

Continuación CSS3:.

Portales web:

www.w3schools.com

<http://www.cssportal.com/>

<http://www.tutorialspoint.com/>

<http://www.mclibre.org/>

<http://www.lawebera.es/>

Para validar:

www.css-validator.org

1) Varias columnas (column-count)

CSS3 permite disponer un bloque de texto en múltiples columnas con la única indicación de la cantidad de columnas que queremos que lo divida:

```
Elemento {  
    column-count: cantidad de columnas;  
}
```

Por ejemplo para generar un cuadro con tres columnas debemos implementar el siguiente código:

```
#recuadro1 {  
    -moz-column-count: 3;  
    -webkit-column-count: 3;  
    column-count: 3; border-  
radius: 20px;  
background-color: #ddd;  
width: 600px;  
padding: 10px;  
}
```

Recuadro 1

Lorem ipsum dolor sit amet,
consectetuer adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient
montes, nascetur ridiculus
mus. Donec quam felis,
ultricies nec,

Lorem ipsum dolor sit amet,

consectetuer adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient
montes, nascetur ridiculus
mus. Donec quam felis,
ultricies nec,

Lorem ipsum dolor sit amet,
consectetuer adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et

magnis dis parturient
montes, nascetur ridiculus
mus. Donec quam felis,
ultricies nec,

Lorem ipsum dolor sit amet,
consectetuer adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient
montes, nascetur ridiculus
mus. Donec quam felis,
ultricies nec,

Tenemos que anteceder **los prefijos -moz, -webkit para** que funcionen con el Firefox y Chrome.

2) Varias columnas dinámicas (column-width)

Con la propiedad column-width también generamos varias columnas. Tiene sentido emplearla cuando nuestro diseño es flexible, es decir que si modificamos el tamaño de la ventana del navegador su contenido se realocaliza.

Con la propiedad `column-width` especificamos el ancho de la columna en píxeles, porcentaje etc. y se generan tantas columnas como quepan en el contenedor.

La sintaxis es la siguiente:

```
Elemento {  
    column-width: ancho;  
}
```

Si queremos crear un recuadro con texto y que cada columna ocupe 150 píxeles el código es el siguiente:

```
#recuadro{  
    -moz-column-width:150px;  
    -webkit-column-width:150px;  
    column-width:150px;  
    border-radius: 20px;  
    background-color:#ddd;  
    padding:10px;  
}
```

El resultado puede variar dependiendo las dimensiones de la ventana del navegador (siempre y cuando hayamos creado una página web fluida, es decir sin un ancho fijo):

Recuadro 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

Lorem ipsum dolor sit amet,

consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et

magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

Recuadro 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec

quam felis, ultricies nec,

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

Lorem ipsum dolor sit amet,

consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

Una mejora a la disposición del texto dentro de las columnas es inicializar la propiedad `text-indent` y `margin`:

```
#recuadro2 p { text-indent: 1em; margin: 0;  
    text-align: justify;  
}
```

El resultado gráfico queda:

Recuadro 2

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

3) Separación entre columnas (column-gap)

El CSS3 nos permite especificar la separación entre las columnas. Su sintaxis es:

```
Elemento {  
    column-gap: valor;  
}
```

Podemos utilizar cualquier unidad para indicar la separación entre columnas (px, em etc.)

Si queremos una separación de 40 píxeles sería:

```
#recuadro1{  
    -moz-column-gap: 40px;  
    -webkit-column-gap: 40px;  
    column-gap: 40px;  
    -moz-column-count: 3;  
    -webkit-column-count: 3;  
    column-count: 3;  
    border-radius: 20px;  
    background-color: #ddd;  
    width: 600px;  
    padding: 10px;  
}
```

Recuadro 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, Lorem ipsum dolor sit amet, consectetur

adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et

magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

4) Línea separadora entre columnas (column- rule)

Podemos configurar una línea separadora entre las columnas cuando utilizamos múltiples columnas. La sintaxis es la siguiente:

```
Elemento {  
    column-rule: grosor estilo color;  
}
```

Con el siguiente ejemplo veamos los distintos valores de esta propiedad. Definimos una línea de un píxel, con trazo sólido de color rojo:

```
#recuadro1{  
    -moz-column-rule: 1px solid #f00;  
    -webkit-column-rule: 1px solid #f00;  
    column-rule: 1px solid #f00;  
    -moz-column-count:3;  
    -webkit-column-count:3;  
    column-count:3;  
    border-radius: 20px;  
    background-color:#ddd;  
    width:600px;  
    padding:10px;  
}
```

Recuadro 1

Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient
montes, nascetur ridiculus
mus. Donec quam felis,
ultrices nec,

Lorem ipsum dolor sit amet,

consectetur adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient
montes, nascetur ridiculus
mus. Donec quam felis,
ultrices nec,

Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et

magnis dis parturient
montes, nascetur ridiculus
mus. Donec quam felis,
ultrices nec,

Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient
montes, nascetur ridiculus
mus. Donec quam felis,
ultrices nec,

Otro ejemplo utilizando línea punteada:

```
#recuadro2{  
    -moz-column-rule: 3px dotted #f00;  
    -webkit-column-rule: 3px dotted #f00;  
    column-rule: 3px dotted #f00;  
    -moz-column-count:3;  
    -webkit-column-count:3;  
    column-count:3;  
    border-radius: 20px;  
    background-color:#ddd;  
    width:600px;  
    padding:10px;  
    margin-top:10px;
```

```
}
```

Hay tres propiedades que permiten asignar en forma independiente el grosor, estilo y color del separador:

```
Elemento {  
    column-rule-width: grosor;  
    column-rule-style: estilo; column-rule-  
    color: color;  
}
```

Por ejemplo si queremos definir un separador con línea discontinua de 2 píxeles de color azul:

```
#recuadro3{  
    -moz-column-rule-width: 2px;  
    -webkit-column-rule-width: 2px;  
    column-rule-width: 2px;  
    -moz-column-rule-style: dashed;  
    -webkit-column-rule-style: dashed; column-  
    rule-style: dashed;  
    -moz-column-rule-color: #00f;  
    -webkit-column-rule-color: #00f;  
    column-rule-color: #00f;  
  
    -moz-column-count:3;  
    -webkit-column-count:3;  
    column-count:3; border-  
    radius: 20px;  
    background-color:#ddd;  
    width:600px;  
    padding:10px;  
    margin-top:10px;  
}
```

Recuadro 3

Lorem ipsum dolor sit amet,
consectetuer adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient
montes, nascetur ridiculus
mus. Donec quam felis,
ultricies nec,

Lorem ipsum dolor sit amet,

consectetuer adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient
montes, nascetur ridiculus
mus. Donec quam felis,
ultricies nec,

Lorem ipsum dolor sit amet,
consectetuer adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et

magnis dis parturient
montes, nascetur ridiculus
mus. Donec quam felis,
ultricies nec,

Lorem ipsum dolor sit amet,
consectetuer adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient
montes, nascetur ridiculus
mus. Donec quam felis,
ultricies nec,

Si queremos una línea doble debemos indicar en el estilo el valor double:

```
#recuadro4{  
    -moz-column-rule: 4px double #aaa;  
    -moz-column-count:3;  
    -webkit-column-count:3;
```



```
column-count:3; border-
radius: 20px;
background-color:#ddd;
width:600px;
padding:10px; margin-
top:10px;
}
```

Recuadro 4

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

Lorem ipsum dolor sit amet,

consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et

magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

5)Bordes redondeados (border-radius)

La propiedad border-radius permite crear esquinas redondeadas. Especificamos en píxeles u otra medida el radio de redondeo de la o las esquinas.

