que no hayan elementos flotantes a la izquierda,
- `right`, que hace que no hayan elementos flotantes a la derecha,
- `both`, que hace que no hayan elementos flotantes ni a derecha ni a izquierda,
- `none`, que permite que hayan elementos flotantes a derecha y a izquierda (valor por omisión).



## 4.2 Posicionamiento absoluto

La propiedad `position` establece el esquema de posicionamiento absoluto de un elemento. Existen cuatro tipos en el esquema de posicionamiento absoluto:

### static (`position: static`)

El posicionamiento normal o estático es el modelo que utilizan los navegadores para mostrar los elementos de las páginas por defecto. En este modelo, ninguna caja se desplaza respecto de su posición original. Sólo se tiene en cuenta el tipo de elemento (block o in-line).

### relativo (`position: relative`)

El posicionamiento relativo permite desplazar una caja respecto de su posición. El desplazamiento de la caja se controla con las propiedades `top`, `right`, `bottom` y `left`.

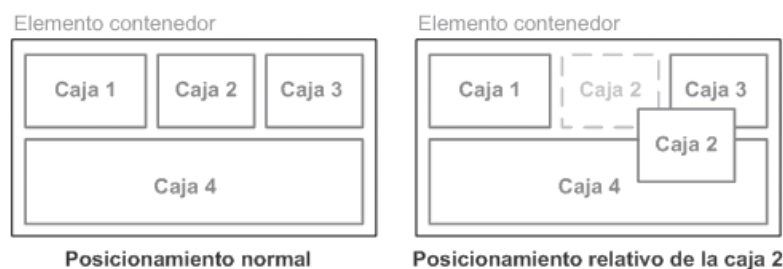
El desplazamiento vertical se puede establecer con la propiedad `top` o `bottom`:

- La propiedad `top` desplaza el elemento contando desde su borde superior. Si se da un valor positivo, el elemento se desplaza hacia abajo. Si se da un valor negativo, el elemento se desplaza hacia arriba.
- La propiedad `bottom` desplaza el elemento contando desde su borde inferior. Si se da un valor positivo, el elemento se desplaza hacia arriba. Si se da un valor negativo, el elemento se desplaza hacia abajo.

El desplazamiento horizontal se puede establecer con la propiedad `left` o `right`:

- La propiedad `left` desplaza el elemento contando desde su borde izquierdo. Si se da un valor positivo, el elemento se desplaza hacia la derecha. Si se da un valor negativo, el elemento se desplaza hacia la izquierda.
- La propiedad `right` desplaza el elemento contando desde su borde derecho. Si se da un valor positivo, el elemento se desplaza hacia la izquierda. Si se da un valor negativo, el elemento se desplaza hacia la derecha.

El desplazamiento de una caja no afecta al resto de cajas adyacentes, que se muestran en la misma posición que si la caja desplazada no se hubiera movido de su posición original.



La caja 2 se desplazó a la derecha y abajo. Como el resto de las cajas de la página no modifican su posición, se producen solapamientos entre los contenidos de las cajas.

O único problema de posicionar elementos de forma relativa é que se poden producir solapamientos con outros elementos da páxina.

#### fijo (`position:fixed`)

Un elemento `fixed` (fijo) se posiciona de manera relativa a la ventana del navegador, lo que significa que se mantendrá en el mismo lugar incluso después de hacer scroll en la página. Al igual que con `relative`, se establece su posición con las propiedades `top`, `right`, `bottom`, y `left`.

Un elemento posicionado con `fixed` no deja espacio en el lugar de la página donde estaba ubicado normalmente.

#### absoluto (`position:absolute`)

Un elemento posicionado con valor `absolute`, se comporta como `fixed` pero es relativo a su ancestro posicionado más cercano en lugar de ser relativo a la ventana del navegador. Si un elemento con `position: absolute;` no tiene ancestros posicionados, usará el elemento `body` del documento, y se seguirá moviendo al hacer scroll en la página.

*Un elemento "posicionado" es aquel cuyo valor es cualquiera excepto `static`.*

Al igual que con `relative` y `fixed`, se establece su posición con las propiedades `top`, `right`, `bottom`, y `left`.

Cuando una caja se posiciona de forma absoluta, el resto de elementos de la página la ignoran y ocupan el lugar original ocupado por la caja posicionada. Al igual que en el posicionamiento relativo, cuando se posiciona de forma absoluta una caja es probable que se produzcan solapamientos con otras cajas. En el ejemplo, se posiciona de forma absoluta la caja



2:

La caja 2 está posicionada de forma absoluta, lo que implica que el resto de elementos ignoran que esa caja existe. Por este motivo, la caja 3 deja su lugar original y pasa a ocupar el hueco dejado por la caja 2.

Si queremos posicionar un elemento de forma absoluta respecto a su elemento contenedor, es imprescindible posicionar el elemento contenedor. Para ello, solo es necesario añadir la propiedad `position: relative`, por lo que no es obligatorio desplazar el elemento contenedor respecto de su posición original.

## 4.3 Superposición de cajas

Además de posicionar una caja de forma horizontal y vertical, CSS permite controlar la posición tridimensional de las cajas posicionadas, determinando el orden de superposición de estas. Así, podemos indicar qué cajas van a ir delante o detrás de otras. Esto es útil cuando se producen solapamientos, y se hace utilizando la propiedad `z-index`.

El valor más común de la propiedad `z-index` es un número entero. Aunque la especificación oficial permite los números negativos, en general se considera el número 0 como el nivel más bajo. Por lo tanto, cuanto más alto sea el valor más cerca del usuario se mostrará la capa (una capa con valor `z-index:10` se visualizará por encima de otra con valor `z-index:9`).

La propiedad `z-index` solo tiene efectos en los elementos posicionados, por lo que es obligatorio que la propiedad `z-index` vaya acompañada de la propiedad `position`.

## 4.4 La propiedad display

`display` es la propiedad más importante para controlar estructuras. Cada elemento tiene un valor de `display` por defecto dependiendo de qué tipo de elemento sea. El valor por defecto para la mayoría de los elementos es usualmente `block` (de bloque) o `inline` (en línea).

Un elemento en bloque ocupa todo el espacio de su elemento padre (contenedor), creando así un "bloque". Los navegadores suelen mostrar el elemento a nivel de bloque con un salto de línea antes y después del elemento. Elementos de bloque son por ejemplo: `main`, `header`, `footer`, `aside`, `section`, `article`, `div`, `p`, los títulos (del `h1`, al `h6`), `ul`, `ol` o `li`.

Un elemento en línea ocupa sólo el espacio delimitado por las etiquetas que lo definen. Ejemplo de elementos en línea son: `em`, `strong`, `time`, `a`, `img`, `span`, `input`, `button` o `select`.

Es posible cambiar el modo en el que se visualiza un elemento HTML de una página usando la propiedad `display`. Algunos de sus posibles valores son:

- `none`: El elemento desaparece de la página y no ocupa sitio.
- `block`: Hace que el elemento se comporte como un elemento en bloque. No admite otro elemento a su lado.
- `inline`. El elemento se comporta como un elemento en línea.
- `inline-block`. El elemento tendrá un comportamiento mezcla entre los dos anteriores. Los elementos `inline-block` fluyen con el texto y demás elementos como si fueran elementos en-línea y además respetan el ancho, el alto y los márgenes verticales.