Podemos indicar un único valor que se asignará a los cuatro vértices:

```
#recuadro1{
  border-radius: 20px;
  background-color:#ddd;
  width:200px;
  padding:10px;
}
```

Recuadro 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,

También podemos indicar el redondeo de cada vértice en forma independiente (el orden de los valores son la esquina superior izquierda, la esquina superior derecha, la esquina inferior derecha y por último la esquina inferior izquierda):

```
#recuadro2{
  border-radius: 20px 40px 60px 80px; background-
color:#aa0;
```

```
width:200px;
padding:10px;
margin-top:10px;
}
```

Recuadro 2

Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient montes,
nascetur ridiculus mus. Donec
quam felis, ultricies nec,

6) Bordes redondeados de alguno de los vértices (border-top-left-radius etc.)

Si tenemos que redondear solo alguno de los cuatro vértices podemos utilizar alguna de las siguientes propiedades:

border-top-left-radius

border-top-right-radius

border-bottom-right-radius

border-bottom-left-radius

Con esta definición de la propiedades border-top-left-radius y border-bottom-right-radius deben aparecer redondeados los vértices superior izquierdo e inferior derecho:

```
#recuadro1{
  border-top-left-radius: 20px;
  border-bottom-right-radius: 20px;
  background-color:#ddd;
  width:200px;
  padding:10px;
}
```

Recuadro 1

Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient montes,
nascetur ridiculus mus. Donec
quam felis, ultricies nec,

El segundo ejemplo muestra redondeado solo el vértice superior derecho:

```
#recuadro2{
  border-top-right-radius: 40px;
  background-color:#aa0;
  width:200px;
  padding:10px;
  margin-top:10px;
```


}

Recuadro 2

Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient montes,
nascetur ridiculus mus. Donec
quam felis, ultricies nec,

7) Sombras (box-shadow)

La propiedad box-shadow permite definir una sombra a un objeto de la página. Debemos definir tres valores y un color, por ejemplo:

```
#recuadro1{  
  box-shadow: 30px 10px 20px #aaa; border-  
  radius: 20px;  
  background-color:#ddd;  
  width:200px;  
  padding:10px;  
}
```

Recuadro 1

Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient montes,
nascetur ridiculus mus. Donec
quam felis, ultricies nec,

La propiedad tiene 3 valores y un color, los valores son los siguientes:

- El desplazamiento horizontal de la sombra, positivo significa que la sombra se encuentra a la derecha del objeto, un desplazamiento negativo pondrá la sombra a la izquierda.
- El desplazamiento vertical, uno negativo la sombra será en la parte superior del objeto, uno positivo la sombra estará por debajo.
- El tercer parámetro es el radio de desenfoque, si se pone a 0 la sombra será fuerte y con color liso, más grande el número, más borrosa será.
- El último valor es el color a aplicar a la sombra.

8) Sombras múltiples (box-shadow)

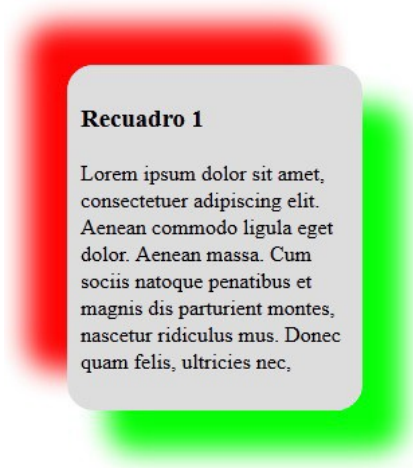
Con la propiedad box-shadow podemos aplicar múltiples sombras a un objeto. Para esto debemos aplicar la siguiente sintaxis:

```
box-shadow: [desplazamiento en x] [desplazamiento en y]  
[desenfoque] [color] ,
```

[desplazamiento en x] [desplazamiento en y]
[desenfoue] [color] etc.

Por ejemplo si queremos que aparezca una sombra de color rojo en la parte superior izquierda y una sombra verde en la parte inferior derecha podemos aplicar lo siguiente:

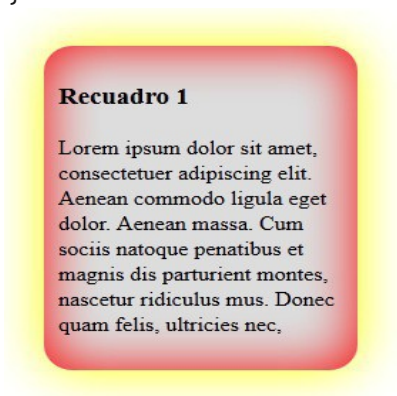
```
#recuadro1{  
  box-shadow: -30px -30px 20px #f00,  
              30px 30px 20px #0f0;  
  border-radius: 20px;  
  background-color:#ddd;  
  width:200px;  
  padding:10px;  
}
```



9) Sombras interiores (box-shadow)

Otra posibilidad de la propiedad box-shadow es la de implementar la sombra interior al objeto, para esto debemos anteceder a los valores la palabra inset. Por ejemplo si queremos un recuadro con sombra interior de color rojo y sombra exterior de color amarilla podemos aplicar los siguientes valores:

```
#recuadro1{  
  box-shadow: inset 0px 0px 40px #f00,  
              0px 0px 40px #ff0;  
  border-radius: 20px;  
  background-color:#ddd;  
  width:200px;  
  padding:10px;  
}
```



10) Sombras con transparencias (box-shadow)

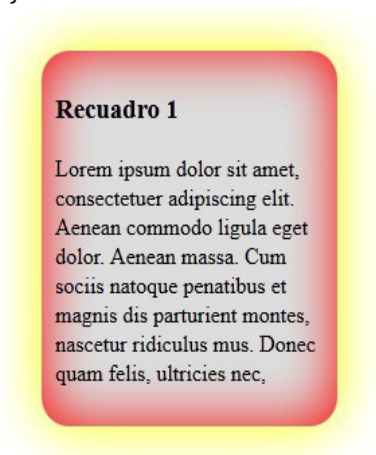
Otra posibilidad que podemos aplicar a la propiedad box-shadow es definir un valor de transparencia de la sombra. Para esto debemos utilizar la función rgba, ejemplo:

```
box-shadow: 30px 30px 30px rgba(255,0,0,0.3);
```

Mediante la función rgba indicamos el color de la sombra con los primeros tres parámetros (en este ejemplo 255 de rojo, 0 de verde y 0 de azul), luego el cuarto parámetro es el índice de transparencia (donde 0 indica quee stotalmente transparente la sombra y 1 es totalmente opaca, podemos utilizar cualquier valor entre 1 y 0)

Para confeccionar un recuadro con sombra de color rojo con un índice de transparencia del 50% debemos especificarlo de la siguiente manera:

```
#recuadro1{  
  box-shadow: 30px 30px 30px rgba(255,0,0,0.5);  
  border-radius: 20px;  
  background-color:#ddd;  
  width:200px;  
  padding:10px;  
}
```



11) Sombras de texto (text-shadow)

La propiedad text-shadow nos permite definir una sombra a un texto, la sintaxis más común es: Elemento {

```
  text-shadow: desplazamientoX desplazamientoY radio-de-  
desenfoque color;  
}
```

La propiedad tiene 4 valores que son los siguientes:

- El desplazamiento horizontal de la sombra, un desplazamiento negativo pondrá la sombra a la izquierda.
- El desplazamiento vertical, un valor negativo dispone la sombra en la parte superior del texto, uno positivo la sombra estará por debajo del texto.
- El tercer parámetro es el radio de desenfoque, si se pone a 0 la sombra será fuerte y con color liso, más grande el número, más borrosa será.
- El último parámetro es el color de la sombra.