- `run-in`. El elemento se comporta como `block` o como `inline` dependiendo del contexto.
- `list-item`. El elemento se considerará un ítem de una lista (al igual que la etiqueta `li`). Por lo tanto podemos aplicar las propiedades CSS de las listas como `list-style-type` a los elementos así marcados.
- `table`. Considera que el elemento contiene una tabla.
- `table-caption`. Considera al elemento un título de tabla.
- `table-cell`, Considera al elemento una celda de una tabla
- `table-row` Considera al elemento una fila de una tabla
- `table-row-group`: Considera al elemento un grupo de filas (como `tbody`).
- `table-column`: Considera al elemento una columna (como `col`).
- `table-column-group`: Considera al elemento un grupo de columnas (como `colgroup`).
- `table-footer-group`: Considera al elemento un pie de tabla (como `tfoot`).
- `table-header-group`: Considera al elemento una cabecera de tabla (como `thead`).
- `inline-table`: Elemento `inline` interior a una tabla.

También tenemos otros valores que nos permiten maquetar la página y que veremos más adelante como `flex` o `grid`.

## 4.5 La propiedad `visibility`

`visibility` es una propiedad que únicamente permite hacer visibles o invisibles los elementos de una página. Puede tomar los siguientes valores:

- `visible` es el valor por defecto.
- `hidden` hace que la caja sea invisible para que no se muestren sus contenidos. El resto de elementos de la página se situará como si la caja fuese visible, por lo que en el lugar que ocuparía se verá un hueco.
- `Collapse` \_ esconde filas o columnas de una tabla.

## 4.6 Contenido desbordado. Propiedad `overflow`.

Normalmente, los contenidos de un elemento se pueden mostrar en el espacio reservado para ese elemento. Pero en algunas ocasiones el contenido de un elemento no cabe en el espacio reservado para este y se desborda.

La situación más habitual en la que el contenido sobresale del espacio reservado es cuando se establece la anchura y/o altura del elemento mediante la propiedad `width` y/o `height`.

CSS define la propiedad `overflow` para controlar el modo en que se visualizan los contenidos que sobresalen (desbordados) del elemento. Puede tomar los siguientes valores:

- `visible`: El contenido se muestra aun cuando rebase el tamaño previsto. Es el valor por defecto.
- `hidden`: El contenido desbordado se oculta. Sólo se ve la parte que cabe dentro de la zona reservada para el elemento.





- `scroll`: Se visualiza el contenido que cabe dentro de la zona reservada para el elemento y aparece una barra de scroll que permite ver el resto de contenido.
- `auto`: El comportamiento depende del navegador. Por lo general es equivalente al valor `scroll`.

Podemos tratar de forma independiente el desbordamiento horizontal o vertical usando las propiedades `overflow-x` y `overflow-y`. Los valores a aplicar son los mismos.

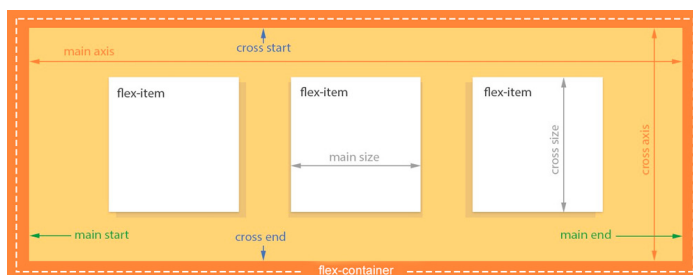
## 4.7 Flexbox. Cajas flexibles

Uno de los aspectos más frustrantes que se encuentra uno en el mundo del diseño web es el relacionado con todo aquello de reordenar, colocar y distribuir diferentes elementos dentro de una web. Flexbox ( o el módulo de cajas flexibles) es probablemente uno de los más completos y eficaces módulos de maquetación. Todo lo que era complicado en versiones anteriores de CSS ( como centrar verticalmente o diseñar estructuras que se redimensionen con elegancia ) con flexbox es ya una tarea muy fácil.

Los elementos flex tienen la capacidad de redimensionarse y recolocarse dentro de la caja flex con facilidad. También tienen la capacidad de alinearse tanto horizontalmente como verticalmente y todo esto puede ser muy interesante a la hora de diseñar páginas web adaptativas.

Flexbox representa un modelo básico de maquetación que supone la existencia de una caja padre llamada **contenedor flexible (flex container)** o **caja flex**. Los hijos situados inmediatos serán los elementos flexibles (flex ítems) o **ítems flex**.

En la imagen vemos como la W3C nos presenta gráficamente el modelo flexbox con sus diferentes propiedades y su pensado funcionamiento.



El conjunto de propiedades que nos presenta la especificación CSS3 en cuanto al módulo flexbox las podemos dividir en dos grupos, las propiedades para el elemento contenedor flex y las otras propiedades para los elementos hijos flexibles.

Son propiedades del contenedor flex: `display`, `flex-direction`, `flex-wrap`, `flex-flow`, `align-items`, `justify-content`, `align-content`.

Son propiedades de los elementos flex (items): `align-self`, `flex-grow`, `flex-shrink`, `flex-basis`, `flex`, `order`.

### `display: flex`

Estableciendo la propiedad `display` a `flex` configuramos el contenedor principal y de esta manera todos sus hijos inmediatos se convertirán en elementos flexibles de forma automática.

```
.flex-container {
    display: flex;
}
```

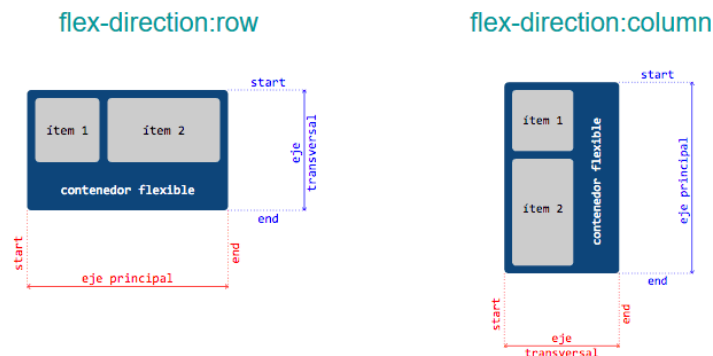
### `flex-direction`

Es una propiedad del contenedor flex.

En una caja flex podemos colocar los elementos en cualquier dirección:

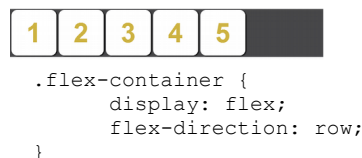


horizontalmente (`row`) o verticalmente (`column`), en el sentido lógico o en sentido contrario (`-reverse`). Para hacerlo utilizamos la propiedad `flex-direction`, que establece cual es el eje principal de la caja y por lo tanto la dirección de los elementos hijos.

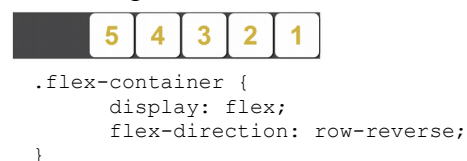


La propiedad `flex-direction` puede tomar una de estas valores:

- `row` (valor por defecto). Establece el eje horizontal como eje principal ( *main axis* ), y el eje vertical como eje transversal ( *cross axis* ). Los flex items se apilan en una fila de izquierda a derecha.



- `row-reverse`. Igual que el anterior pero coloca los elementos en sentido contrario.



- `column`. El eje vertical es el eje principal ( *main axis* ) mientras que el eje horizontal es el eje transversal ( *cross axis* ). Los flex items se apilan en una columna de arriba hacia abajo.

```

.flex-container {
  display: flex;
  flex-direction: column;
}

```



- `column-reverse`. Igual que el anterior pero coloca los elementos de abajo hacia arriba.

```

.flex-container {
  display: flex;
  flex-direction: column-reverse;
}

```



## flex-wrap

Es una propiedad del contenedor flex.

Especifica si puede haber un cambio de línea ( `wrap` ) o no ( `nowrap` ). Puede tomar los siguientes valores:

- nowrap (valor por defecto). Los elementos encajen en el ancho del contenedor flexible aun modificando la apariencia de estos. Hay ocasiones en las que puede provocar desbordamiento.



- wrap . Puede haber cambio de línea. Los elementos flex aparecen colocados en varias líneas.



- wrap-reverse. Igual que wrap pero los elementos aparecen ordenados en sentido contrario.



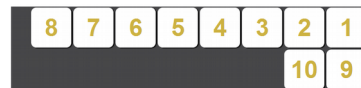
## flex-flow

Es una propiedad del contenedor flex.

Permite abreviar las propiedades: flex-direction y flex-wrap (opcional) en una sola. Valor por defecto es row nowrap.

Ejemplo:

```
.flex-container {
  display: flex;
  flex-flow: row-reverse wrap;
}
```



## justify-content

Es una propiedad del contenedor flex.

Permite controlar el alineamiento de los elementos de una caja flexible a lo largo de su eje principal. Puede tomar los siguientes valores:

- flex-start. (valor por defecto) Los elementos aparecen agrupados al principio (start) del eje principal.

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



- flex-end. Los elementos aparecen agrupados al final (end) del eje principal.

```
.flex-container {
  display: flex;
  justify-content: flex-end;
}
```



- center. Los elementos aparecen agrupados en el centro (center) del contenedor.

```
.flex-container {
  display: flex;
  justify-content: center;
}
```



- space-between. Los elementos aparecen distribuidos dejando la misma distancia entre ellos dentro del contenedor y quedando el primer y último elementos alineados al borde de este.

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



- space-around. Los elementos aparecen distribuidos uniformemente y con un espacio igual entre ellos.

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



## align-items

Es una propiedad del contenedor flex.

Permite controlar el alineamiento de los elementos de una caja flexible a lo largo de su eje transversal. Puede tomar los siguientes valores:

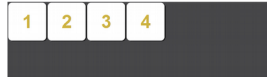
- stretch. (valor por defecto) Los elementos aparecen estirados para ocupar todo el espacio.

```
.flex-container {
  display: flex;
  align-items: stretch;
}
```



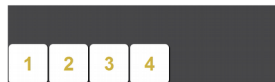
- flex-start. Los elementos aparecen agrupados al principio (start) del eje transversal.

```
.flex-container {
  display: flex;
  align-items: flex-start;
}
```



- flex-end. Los elementos aparecen agrupados al final (end) del eje transversal.

```
.flex-container {
  display: flex;
  align-items: flex-end;
}
```



- center. Los elementos aparecen agrupados en el centro (center) de la caja.

```
.flex-container {
  display: flex;
  align-items: center;
}
```



- baseline. Los elementos aparecen alineados a su línea de base (baseline). Esta se define en base a los textos de cada ítem.

```
.flex-container {
  display: flex;
  align-items: baseline;
}
```



## align-content

Es una propiedad del contenedor flex.

Podemos controlar el alineamiento de los elementos de una caja flexible ( flexbox ) a lo largo de su eje principal con justify-content, o a lo largo de su eje transversal con align-items.

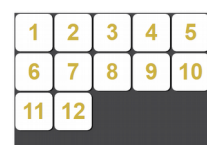
Pero, a veces, los elementos de la caja flex pueden ocupar varias líneas (flex-wrap:wrap). Y es en este caso cuando usaremos esta propiedad para controlar el alineamiento de los elementos. Puede tomar los siguientes valores:

- stretch. (valor por defecto) Los elementos aparecen estirados para ocupar todo el espacio.

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: stretch;
}
```



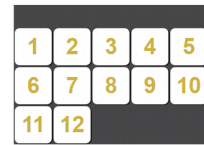
- flex-start. (valor por defecto) Los elementos aparecen agrupados al principio (start) del eje principal.



```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: flex-start;
}
```

- **flex-end.** Los elementos aparecen agrupados al final (end) del eje principal.

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: flex-end;
}
```



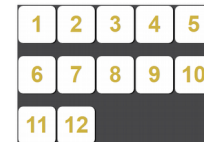
- **center.** Los elementos aparecen agrupados en el centro (center) del contenedor.

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: center;
}
```



- **space-between.** Los elementos aparecen distribuidos dejando la misma distancia entre ellos dentro del contenedor y quedando el primer y último elementos alineados al borde de este en el eje principal.

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: space-between;
}
```



- **space-around.** Los elementos aparecen distribuidos uniformemente y con un espacio igual entre ellos con respecto al eje principal.

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: space-around;
}
```



## align-self

Es una propiedad de los ítems (elementos) del contenedor.

Reposiciona elementos individuales con respecto al eje transversal de la caja. Generalmente se trata de elementos posicionados con align-items. Puede tomar los siguientes valores:

- **stretch.** (valor por defecto) El elemento aparece estirado para ocupar todo el espacio.

```
.flex-container {
  display: flex;
  align-items: center;
}
.desplazado {align-self: stretch;}
```



- **flex-start.** El elemento aparece al principio (start) del eje transversal.

```
.flex-container {
  display: flex;
  align-items: center;
}
.desplazado {align-self: flex-start;}
```



- **flex-end.** El elemento aparece al final (end) del eje transversal.

```
.flex-container {
  display: flex;
  align-items: center;
}
.desplazado {align-self: flex-end;}
```



- **center.** El elemento aparece en el centro (center) de la caja.

```
.flex-container {
  display: flex;
}
```



```

    align-items: flex-start;
  }
  .desplazado {align-self: center;}

```

- **baseline.** El elemento aparece alineado a su línea de base (baseline).

```

.flex-container {
  display: flex;
  align-items: center;
}
.desplazado {align-self:baseline;}

```

## flex-grow

Es una propiedad de los ítems (elementos) del contenedor.

Establece cuanto puede crecer un elemento flex en relación al resto de elementos de la misma caja flex. Su valor es un número. Por defecto toma el valor 0 que indica que el elemento no puede crecer.

Si todos los ítems de una caja tienen el mismo valor de `flex-grow`, por ejemplo `flex-grow:1`; quiere decir que todos los ítems tienen que crecer en igual proporción, hasta ocupar todo el espacio disponible.

Por ejemplo: en el siguiente ejemplo tenemos 5 cajas y sobra espacio. Para rellenarlo hacemos que dos cajas que crezcan ocupando una parte del espacio (`flex-grow:1`) y otra tres partes (`flex-grow:3`) hasta ocupar todo el espacio.