Por ejemplo si queremos que un texto tenga una sombra en la parte inferior a derecha con un pequeño desenfoque de color gris luego debemos implementar el siguiente código:

```
#titulo1 {  
  text-shadow: 5px 5px 5px #aaa;  
}
```

Titulo sombreado

SI queremos que la sombra se disponga en la parte superior izquierda de cada letra luego debemos definir los siguientes valores:

```
#titulo2 {  
  text-shadow: -5px -5px 5px #aaa;  
}
```

Titulo sombreado

Otra sintaxis de text-shadow es aplicar varias sombras al texto, por ejemplo:

```
#titulo3 {  
  text-shadow: 3px 3px 5px #f00,  
               6px 6px 5px  
               #0f0, 9px 9px  
               5px #00f;  
}
```

Titulo sombreado

}

12) Transformaciones 2D: rotación (transform: rotate)

Las transformaciones todavía no están definidas como un estándar en todos los navegadores, por lo que es necesario agregar el prefijo del navegador que la implementa:

```
Elemento {  
  -ms-transform: función de transformación(valor(es)); /*  
Internet Explorer */  
  -webkit-transform: función de transformación(valor(es));  
  /*  
WebKit */  
  -moz-transform: función de transformación(valor(es)); /*  
Firefox */  
  -o-transform: función de transformación(valor(es)); /*  
Opera */  
}
```

Tengamos en cuenta que la propiedad de transformación 2D definitiva será:

```
Elemento {  
  transform: función de transformación(valor(es));  
}
```

La primer función de transformación que veremos será la de rotar un elemento HTML.

La función de rotación se llama rotate y tiene un parámetro que indica la cantidad de grados a rotar. La rotación es en el sentido de las agujas del reloj. Podemos indicar un valor negativo para rotar en sentido antihorario.

Para rotar un recuadro 45 grados en sentidos de las agujas de un reloj y que funcione en la mayoría de los navegadores deberemos implementar el siguiente código:

```
#recuadro1 {  
  -ms-transform: rotate(45deg);
```



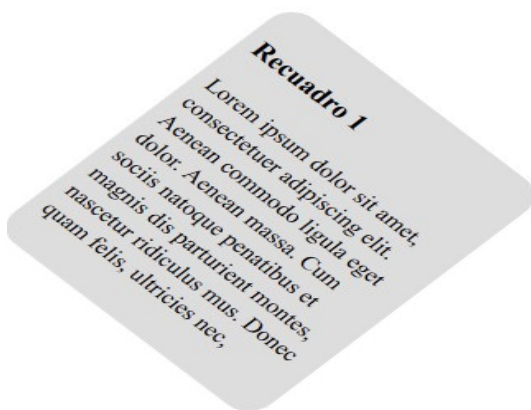
```

-webkit-transform: rotate(45deg);
-moz-transform: rotate(45deg);
-o-transform: rotate(45deg);
transform:
rotate(45deg);
border-radius: 20px;
background-
color:#ddd;
width:200px;
padding:10px;
}

```

Tengamos en cuenta que se ejecuta la propiedad -ms-transform, -webkit-transform etc. según el navegador que está procesando la página, inclusive hemos agregado la propiedad transform: rotate(45deg) que será la que en un futuro todos los navegadores interpretarán.

El resultado visual es el siguiente:



Otra cosa importante para anotar es que el punto de rotación coincide con el centro del recuadro (es como clavar una alfiler en el centro del recuadro y luego rotar el recuadro en el sentido de las agujas del reloj).

También podemos rotar en sentido antihorario indicando el valor del grado con un valor negativo:

```

#recuadro2{
-ms-transform: rotate(-45deg);
-webkit-transform: rotate(-45deg);
-moz-transform: rotate(-45deg);
-o-transform: rotate(-
45deg); transform:
rotate(-45deg); border-
radius: 20px; background-
color:#ddd; width:200px;
padding:10px;
}

```

También podemos observar que cuando la rotación se ejecuta no ocupa más espacio el elemento HTML sino que se solapa con otros elementos de la página:



13) Transformaciones 2D: punto de origen (transform-origin)

Como vimos en el punto anterior cuando rotamos un elemento HTML siempre hay un punto fijo (por defecto es el punto central del recuadro)

Podemos variar dicho punto y ubicar por ejemplo en cualquier uno de los cuatro vértices del recuadro con la siguiente sintaxis:

```
Elemento {
    transform-origin: left top;
}

Elemento {
    transform-origin: right top;
}

Elemento {
    transform-origin: left bottom;
}

Elemento {
    transform-origin: right bottom;
}
```

Por ejemplo si queremos dejar fijo el vértice superior izquierdo y seguidamente rotar 10 grados en sentido horario luego debemos codificar:

```
#recuadro1{
    -ms-transform: rotate(10deg);
    -webkit-transform: rotate(10deg);
    -moz-transform: rotate(10deg);
    -o-transform: rotate(10deg);
    transform: rotate(10deg);

    -ms-transform-origin: left top;
    -webkit-transform-origin: left top;
    -moz-transform-origin: left top;
    -o-transform-origin: left top;
```

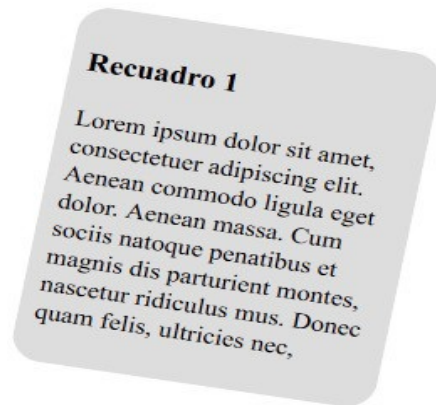
```

transform-origin: left top;

border-radius:
20px; background-
color:#ddd;
width:200px;
padding:10px;
}

```

Como podemos ver es como poner un alfiler en el vértice superior izquierdo y seguidamente rotar en el sentido de las agujas de un reloj 10 grados:



Con lo visto podemos disponer el punto de origen en cinco lugares (el centro y los cuatro vértices), pero podemos trasladar a cualquier punto dentro del recuadro el punto del origen con las siguientes sintaxis:

```
transform-origin: 0% 50%;
```

El primer valor representa las "x" y el segundo valor representa las "y".

Si queremos disponer el punto de origen en el vértice superior derecho podemos hacerlo como ya conocemos:

```

Elemen
to {
    transform-origin: right top;
}

```

O mediante porcentajes:

```

Elemento {
    transform-origin: 100% 0%;
}

```

Luego podemos mediante porcentajes en x e y desplazar el punto de origen a cualquier parte dentro del elemento HTML (div en este caso)

Por ejemplo si queremos disponer el punto de origen en la mitad del lado izquierdo:

```

#recuadro2{
    -ms-transform: rotate(10deg);
    -webkit-transform: rotate(10deg);
    -moz-transform: rotate(10deg);
    -o-transform: rotate(10deg);
    transform: rotate(10deg);

    -ms-transform-origin: 0% 50%;
    -webkit-transform-origin: 0% 50%;
    -moz-transform-origin: 0% 50%;
}

```

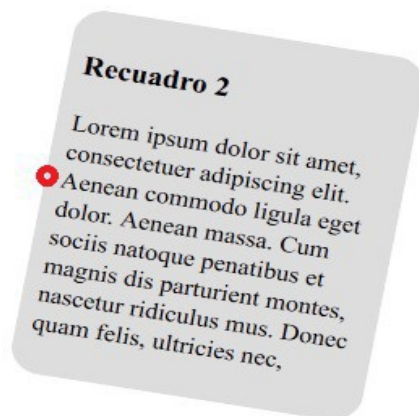
```

-o-transform-origin: 0% 50%;
transform-origin: 0% 50%;

border-radius: 20px;
background-
color:#ddd;
width:200px;
padding:10px;
margin-top:50px;
}

```

El punto rojo indica el "punto de origen":



Otra posibilidad es indicar el punto con alguna medida permitida en CSS (px, em etc.), por ejemplo podemos indicar que el punto de origen este 20 píxeles en x y 40 en y:

```

#recuadro3{
  -ms-transform: rotate(10deg);
  -webkit-transform: rotate(10deg);
  -moz-transform: rotate(10deg);
  -o-transform:
  rotate(10deg); transform:
  rotate(10deg);

  -ms-transform-origin: 20px 40px;
  -webkit-transform-origin: 20px 40px;
  -moz-transform-origin: 20px 40px;
  -o-transform-origin: 20px 40px; transform-
  origin: 20px 40px;

  border-radius: 20px;
  background-color:#ddd;
  width:200px;
  padding:10px;
  margin-top:50px;
}

```

14) Transformaciones 2D: escalado (transform: scale)

Otra función de transformación 2D es el escalado, esta función permite agrandar o reducir el tamaño del elemento. La primer sintaxis que podemos utilizar es la siguiente:

```

Elemento {
  transform: scale(valorx ,valory);
}

```


El primer parámetro indica la escala para x (con el valor 1 el elemento queda como está, con un valor mayor crece y con un valor menor decrece), el segundo parámetro es para la escala en y.

Por ejemplo si queremos que sea del doble de ancho y la mitad de altura luego será:

```
Elemento {  
    transform: scale(2 ,0.5);  
}
```

Un recuadro escalado con 20% menos de ancho y 20% más de alto:

```
#recuadro1{  
    -ms-transform: scale(0.8 , 1.2);  
    -webkit-transform: scale(0.8 , 1.2);  
    -moz-transform: scale(0.8 , 1.2);  
    -o-transform: scale(0.8 , 1.2);  
    transform: scale(0.8 , 1.2);  
  
    -ms-transform-origin: left top;  
    -webkit-transform-origin: left top;  
    -moz-transform-origin: left top;  
    -o-transform-origin: left  
    top; transform-origin:  
    left top;  
  
    border-radius: 20px;  
    background-  
    color:#ddd;  
    width:200px;  
    padding:10px;  
}
```

Recuadro 1

Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient montes,
nascetur ridiculus mus. Donec
quam felis, ultricies nec,

Si tenemos que escalar solo en x o en y podemos utilizar alguna de las dos funciones scaleX o scaleY.

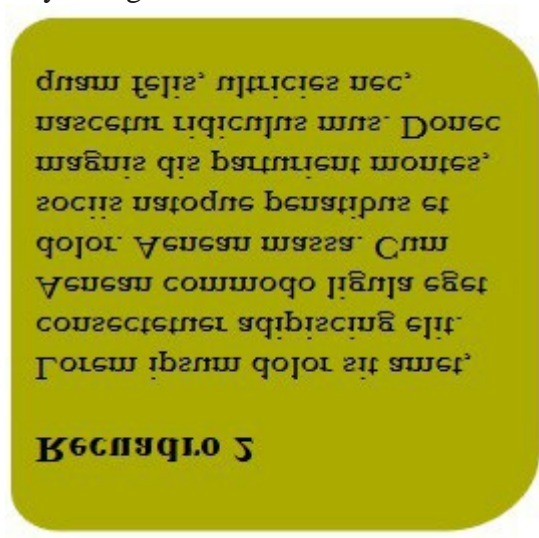
```
Elemento {  
    transform: scaleX(2);  
}  
Elemento {  
    transform: scaleY(0.5);
```

```
}
```

Otra posibilidad es utilizar un valor negativo, lo que nos permite tener una reflexión del elemento. Por ejemplo un recuadro con reflexión en y:

```
#recuadro2{  
  -ms-transform: scale(1.20 , -1);  
  -webkit-transform: scale(1.20 , -1);  
  -moz-transform: scale(1.20 , -1);  
  -o-transform: scale(1.20 , -1);  
  transform: scale(1.20 , -1);  
  
  border-radius: 20px 40px 40px 20px; background-  
  color:#aa0;  
  width:200px;  
  padding:10px;  
  margin-top:70px;  
}
```

Cuya imagen es:



15) Transformaciones 2D: translación (transform: translate)

La función translate permite desplazar un elemento HTML indicando una medida (en porcentaje, píxeles, em, etc.) la sintaxis es:

```
Elemento {  
  transform: translate(valorx ,valory);  
}
```

Por ejemplo si queremos trasladar 25 píxeles en "x" y 10 píxeles en "y" el código es:

```
#recuadro1{  
  -ms-transform: translate(25px,10px);  
  -webkit-transform: translate(25px,10px);  
  -moz-transform: translate(25px,10px);  
  -o-transform: translate(25px,10px);  
  transform: translate(25px,10px);  
  
  border-radius: 20px;  
  background-color:#ddd;  
  width:200px;  
  padding:10px;
```

}

Si vemos el primer recuadro tiene aplicada la traslación:

Recuadro 1

Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient montes,
nascetur ridiculus mus. Donec
quam felis, ultricies nec,

Recuadro 2

Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Aenean commodo ligula eget
dolor. Aenean massa. Cum
sociis natoque penatibus et
magnis dis parturient montes,
nascetur ridiculus mus. Donec
quam felis, ultricies nec,

También disponemos de las funciones `translateX` y `translateY` para los casos donde solo debemos trasladar en x o y:

```
Elemento {  
    transform: translateX(valor);  
}
```

```
Elemento {  
    transform: translateY(valor);  
}
```

16) Transformaciones 2D: torcer (`transform: skew`)

La función `skew` permite torcer el elemento HTML en x e y, la sintaxis es la siguiente:

```
Elemento {  
    transform: skew(gradosx ,gradosy);  
}
```

Por ejemplo si queremos torcer en x 15 grados un recuadro luego debemos codificar:

```
#recuadro1{  
    -ms-transform: skew(15deg,0deg);  
    -webkit-transform: skew(15deg,0deg);
```

```

-moz-transform: skew(15deg,0deg);
-o-transform: skew(15deg,0deg);
transform: skew(15deg,0deg); border-
radius: 20px;
background-color:#ddd;
width:200px;
padding:10px;
}

```

Recuadro 1

Lorem ipsum dolor sit amet,
 consectetur adipiscing elit.
 Aenean commodo ligula eget
 dolor. Aenean massa. Cum
 sociis natoque penatibus et
 magnis dis parturient montes,
 nascetur ridiculus mus. Donec
 quam felis, ultricies nec,

Si queremos torcer en y 15 grados un recuadro luego debemos codificar:

```

#recuadro2{
-ms-transform: skew(0deg,15deg);
-webkit-transform: skew(0deg,15deg);
-moz-transform: skew(0deg,15deg);
-o-transform: skew(0deg,15deg);
transform: skew(0deg,15deg); border-
radius: 20px;
background-color:#ddd;
width:200px;
padding:10px;
margin-top:50px;
}

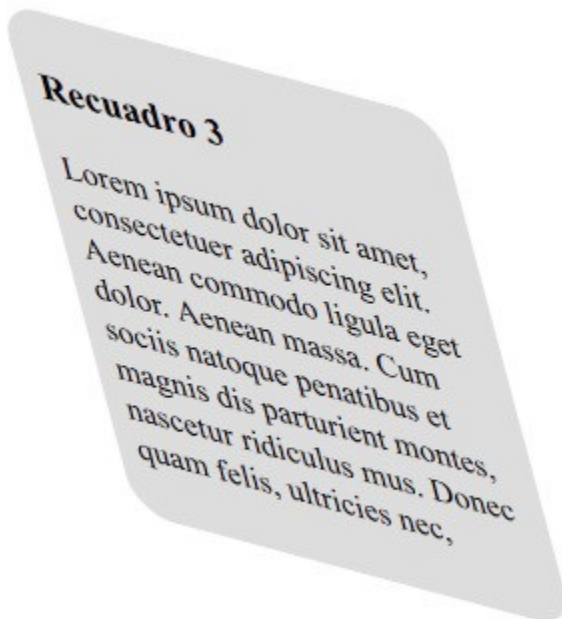
```

Recuadro 2

Lorem ipsum dolor sit amet,
 consectetur adipiscing elit.
 Aenean commodo ligula eget
 dolor. Aenean massa. Cum
 sociis natoque penatibus et
 magnis dis parturient montes,
 nascetur ridiculus mus. Donec
 quam felis, ultricies nec,

Por último si aplicamos tanto en x e y tenemos:

```
#recuadro3{
  -ms-transform: skew(15deg,15deg);
  -webkit-transform: skew(15deg,15deg);
  -moz-transform: skew(15deg,15deg);
  -o-transform: skew(15deg,15deg);
  transform: skew(15deg,15deg);
  border-radius: 20px;
  background-color:#ddd;
  width:200px;
  padding:10px; margin-
  top:70px;
}
```



También disponemos de las funciones skewX y skewY para los casos donde solo debemos torcer en x o y:

```
Elemento {
  transform: skewX(grados);
}
```

```
Elemento {
  transform: skewY(grados);
}
```

Los grados también pueden ser un valor negativo.