```

.crece {flex-grow:1;}
.crece3 {flex-grow:3;}

```



## flex-shrink

Es una propiedad de los ítems (elementos) del contenedor.

Establece cuanto puede disminuir un elemento flex en relación al resto de elementos de la misma caja flex. Su valor es un número. Por defecto toma el valor 1, lo que quiere decir que los elementos de una caja flex disminuirán en igual proporción.

Por ejemplo: en el siguiente ejemplo tenemos 5 cajas y le asignamos un ancho del 30% a cada una (`width:30%`). Para rellenarlo hacemos que dos cajas se reduzcan con (`flex-grow:3`) y otra tres partes (`flex-grow:3`). El resultado es:

```

.crece {flex-shrink:2;}
.crece3 {flex-shrink:3;}

```



## flex-basis

Es una propiedad de los ítems (elementos) del contenedor.

Indica el valor inicial del tamaño principal de un elemento flex, antes de que esté redimensionado con `flex-grow` o `flex-shrink`.

El valor a establecer será la anchura (`width`) en los contenedores horizontales (`flex-direction:row`) y la altura (`height`) en los contenedores verticales horizontales (`flex-direction:column`)

El valor por defecto es `auto`.

Por ejemplo: si tenemos 5 cajas con un ancho del 0% y establecemos `flex-basis` a 25% para las dos primeras, estas se redimensionarán para ocupar cada una el 25% del espacio.



## flex

Es una propiedad de los ítems (elementos) del contenedor.

Permite especificar en una sola propiedad los valores de `flex-grow`, `flex-shrink` (opcional) y `flex-basis` (opcional). Los valores por defecto son 0 1 y auto.

```
flex: 0 1 auto;
```

## order

Es una propiedad de los ítems (elementos) del contenedor.

Por defecto los elementos flex aparecen en el mismo orden que en el código. Podemos alterar el orden de los elementos flex utilizando la propiedad `order`.

El valor por defecto es 0 y permite números positivos y negativos. Los elementos flex con el mismo valor, aparecen en el mismo orden que en el código.

Por ejemplo, si a la caja número 4 le damos el valor -1 en esta propiedad se pondrá delante de todos.

```
.delante {order:-1;}
```



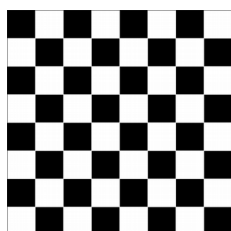
## 4.8 Grid Layout

Grid Layout no es algo que venga a sustituir a Flexbox. De hecho, si queremos controlar de forma precisa la maquetación de nuestra web, es importante conocer ambos, y usarlos indistintamente según nuestras necesidades.

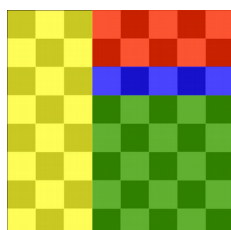
El grid layout es un sistema bidimensional, que transforma un elemento HTML en una cuadrícula. Los elementos hijos de este pueden ser posicionados dentro de las celdas de esta cuadrícula en base a estas filas y columnas que hemos creado, en términos de tamaño, posición y orden.

Tal y como ocurre en Flexbox, CSS Grid funciona con la idea de un contenedor-padre que alberga unos elementos-hijo. Por lo que nuestro contenedor-padre es un `grid-container`, y nuestro elemento-hijo es un `grid-item`. Vamos a verlo con un ejemplo:

Imaginemos un tablero de ajedrez:



Podríamos marcar como queremos colocar la estructura de nuestros divs.



Ahora lo implementamos en HTML/CSS partiendo de Grid. Para el siguiente html:

```
<div id="tablero">
  <div id="azul"></div>
```

```

        <div id="amarillo"></div>
        <div id="rojo"></div>
        <div id="verde"></div>
    </div>

```

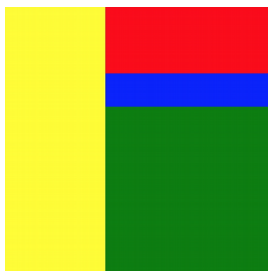
Le aplicamos el siguiente css

```

#tablero {
    display: grid;
    grid-template-columns: 300px 500px;
    grid-template-rows: 200px 100px 500px;
    width: 800px;
    height: 800px;
}
#azul {
    background-color: blue;
    grid-column: 2 / 3;
    grid-row: 2 / 3;
}
#amarillo {
    background-color: yellow;
    grid-column: 1 / 2;
    grid-row: 1 / 4;
}
#rojo {
    background-color: red;
    grid-column: 2 / 3;
    grid-row: 1 / 2;
}
#verde {
    background-color: green;
    grid-column: 2 / 3;
    grid-row: 3 / 4;
}

```

Y obtendríamos como resultado:



Con muy pocas líneas de CSS, y sin alterar la estructura del HTML, hemos conseguido estructurar a nuestro gusto y sin limitaciones aparentes. Veamos poco a poco como funciona.

### Conceptos básicos

Para empezar debemos tener claro que existen dos partes bien diferenciadas: un padre y sus hijos. El padre será el tablero o la etiqueta que envuelva todo. Llevará el estilo `display: grid`. Y los hijos son los contenedores o elementos que queremos posicionar dentro de nuestro tablero. El funcionamiento es similar a `flex`. Un ejemplo, si al html anterior le aplicamos el siguiente css donde para el contenedor indicamos este tipo de

posicionamiento, y para los hijos sólo cambiamos el color, lo veríamos como en la imagen:

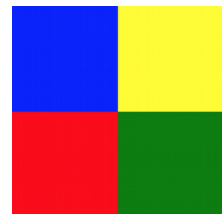
```
#tablero {
    display: grid;
    width: 800px;
    height: 800px;
}
#azul { background-color: blue; }
#amarillo { background-color: yellow; }
#rojo { background-color: red; }
#verde { background-color: green; }
```



Para indicar a cada hijo donde debe colocarse:

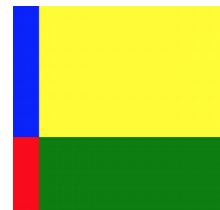
- El padre indica cual va a ser el número de filas y columnas que va a tener el tablero. Incluso va a decir cuánto mide cada una. Por ejemplo, si se quiere una estructura de 2x2 y nuestro tablero mide 800px por 800: 2 columnas de 400px y 2 filas de 400px.

```
#tablero {
    display: grid;
    // Columnas
    grid-template-columns: 400px 400px;
    // Filas
    grid-template-rows: 400px 400px;
    width: 800px;
    height: 800px;
}
```



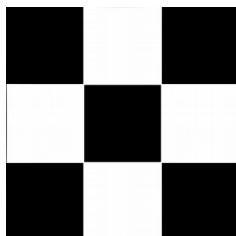
- Podemos cambiar los tamaños de las celdas. Por ejemplo:

```
#tablero {
    display: grid;
    // Columnas
    grid-template-columns: 100px 700px;
    // Filas
    grid-template-rows: 500px 300px;
    width: 800px;
    height: 800px;
}
```

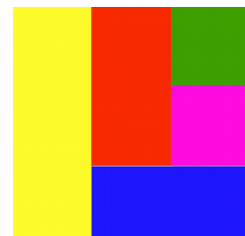


## Posicionar a los hijos

Vamos a ver como posicionar a cada hijo donde queramos con un tablero de 3x3



Donde queremos colocar 5  
elementos con esta estructura



En este caso nuestro html es:

```
<div id="tablero">
    <div id="azul"></div>
    <div id="amarillo"></div>
```



```

        <div id="rojo"></div>
        <div id="verde"></div>
        <div id="rosa"></div>
    </div>

```

El contenedor lo definimos en css:

```

#tablero {
    display: grid;
    grid-template-columns: 300px 300px 300px;
    grid-template-rows: 300px 300px 300px;
    width: 900px;
    height: 900px;
}

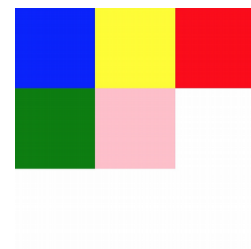
```

Si solo indicamos el color en css el tablero se vería:

```

#azul {
    background-color: blue;
}
#amarillo {
    background-color: yellow;
}
#rojo {
    background-color: red;
}
#verde {
    background-color: green;
}
#rosa {
    background-color: pink;
}

```



Cada hijo debe ser posicionado en un inicio y un final de columna y fila. Nuestro tablero tiene las siguientes líneas.

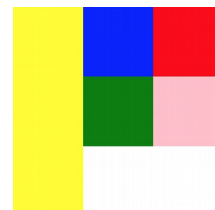
	1	2	3	4
1	yellow	red	green	
2	yellow	red	pink	
3	yellow	blue	blue	
4				

Empezamos con el amarillo. Ocupa el área entre la primera y la segunda columna de la fila 1 a la 4.

```

#amarillo {
    background-color: yellow;
    grid-column-start: 1;
    grid-column-end: 2;
    grid-row-start: 1;
    grid-row-end: 4;
}

```



Una versión resumida de lo anterior:

```

#amarillo {
    background-color: yellow;
    grid-column: 1 / 2;
}

```

```

        grid-row: 1 / 4;
    }

```

Del mismo modo, los otros colores quedarían:

```

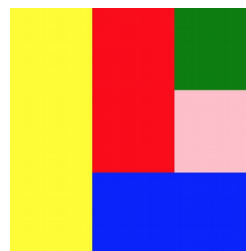
#azul {
    background-color: blue;
    grid-column: 2 / 4;
    grid-row: 3 / 4;
}

#rojo {
    background-color: red;
    grid-column: 2 / 3;
    grid-row: 1 / 3;
}

#verde {
    background-color: green;
    grid-column: 3 / 4;
    grid-row: 1 / 2;
}

#rosa {
    background-color: pink;
    grid-column: 3 / 4;
    grid-row: 2 / 3;
}

```



### Fracción de grid

Aunque podemos utilizar cualquier otras unidades para longitud disponibles en CSS ( px, %, em, rem ...etc ), es importante saber que el grid tiene una unidad específica: `fr` que representa una fracción del espacio disponible dentro del contenedor grid. Por ejemplo: si queremos una estructura de 3 columnas iguales podemos escribir:

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

Si queremos que la del medio sea el doble que las otras podríamos poner:

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

### grid-gap

Esta propiedad establece el tamaño del espacio que tiene que haber entre columnas, si no queremos que estas aparezcan pegadas. Puede definirse por separado para filas y columnas con `grid-rows-gap` y `grid-column-gap`.



Haz las tareas 2 y 3 de la hoja de tareas.

## 5. Más propiedades CSS

### 5.1 Propiedades de fuentes y texto

Con CSS3 aparecen nuevas propiedades que podemos aplicar a las fuentes, aunque no todas son soportadas por todos los navegadores. Vamos a ver algunas:

## font-variant-position

Esta propiedad permite colocar el texto como superíndice o subíndice. Únicamente está soportada por últimas versiones de Firefox. Sus posibles valores son:

- normal
- sub → Subíndice
- super → Superíndice

## text-overflow

Indica que hacer con el texto cuando este está dentro de un contenedor que no tiene el tamaño suficiente. Puede tomar los siguientes valores:

- clip → Únicamente se ve el texto que cabe en la capa, el resto no se muestra.
- ellipsis → Sólo se visualiza lo que cabe pero al final del texto recortado pone unos puntos suspensivos.
- Se puede indicar una cadena que será lo que muestra cuando ya no cabe más texto.  
`text-overflow: " [y más..]";`

## word-wrap

Esta propiedad permite partir el texto cuando la palabra es muy larga y no cabe. Para ello debe ponerse el valor `break-word`.

### 5.1.1 Sombreado de texto

Es muy fácil aplicar sombras a textos con CSS3. La propiedad `text-shadow` de CSS le permite crear una copia un poco borrosa, ligeramente desplazada del texto, que termina pareciéndose a una sombra del mundo real. Según la especificación CSS3, la propiedad `text-shadow` puede tener los siguientes valores:

`none | [<shadow>,* <shadow>],`

`<shadow>` se define como:

`[<color>? <length> <length> <length>? | <length> <length> <length>? <color>? ],`

donde los dos primeros representan la longitud horizontal y vertical y la tercera un radio opcional de difuminado. Vamos a ver algunos ejemplos.

Podemos hacer una simple sombra de esta forma, por ejemplo:

`text-shadow: 2px 2px 3px #000;`

#### Una sombra simple

Si no queremos que se vea difuminado bastaría con poner a 0 la opción de difuminado, y variando el color se vería:

`text-shadow: 2px 2px 0 #888;`

#### No me gusta el difuminado

Podemos hacer que el texto se vea borroso:

`text-shadow: 0px 0px 5px #000;`

#### Texto borroso

Por último, puede añadir "más de una sombra", lo que permite crear efectos especiales como:

`text-shadow: 0 0 4px blanco, 0-5px 4px #FFFFFF33, 2px-10px 6px #FFDD33,  
-15px 11px-2px #FF8800, 25px 18px-2px #FF2200`

## Múltiples sombras al fuego

Al igual que todas las propiedades CSS se puede modificar *text-shadow* sobre la marcha utilizando JavaScript.

Existen muchas ventajas en la utilización de texto en lugar de imágenes: No utilizar imágenes disminuye el consumo de ancho de banda y de conexiones HTTP. Se mejora la accesibilidad, tanto para las personas que utilizan lectores de pantalla como para los motores de búsqueda. Y el zoom de la página funcionará mejor porque el texto se puede ampliar en lugar de utilizar la interpolación de píxeles para aumentar la escala de una imagen.

### 5.1.2 Sombreado de cajas

Es similar al sombreado del texto, pero tiene unas pequeñas diferencias ya que además del desplazamiento horizontal y vertical, desenfoque y color podemos indicar la propagación y el sombreado interior. Sintaxis:

**box-shadow:** desplazamientoH desplazamientoV desenfoque propagación color inset;

La *propagación* es la “fuerza” de la sombra, a mayor fuerza, más contraste tendrá y sus bordes invadirán más al resto de elementos. Por defecto vale 0.

El sombreado interior (inset) hace que la sombra se muestre hacia el interior de la caja, haciendo un efecto de hundimiento de esta.



Haz la tarea 4 de la hoja de tareas.

## 5.2 Background en CSS3

Las diferencias más significativas que ofrece la propiedad background en CSS3 son:

- La posibilidad de colocar varias imágenes de fondo
- El tamaño de nuestro fondo
- La posición de estas imágenes.

Vamos a ver un ejemplo aplicando unos estilos a la etiqueta div

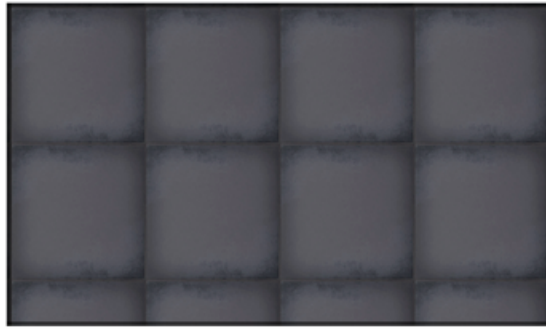
```
div{
  width:600px;
  height:350px;
  border: 5px solid hsla(30, 8%, 5%, .5); }
```

Ahora añadiremos a nuestro background una imagen cuyas medidas no son múltiplos de las de nuestro DIV con lo cual quedaría:

```
div {
  width:600px;
  height:350px;
  border: 5px solid hsla(30, 8%, 5%, .5);

  background-image:url(azulejo.png);
  /*Tamaño de la imagen 125 x 125*/
}
```

con lo cual se vería como se muestra a continuación, ya que la imagen de los azulejos no es múltiplo de nuestro div y los azulejos son cortados en la parte inferior. Pudiendo ser cortados también en la parte de la derecha.



Para evitar esto que nos ha sucedido, deberíamos emplear `Background-size` asignándole dos valores, uno para el ancho y otro para el alto. Si queremos mantener las proporciones en la imagen o ratio debemos poner uno de los dos valores auto y que se redimensione en función a su otro valor en px. Si dejamos el segundo valor en blanco el navegador lo interpretará como auto también. Así, nuestra imagen se adapta al tamaño del background.

```
div {
  width:600px;
  height:350px;
  border: 5px solid hsla(30, 8%, 5%, .5);

  background-image:url(azulejo.png);
  /*Tamaño de la imagen 125 x 125*/

  background-size: 50px auto;
}
```

De este modo, la imagen de los azulejos se adaptó perfectamente a nuestro div sin perder las proporciones y sin que nuestra imagen haya sido cortada por ningún lado.



Como unidad de medida hemos utilizado **px** haciendo referencia al tamaño que queremos para nuestra imagen de fondo. También podemos emplear tantos por ciento. Éstos hacen referencia al tamaño del contenedor. Si cambiamos los valores de `background-size`

```
background-size: 50% 100%;
```

se vería:



## background-clip

Esta propiedad especifica hasta donde se extiende el fondo (imagen o color) de un elemento. Por defecto, el fondo ( `background` ) se extiende hasta el límite externo del borde. Sus posibles valores son:

- `border-box` → Valor por defecto. El fondo se extiende hasta el límite externo del borde.
- `padding-box` → El fondo se extiende hasta el límite externo del relleno (`padding`).
- `content-box` → El fondo se dibuja en la zona de contenido: dónde va el texto del elemento.

Con esta propiedad podemos hacer bordes transparentes. Para ello debemos usar un color `rgba`. Con la siguiente línea creamos un borde verde con una opacidad del 50%.

```
border: 20px solid rgba(0, 190, 0, 0.5);
```

Aunque sólo con esto, si la caja tiene fondo no conseguimos que se vea transparente ya que el fondo se extiende hasta el borde. Estableciendo la propiedad `background-clip` a `padding-box` hacemos que el fondo sólo se extienda hasta el límite del relleno. Por ejemplo, con el siguiente `css` veríamos:

```
.transparente{
    background:#FFF;
    border: 20px solid rgba(0, 190, 0, 0.5);
    padding: 10px 10px 0 10px;
    position:absolute;
    top:15px;
    left:15px;
    background-clip: padding-box; }
```

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo más vaca que carnero, salpicón, algunas lentejas los viernes, algún palomino de añadidura los sábados, consumían las tres partes de su hacienda. El resto della concluían sayo de velarte, calzas de velludo para las fiestas con sus pantuflos de lo mismo, los días de entre semana se honraba con su vellori de lo más fino. Tenía en su casa una ama que pasaba de los cuarenta, y una sobrina que no llegaba a los veinte, y un mozo de campo y plaza, que así ensillaba el rocín como tomaba la podadera.

### 5.2.1 Background con varias imágenes

Podemos insertar varias imágenes de fondo fácilmente, empleando el mismo elemento `background` y separando las imágenes por comas. Cada una de estas imágenes podrá ser posicionada, repetida y dimensionada, pudiendo aplicar a éstas las mismas variaciones que cuando trabajamos con una sola imagen.

Vamos a colocar dos imágenes encima del azulejo. Las imágenes se muestran en orden inverso a su colocación. Es decir, la imagen de azulejos que tenemos intención que vaya detrás del todo hay que colocarla en nuestro código la última, de la misma forma con las demás.

Como hay navegadores que no soportan este tipo de `background`, es recomendable colocar un estilo `background` antes del de múltiples imágenes. De esta forma si el navegador no reconoce `background` para varias imágenes habrá tomado el estilo definido una línea anterior. Así, en nuestro ejemplo, si queremos incluir una imagen de un príncipe y otra de una rana sobre el azulejo:

```
div {
    width:600px;
    height:350px;
    border: 5px solid hsla(30, 8%, 5%, .5);

    background-size: 50% 100%;

    /*Para los navegadores que no soportan varias imágenes*/
    background:url(azulejo.png);

    /*En caso de soportar varias imágenes sustituye el estilo por este*/
```

```
background: url(principe.png),
            url(rana.png),
            url(azulejo.png);
}
```

Donde la propiedad `background-size` se la aplica a todas las imágenes:



Ahora añadiremos valores de posicionamiento para colocar la rana y el príncipe cada uno en un azulejo y al mismo tiempo le vamos a decir que no se repita la imagen. Estos valores de posicionamiento pueden ser px y tantos por ciento %.

```
background:url(principe.png) 0% 5px no-repeat,
            url(rana.png) 100% 5px no-repeat,
            url(azulejo-pequeno.png);
```

Al príncipe le indicamos que se sitúe en el eje X a 0% y a 5 px en el eje Y. Del mismo modo colocamos la rana.



## background-size

Esta propiedad permite redimensionar las imágenes de fondo.

Se puede indicar el tamaño de la imagen redimensionada:

```
background-size: tamañoX [tamañoY];
```

El segundo valor es opcional: si no se indica la imagen será redimensionada proporcionalmente.

`contain` → La caja contiene la imagen. La imagen aparece integralmente en la caja y es redimensionada proporcionalmente al tamaño máximo: o bien es tan alta como la caja, o es tan ancha como la caja.

`cover` → La imagen es redimensionada proporcionalmente y cubre toda la caja.

`auto` → La palabra clave `auto`: redimensiona la imagen guardando las proporciones. Si tanto el valor de `tamañoX` como el de `tamañoY` son `auto`, la imagen conserva su tamaño inicial. Este es el valor por defecto.

Por último, podríamos indicar un tamaño para cada una de nuestras imágenes en la propiedad `background-size`. El orden para dar estos valores es de arriba hasta abajo en la lista de imágenes y se aplicará de la siguiente forma, quedando nuestra imagen:

```
background-size: auto, auto, 50% 100%;
```



Con el valor auto le estamos indicando que tome el valor original de la imagen para principe.png y rana.png.



Haz la tarea 5 de la hoja de tareas

## 5.3 Bordes redondeados

CSS 3 incorpora nuevas propiedades para el control de bordes de los elementos. La propiedad `border-radius` permite definir bordes redondeados en las esquinas, especificando las medidas del radio que deben darse a la curva de las esquinas. Por ejemplo:

```
border-radius: 5px;
```

Definiría un radio de 5 píxeles en el redondeo de las esquinas del elemento. Por el momento para Mozilla el nombre del atributo es `-moz-border-radius` y para los navegadores basados en WebKit, como Google Chrome o Safari es `-webkit-border-radius`.

Vamos a ver un ejemplo donde queremos que todos los div tengan un borde redondeado en las esquinas de radio de 7 píxeles.:

```
DIV {  
  border: 1px solid #000000;  
  -moz-border-radius: 7px;  
  -webkit-border-radius: 7px;  
  padding: 10px;  
}
```

Esto es un div con los bordes redondeados. Pero el estilo definido por `-moz-border-radius: 7px;` sólo se verá en navegadores basados en Mozilla.

Se pueden definir los valores para el radio de las cuatro esquinas por separado. De esta manera:

```
-moz-border-radius: 7px 27px 100px 0px;
```

con lo cual se vería:

Esto es un div con los bordes redondeados. Pero el estilo definido por `-moz-border-radius: 7px 27px 100px 0px;` sólo se verá en navegadores basados en Mozilla.

## 5.4 Trabajar con columnas en CSS3

En cualquier página web resulta difícil estructurar un texto en columnas. Por lo general es necesario crear un complejo sistema de tablas para lograrlo.



En CSS3 tenemos nuevas propiedades que nos permiten trabajar con columnas. Es posible especificar el número de columnas que necesitamos y el navegador asignará automáticamente el ancho de cada columna para que quepa la cantidad que especificamos. Otra manera de trabajar con columnas con CSS3 es especificando el ancho de las columnas y se crearán las columnas que quepan de ese ancho; además se pueden especificar el margen entre cada columna, así como definir que haya una especie de borde que separe las columnas.

Como con la mayoría de las propiedades de CSS3, necesitamos agregar prefijos específicos de cada servidor. Para los navegadores basados en Webkit como Safari o Chrome se necesita el prefijo `-webkit-`; para los navegadores de Mozilla, como Firefox, se necesita `-moz-`; para Opera se utiliza `-o-`. Esperemos que se adopte CSS3 como un estándar para ya no necesitar estos prefijos.

- `Column-count` permite especificar el número de columnas que se desean. El navegador repartirá el contenido en el número de columnas especificadas. El navegador calcula el ancho de cada columna para que quepa dentro del espacio especificado, y también calculará el alto de las columnas.
- `Column-width` permite especificar el ancho de las columnas y el navegador calculará la cantidad de columnas que caben en el espacio asignado.

Aparte de las dos propiedades para definir las columnas, tenemos otras dos para dar estilo a las columnas:

- `column-gap` se utiliza para especificar un margen entre cada columna.
- `column-rule` es como un borde entre cada columna y tiene las mismas propiedades que la propiedad `border`: `width`, `color` mediante el uso de los estilos `column-rule-width`, `column-rule-color`, and `column-rule-style`.

También se puede especificar la altura de cada columna con la propiedad `column-height`. Esta se debe usar con cuidado ya que las columnas no van a exceder esta altura y si el contenido no cabe en el espacio disponible el navegador va a agregar nuevas columnas haciendo que se salga del espacio especificado.

```
.columnas{
  -moz-column-count: 3;
  -webkit-column-count: 3;
  -o-column-count: 3;
  column-count: 3;
  -moz-column-gap: 20px;
  -webkit-column-gap: 20px;
  -o-column-gap: 20px;
  column-gap: 20px;
}
```



Haz la tarea 6 de la hoja de tareas

## 5.5 Transform y transition

Los elementos HTML, cuando son creados, son como bloques sólidos e inamovibles. Pueden ser movidos usando código Javascript o aprovechando librerías populares como jQuery ([www.jquery.com](http://www.jquery.com)), pero no existía un procedimiento estándar para este propósito hasta que CSS3 presentó las propiedades `transform` y `transition`. Ajustando unos pocos parámetros nuestro sitio web puede ser tan flexible y dinámico como queramos.

La propiedad `transform` puede operar cuatro transformaciones básicas en un elemento: `scale` (escalar), `rotate` (rotar), `skew` (inclinarse) y `translate` (trasladar o mover). Veamos

cómo funcionan:

### 5.5.1 Transform

Todas las transformaciones se consiguen por medio de la propiedad `transform`, la cual tiene como valor diversos métodos.

El método actúa como una función interna. La forma de escribir un método es igual que una llamada a una función, tal como hacemos en javascript o php.

```
transform: metodo(a,b);
```

donde `metodo` es siempre una palabra clave y `a,b` son los parámetros o argumentos que le pasamos al método. Veamos los distintos métodos de la propiedad `transform`.

#### El método `translate`

El método `translate` cambia de posición el elemento (traslación), moviendolo hacia donde indican los parámetros.

```
transform: translate(30px,5px);
```

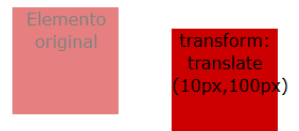
Dentro del paréntesis pondremos dos parámetros, el primero es la traslación que se produce en horizontal, hacia la derecha, (eje X), y el segundo la traslación en vertical, hacia abajo (eje Y).

Los parámetros deben ser siempre medidas de longitud, si su valor es negativo el elemento se desplazara hacia la izquierda (primer parámetro) o hacia arriba (segundo parámetro).

Veamos el siguiente ejemplo, le aplicamos a un elemento el siguiente código:

```
#capa1 { transform: translate(150px,20px); /*W3C*/  
-webkit-transform: translate(150px,20px); /*Safari y Chrome*/  
-o-transform: translate(150px,20px); /*Opera*/  
}
```

El resultado lo vemos aquí a la derecha ( Mostramos aquí cual era la posición original y cual la que toma tras la traslación):



Los métodos `translateX` y `translateY` desplazan también el elemento de su posición original, funcionan igual que el método `translate` pero sólo en una dirección, por eso sólo tienen un parámetro:

```
transform: translateX(150px);
```

Desplaza el elemento a lo largo del eje X (horizontal).

```
transform: translateY(20px);
```

Desplaza el elemento a lo largo del eje Y (vertical).

#### El método `rotate`

El método `rotate` gira el elemento respecto a su posición original, el ángulo de giro viene indicado en el parámetro:

```
transform: rotate(30deg);
```

El ángulo de rotación indicado en el parámetro puede indicarse en grados (deg) o en radianes (rad), veamos un ejemplo:

```
#capa2 {transform: rotate(30deg);}
```

El resultado de aplicar este código a un div id="capa2" lo vemos a la derecha. (En color más claro está el elemento original. El elemento gira en el sentido de las agujas del reloj. Un ángulo negativo hace que gire en sentido contrario.



Al igual que en los métodos anteriores, la propiedad `transform` debemos repetirla con el prefijo de los distintos navegadores, para que se pueda ver.

### El método `scale`

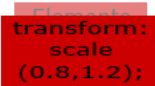
El método `scale` aumenta o reduce de tamaño el elemento original (escalar). Los ejes horizontal y vertical se controlan independientemente, por lo que podemos escalarlo de distinta manera en los dos ejes. los parámetros son números que indican la proporción en la que se debe aumentar o reducir. los decimales menores que 1 reducen el tamaño.