17) Transformaciones 2D: múltiples transformaciones en forma simultanea

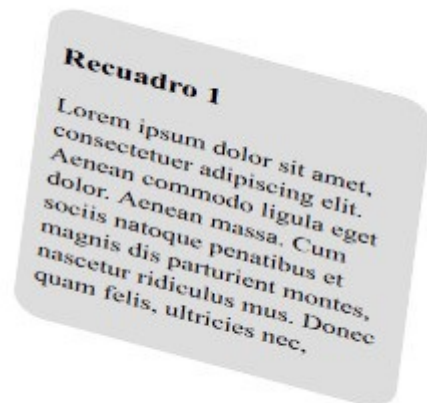
La sintaxis de css3 nos permite definir en la propiedad transform múltiples transformaciones en forma simultanea (debemos dejar al menos un espacio en blanco entre cada llamada a una función de transformación):

```
Elemento {
  transform: rotate(grados) scale(valorx ,valory)
  translate(valorx ,valory) skew(gradosx ,gradosy);
}
```

```
}
```

Por ejemplo apliquemos múltiples transformaciones a un recuadro:

```
#recuadro1{
  -moz-transform: rotate(15deg) scale(0.9,0.6)
  translate(10px,10px) skew(5deg,0deg);
  -ms-transform: rotate(15deg) scale(0.9,0.6)
  translate(10px,10px) skew(5deg,0deg);
  -webkit-transform: rotate(15deg) scale(0.9,0.6)
  translate(10px,10px) skew(5deg,0deg);
  -o-transform: rotate(15deg) scale(0.9,0.6)
  translate(10px,10px) skew(5deg,0deg);
  transform: rotate(15deg) scale(0.9,0.6) translate(10px,10px)
  skew(5deg,0deg);
  border-radius: 20px;
  background-color:#ddd;
  width:200px;
  padding:10px;
}
```



18)Opacidad (opacity)

La opacidad es una característica de los objetos de no dejar pasar la luz (mientras un objeto es más opaco significa que no deja pasar la luz) Un elemento HTML dispone de la propiedad opacity para definir cual es su opacidad. La sintaxis es la siguiente:

```
Elemento {
  opacity: valor;
}
```

El valor es un número comprendido entre 0 y 1. El 0 significa que es totalmente transparente (luego no se verá nada en pantalla, pero el espacio ocupado por el elemento HTML queda reservado), el 1 significa que es totalmente opaco (no deja pasar la luz)

Veamos tres recuadro con una imagen de fondo y un texto en su interior con diferentes niveles de opacidad (tengamos en cuenta que cuando le asignamos una opacidad a un elemento HTML todos los elementos contenidos en dicho elementos heredan dicha opacidad):

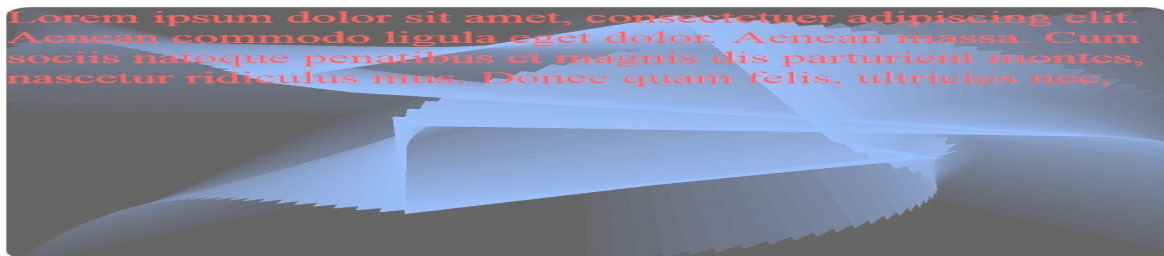
```
#recuadro1 {
  background-image: url("foto1.jpg");
  opacity:0.3;
  color:#f00;
  width:700px;
```

```
height:450px; border-  
radius:15px; font-  
size:30px;  
}
```



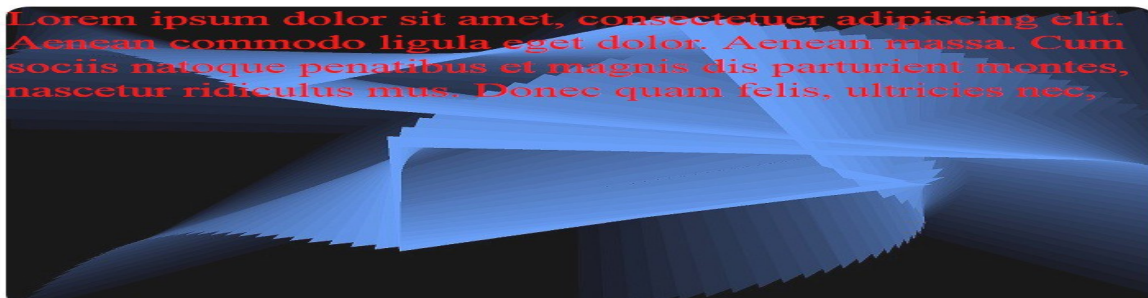
Ahora con una opacidad mayor:

```
#recuadro2 {  
  background-image: url("foto1.jpg");  
  opacity:0.6;  
  color:#f00;  
  width:700px;  
  height:450px; border-  
radius:15px; font-  
size:30px;  
}
```



Finalmente con una opacidad de 0.9 (casi no deja pasar nada de luz):

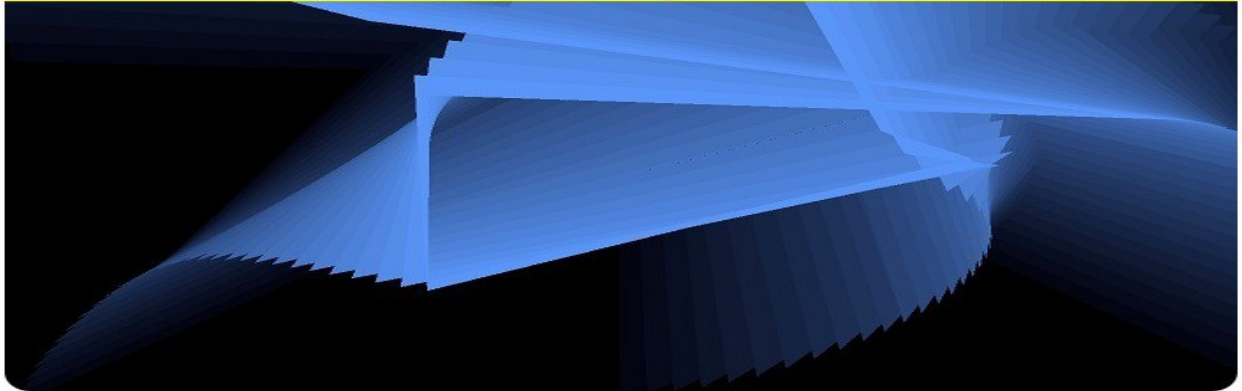
```
#recuadro3 {  
  background-image: url("foto1.jpg");  
  opacity:0.9;  
  color:#f00;  
  width:700px;  
  height:450px; border-  
radius:15px; font-  
size:30px;  
}
```



Veamos que pasa si no ponemos opacidad en el recuadro y el texto tiene un color de fondo:

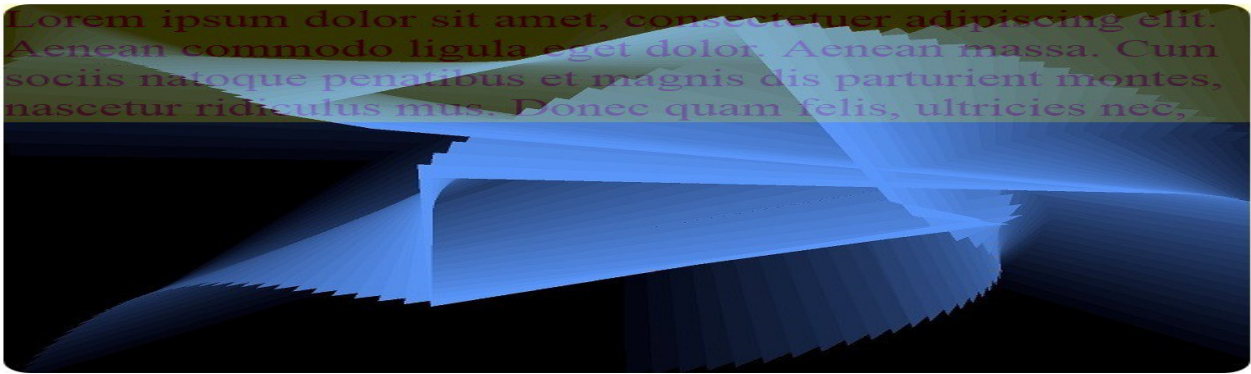
```
#recuadro4 {  
  background-image: url("foto1.jpg");  
  color:#f00;  
  width:700px;  
  height:450px; border-  
  radius:15px; font-  
  size:30px;  
}  
  
.texto4 {  
  background-color:yellow; /*Al no llevar opacidad no se ve la  
imagen de fondo*/  
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,



Definiendo una opacidad en el párrafo:

```
#recuadro5 {  
  background-image: url("foto1.jpg");  
  color:#f00;  
  width:700px;  
  height:450px; border-  
  radius:15px; font-  
  size:30px;  
}  
  
.texto5 {  
  background-color:yellow;  
  opacity:0.2;  
}
```

19) Opacidad (color)

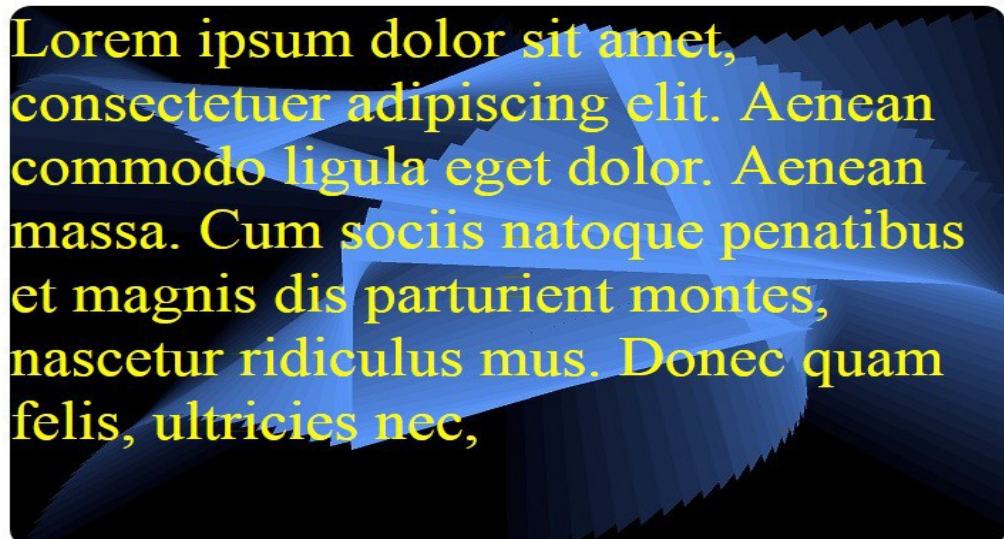
En CSS3 aparece una variante de la asignación del color mediante la función rgba.

```
Elemento {  
    color: rgba(rojo,verde,azul,opacidad);  
}
```

Para definir un color debemos indicar cuatro valores, los tres primeros son valores enteros entre 0 y 255, que indican la cantidad de rojo, verde y azul. El cuarto valor es un número entre 0 y 1 que indica la opacidad que se aplica al color. Si indicamos un 1 se grafica totalmente opaco (es el valor por defecto) un valor menor hará más transparente el gráfico.

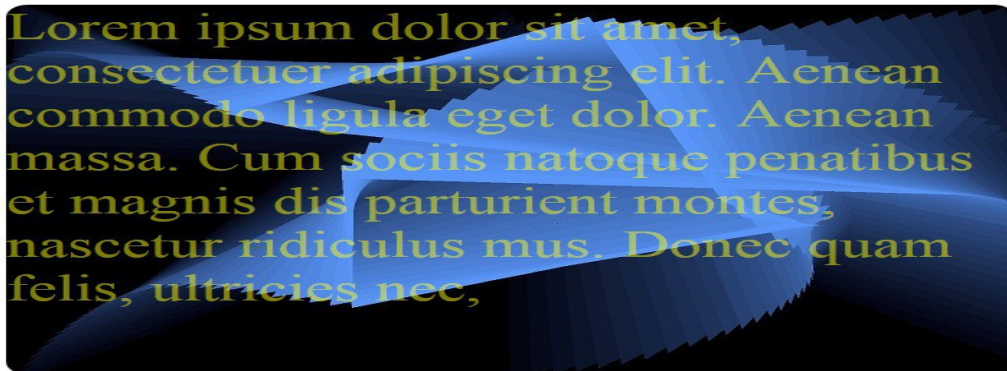
Veamos un ejemplo primero dispondremos un texto dentro de un recuadro que tiene una imagen y no utilizaremos la función rgba para definir la opacidad, sino utilizaremos la función rgb para definir solo el color:

```
#recuadro1 {  
    background-image: url("foto1.jpg");  
    color:#f00;  
    width:700px;  
    height:450px;  
    border-radius:15px;  
    font-size:45px;  
}  
  
.text01 {  
    color:rgb(255,255,0);  
}
```



Y ahora si utilizaremos la función nueva rgba:

```
#recuadro2 {  
  background-image: url("foto1.jpg");  
  color:#f00;  
  width:700px;  
  height:450px;  
  border-radius:15px;  
  font-size:45px;  
  margin-top:10px;  
}  
  
.texto2 {  
  color:rgba(255,255,0,0.5);  
}
```



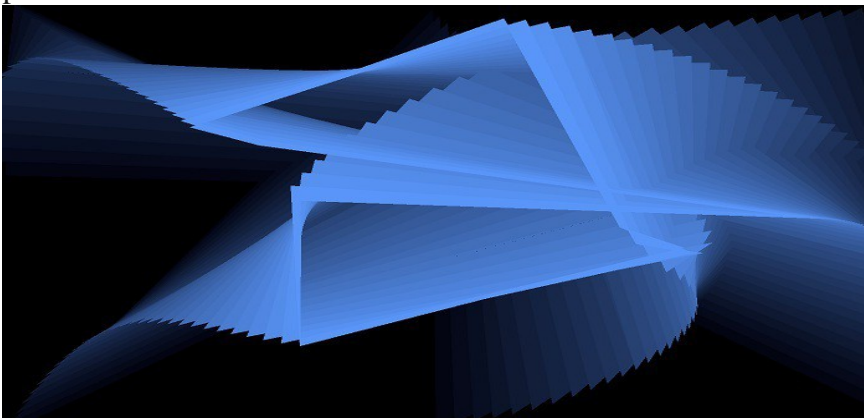
20) Varias imágenes de fondo (background-image)

La propiedad background-image permite insertar un conjunto de imágenes dentro de un elemento. Para ello debemos especificar sus nombres:

```
Elemento {  
  background-image: url("imagen1"), url("imagen2", ...);  
}
```

La primera imagen que se dibuja es la que indicamos al final de la lista.

Por ejemplo vamos a mostrar una imagen de fondo con formato jpg y sobre esta una de tipo png, la primera:



y la imagen png es:



La página que muestra superpuestas las dos imágenes en un recuadro es:

```
<!DOCTYPE html>
<html>
<head>
<title>Prueba</title>

<style type="text/css">

#recuadro1{
    background-image: url("logo1.png"), url("foto1.jpg");
    background-repeat: no-repeat;
    width:700px;
    height:450px;
}

body {
    background:white;
    margin:50px;
}

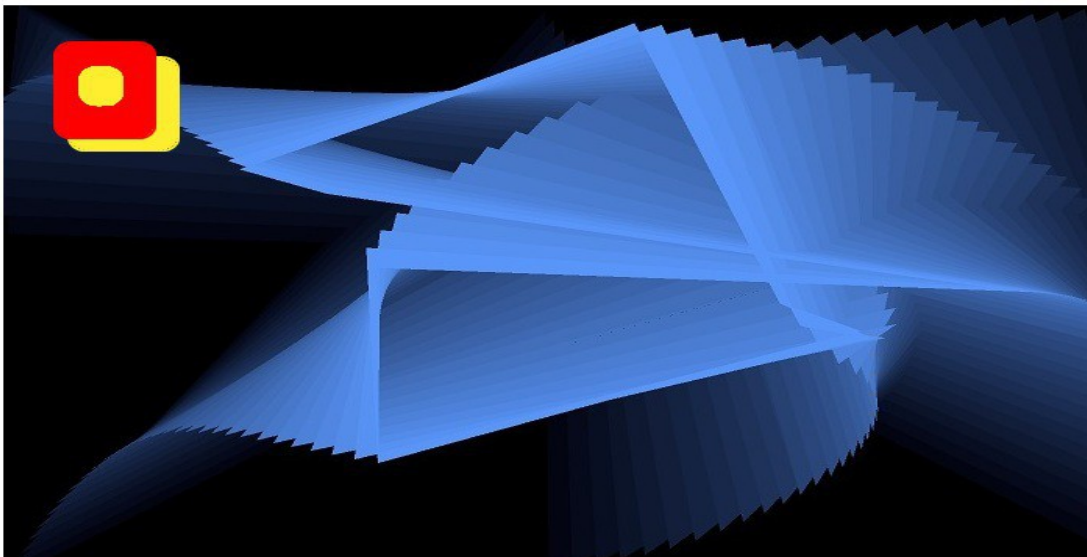
</style>

</head>
<body>

<div id="recuadro1">
</div>

</body>
</html>
```

El resultado es:



Para esto definimos:

```
#recuadro1{
  background-image: url("logo1.png"), url("foto1.jpg");
  background-repeat: no-repeat;
  width:700px;
  height:450px;
}
```

El recuadro coincide con el tamaño de la imagen "foto1.jpg" (700*450) La imagen "logo1.png" es de 150*150 píxeles.

Como vemos primero se dibuja la imagen "foto1.jpg" (que es la última) y luego sobre esta la imagen "logo1.png". Otra cosa importante es inicializar la propiedad background-repeat con el valor no-repeat. En caso de no inicializar dicha propiedad tendremos a la imagen logo1.png repetida dentro del recuadro.

21) Imágenes de fondo en diferentes posiciones (background-position)

Podemos mediante la propiedad background-position indicar la posición de la imagen con respecto al contenedor.

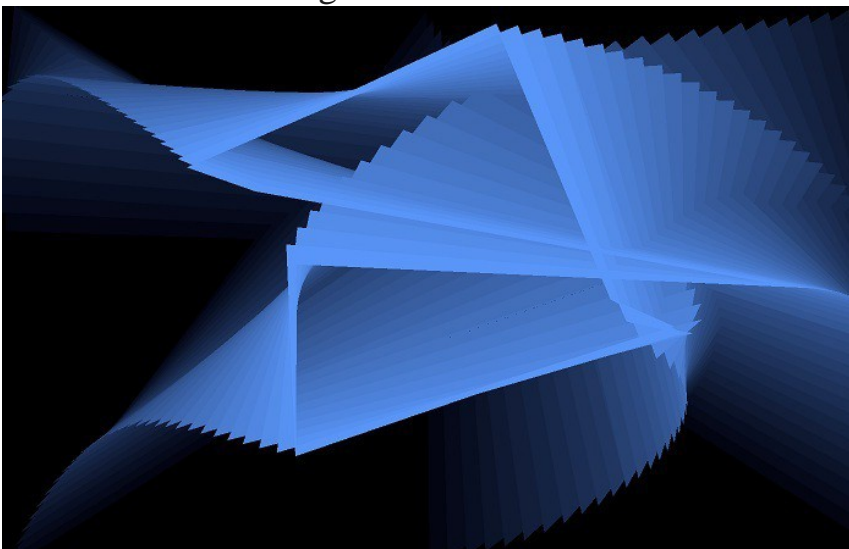
La sintaxis es la siguiente:

```
Elemento {
  background-image: url("imagen1"), url("imagen2", ...);
  background-position: posx posy , posx posy;
}
```

El par de valores que indicamos coincide con la posición de las imágenes que especificamos en la propiedad background-image.

Implementaremos una página que muestre una imagen de 700*400 y dentro de esta otra imagen que se ubique en los cuatro vértices de la imagen más grande.

Utilizaremos las imágenes:



y la imagen:



La página es:

```
<!DOCTYPE html>
<html>
<head>
<title>Prueba</title>

<style type="text/css">

#recuadro1{
    background-image:    url("logo1.png"),
                        url("logo1.png"),
                        url("logo1.png"),
                        url("logo1.png"),
                        url("foto1.jpg");

    background-position: 0% 0%,
                        100% 0%,
                        0% 100%,
                        100% 100%;

    background-repeat: no-repeat;
    width:700px;
    height:450px;
}

body {
    background:white;
    margin:50px;
}

</style>

</head>
<body>

<div id="recuadro1">
</div>

</body>
</html>
```

Hay que recordar que la primer imagen que se muestra es la que especificamos última en la propiedad background-image. Luego en la propiedad background-position indicamos en el mismo orden la ubicación de cada image. La primera la ubicamos en el vértice superior izquierdo ya que especificamos los valores 0% 0%. La segunda imagen la especificamos en el vértice superior derecho con los valores 100% 0%. Así indicamos las otras dos imágenes. **La quinta imagen no es necesario indicar el background-position ya que el ancho y alto coincide con los valores asignados a las propiedades width y height.**

```
#recuadro1{
    background-image:    url("logo1.png"),
                        url("logo1.png"),
```



```

        url("logo1.png"),
        url("logo1.png"),
        url("foto1.jpg");
background-position: 0% 0%,
                    100% 0%,
                    0% 100%,
                    100% 100%;
background-repeat: no-repeat;
width:700px;
height:450px;
}

```

22) Escalar una imagen de fondo (background-size)

Esta propiedad nos permite escalar una imagen dispuesta en el fondo. La sintaxis es:

```

Elemento {
    background-size: ancho alto;
}

```

Estas longitudes se las puede especificar en píxeles, porcentajes etc.

Por ejemplo si queremos mostrar tres imágenes dentro de un div con distintos tamaño:

```

<!DOCTYPE html>
<html>
<head>
<title>Prueba</title>

<style type="text/css">

#recuadro1{
    background-image:    url("logo1.png"),
                        url("logo1.png"),
                        url("foto1.jpg");

    background-position: 0% 0%,
                        50% 50%;

    background-size:    30px 30px,
                        250px 250px,
                        700px 450px;

    background-repeat: no-repeat;
    width:700px;
    height:450px;
}

body {
    background:white;
    margin:50px;
}

</style>

</head>
<body>

```

```
<div id="recuadro1">
</div>

</body>
</html>
```

Podemos ver que dibujamos tres imágenes: "foto1.jpg" aparece en el fondo y dos veces mostramos "logo1.png" y le damos como tamaño al primero 30*30 píxeles y al segundo 250*250.