```
transform: scale(1.2,0.8);
```

Este método utiliza dos parámetros, que son dos números, el primer número indica la proporción en el eje X (horizontal), y el segundo en el eje Y (vertical). Veamos un ejemplo:

```
#capa3 { transform: scale(1.2,0.8);}
```

El resultado de aplicar este método a un div id="capa3" es el que vemos a la derecha.



Los métodos `scaleX` y `scaleY` descomponen el método anterior para los dos ejes del plano. Cada uno de estos métodos controla el escalamiento del elemento en un eje del plano, por eso sólo llevan un parámetro:

```
transform: scaleX(1.2);
```

el método `scaleX` aumenta o disminuye el tamaño en el eje X u horizontal.

```
transform: scaleY(0.8);
```

el método `scaleY` aumenta o disminuye el tamaño en el eje Y o vertical.

### El método `skew`

El método `skew` sesga el elemento original, es decir lo convierte en un romboide. para ello se inclinan los lados horizontal y vertical en los ángulos indicados.

```
transform: skew(10deg,15deg);
```

El primer parámetro indica la inclinación de los lados verticales, y el segundo el de los lados horizontales. Los parámetros son medidas de ángulos los cuales se pueden poner en grados (deg) o en radianes (rad). Los ángulos negativos inclinan el elemento hacia el lado contrario. Veamos un ejemplo:

```
#capa4 {transform: skew(10deg,15deg);}
```

El resultado este método a un div id="capa4" es el que vemos a la derecha.

Los métodos `skewX` y `skewY` descomponen el método anterior en sus dos componentes. `skewX` inclina sólo los lados verticales, mientras que `skewY` inclina sólo los lados horizontales, es por eso que sólo tienen un parámetro:

```
transform: skewX(10deg);
```

Los lados verticales se inclinan 10 grados, y los horizontales se quedan en la misma línea.

```
transform: skewY(15deg);
```

Los lados horizontales se inclinan 15 grados, y los verticales se quedan en la misma línea

### Combinar métodos

Podemos utilizar varios métodos para un mismo elemento, para ello basta con ponerlos seguidos, separados simplemente por uno o varios espacios.

```
#capa6 {transform: scale(1.5,0.5) rotate(45deg);}
```

El resultado es como se ve a la derecha. Después de `transform` podemos poner tantos métodos como queramos, el resultado será el de añadir varios efectos a la vez.



### 5.5.1.1 Transformaciones dinámicas

Con la propiedad `transform` podemos cambiar la apariencia de la web, pero se mantendrá tan estática como siempre. Sin embargo, podemos aprovecharnos de la combinación de transformaciones y pseudo clases para convertir nuestra página en una aplicación dinámica

```
#principal:hover{ transform: rotate(5deg); }
```

Cada vez que el puntero del ratón pasa sobre esta caja, la propiedad `transform` rota la caja en 5 grados, y cuando el puntero se aleja la caja vuelve a rotar de regreso a su posición original. Este efecto produce una animación básica usando únicamente propiedades CSS.

### 5.5.2 Transition

Una transición es un cambio progresivo de un elemento. Aplicado a los estilos CSS significa que dentro de una propiedad, para pasar de un valor a otro, se hace de manera progresiva, pasando por los valores intermedios.

Esto la mayoría de las veces provoca un efecto de animación, sobre todo en propiedades que tienen que ver con el tamaño, la posición o el color.

Las transiciones sólo se dan en las propiedades CSS cuyo valor tenga un componente numérico (número, medidas, colores, tiempo).

Sin embargo lo primero que es necesario para que haya una transición es un cambio de valor en una propiedad. Esto lo podemos hacer mediante javascript o también mediante la pseudoclase `:hover`.

Como ya hemos visto, al aplicar la pseudoclase `hover` a cualquier elemento, ésta hace que el elemento cambie cuando se le pasa el ratón por encima. El cambio se produce de manera instantánea, sin embargo si aplicamos una transición, el cambio será progresivo.

#### transition-property

La propiedad `transition-property` indica a qué propiedades CSS deben aplicarse las transiciones.

```
transition-property: none | all | <propiedad>;
```

Los posibles valores de esta propiedad son:

- `none` : esta palabra clave indica que no se aplican las transiciones a ninguna propiedad. Es el valor por defecto.
- `all` : Esta palabra clave indica que las transiciones se aplicarán a todas las propiedades que puedan admitirlas.
- `<propiedad>+` : Indicamos el nombre de la propiedad o propiedades que queremos que puedan tener transiciones. Si hay más de una éstas deben ir separadas por comas.

#### transition-duration

La propiedad `transition-duration` indica el tiempo que tardará en realizarse la transición.

```
transition-duration: <tiempo>;
```

El valor `<tiempo>`, tal como ocurre en las propiedades de animación, se mide en segundos ("s") o milisegundos ("ms").

Estas dos propiedades vistas hasta ahora son las únicas imprescindibles para que la transición pueda funcionar. Sin embargo tenemos otras propiedades:

### transition-timing-function

La propiedad `transition-timing-function` indica los cambios de la velocidad mientras se produce la transición.

```
transition-timing-function: linear | ease | ease-in | ease-out | ease-in-out |  
cubic-bezier(n,n,n,n);
```

Los valores son los mismos que para la propiedad de animación `animation-timing-function` y sus resultados son prácticamente los mismos:

- `linear` : La velocidad de la transición se mantiene constante durante toda su duración.
- `ease` : Es el valor por defecto. La transición va lenta al principio para volverse rápida en el medio, y terminar mucho más lenta.
- `ease-in` : La transición se muestra lenta al principio, para coger al final su máxima velocidad.
- `ease-out` : La transición es rápida al principio y va disminuyendo en velocidad para ser lenta al final.
- `ease-in-out` : La transición va lenta al principio, se vuelve rápida a la mitad, para acabar lenta.
- `cubic-bezier (n,n,n,n)` : Definimos la velocidad de la transición en base a una curva cúbica de Bezier. Para ello incorporamos cuatro valores "n" que serán números decimales entre el 0 y el 1.

### transition-delay

La propiedad `transition-delay` indica un retraso en el tiempo en empezar la transición.

```
transition-delay: <tiempo>
```

La ejecución de la transición se retrasará el tiempo indicado en el valor `<tiempo>`, el cual se expresa en un número seguido por la letra "s" si está en segundos, o las letras "ms" si está en milisegundos.

Vamos a ver un ejemplo de transición:

```
#capa1 { width: 80px;  
        height: 80px;  
        background-color: lime;  
        transition-property: width;  
        transition-duration: 2s; /*forma estándar*/  
        -webkit-transition-property: width;  
        -webkit-transition-duration: 2s; /*para Safari*/  
    }  
#capa1:hover { width:240px; }
```

Todas las propiedades anteriores pueden agruparse en una sola propiedad:

```
transition: <transition-property> || <transition-duration> || <transition-timing-  
function> || <transition-delay>
```

A continuación ponemos el código CSS que aunque un poco más complejo es parecido al del ejemplo anterior.

```
#capa2 { width: 80px;  
        height: 80px;  
        background-color: fuchsia;  
        opacity: 1;  
        transition: width 2s, background-color 3s linear, opacity 3s ease-out 3s;  
        -webkit-transition: width 2s, background-color 3s linear,  
                           opacity 3s ease-out 3s;  
    }  
#capa2:hover { width: 240px;  
              background-color: green;  
              opacity: 0.2;}
```



Haz la tarea 7 de la hoja de tareas.