```
#recuadro1{
    background-image:    url("logo1.png"),
                        url("logo1.png"),
                        url("foto1.jpg");

    background-position: 0% 0%,
                        50% 50%;

    background-size:     30px 30px,
                        250px 250px,
                        700px 450px;

    background-repeat: no-repeat;
    width:700px;
    height:450px;
}
```

Si queremos que una imagen tome el tamaño por defecto que tiene la imagen sin reescalar podemos utilizar la palabra clave "auto", por ejemplo para mostrar la primera imagen con el tamaño original podemos escribir:

```
#recuadro1{
    background-image:    url("logo1.png"),
                        url("logo1.png"),
                        url("foto1.jpg");

    background-position: 0% 0%,
                        50% 50%;

    background-size:     30px 30px,
                        250px 250px,
                        auto auto;

    background-repeat: no-repeat;
    width:700px;
    height:450px;
}
```

Si utilizamos como unidad porcentajes, la dimensión se basa en el elemento contenedor.

```
#recuadro1{
    background-image:    url("logo1.png"),
                        url("logo1.png"),
                        url("foto1.jpg");

    background-position: 0% 0%,
                        50% 50%;

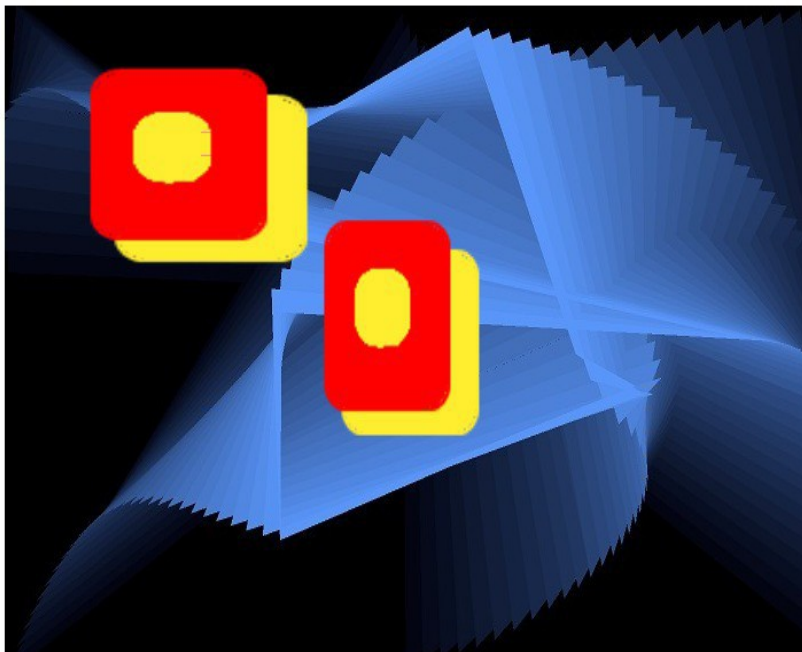
    background-size:     50% 50%,
                        250px 250px,
```

```

        auto auto;

background-repeat: no-repeat;
width:700px;
height:450px;
}

```



23) Modificar el origen de la imagen de fondo (background-origin)

La propiedad background-origin establece el punto donde se comienza a dibujar el fondo. Pudiendo ser en el borde, guardando el espacio de padding o el contenido (border-box, padding-box o content-box)

```

Elemento {
    background-origin: border-box/padding-box/content-box;
}

```

Un ejemplo de tres recuadros con imágenes de fondo inicializando la propiedad background-origin con los tres valores posibles:

```

<!DOCTYPE html>
<html>
<head>
<title>Prueba</title>

<style type="text/css">

#recuadro1{
    background-origin: border-box;
    border: 20px solid #eee;
    padding: 20px;
    background-image: url("casa.png");
    background-color:#ff0;

```

```

background-repeat: no-repeat;
width:300px;
height:200px;
}

#recuadro2{
background-origin: padding-box;
border: 20px solid #eee;
padding: 20px;
background-image: url("casa.png"); background-
color:#ff0;
background-repeat: no-repeat;
width:300px;
height:200px; margin-
top:50px;
}

```

```

#recuadro3{
background-origin: content-box;
border: 20px solid #eee;
padding: 20px;
background-image: url("casa.png"); background-
color:#ff0;
background-repeat: no-repeat;
width:300px;
height:200px; margin-
top:50px;
}

```

```

body {
background:white;
margin:50px;
}

```

```
</style>
```

```
</head>
```

```
<body>
```

```
<div id="recuadro1">
```

```
<p>EL ENTUSIASMO MAGNETIZA DE TAL FORMA NUESTROS PENSAMIENTOS, QUE
NO HAY MANERA DE DETENER LA AVALANCHA ENLOQUECEDORA DE POSITIVISMO
QUE RODEA AL PENSADOR ENTUSIASTA.
```

```
ES TAL LA FUERZA DE UN PENSAMIENTO ENTUSIASTA, QUE LOGRA CONTAGIAR
HASTA EL MAS FRIÓ Y ESCÉPTICO INDIVIDUO, ARRASTRÁNDOLO DE UN POLO A
OTRO EN SU FORMA DE OBSERVAR LA VIDA. </p>
```

```
</div>
```

```
<div id="recuadro2">
```

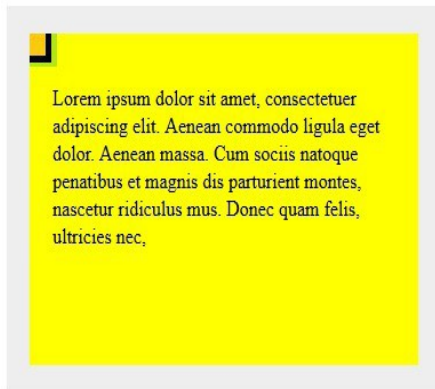
```
<p>EL ENTUSIASMO MAGNETIZA DE TAL FORMA NUESTROS PENSAMIENTOS, QUE NO HAY
MANERA DE DETENER LA AVALANCHA ENLOQUECEDORA DE POSITIVISMO QUE RODEA AL
PENSADOR ENTUSIASTA.
```

```
ES TAL LA FUERZA DE UN PENSAMIENTO ENTUSIASTA, QUE LOGRA CONTAGIAR HASTA EL MAS
FRIÓ Y ESCÉPTICO INDIVIDUO, ARRASTRÁNDOLO DE UN POLO A OTRO EN SU FORMA DE
OBSERVAR LA VIDA. </p></div>
```

```
<div id="recuadro3">
<p>EL ENTUSIASMO MAGNETIZA DE TAL FORMA NUESTROS PENSAMIENTOS,
QUE NO HAY MANERA DE DETENER LA AVALANCHA ENLOQUECEDORA DE
POSITIVISMO QUE RODEA AL PENSADOR ENTUSIASTA.
ES TAL LA FUERZA DE UN PENSAMIENTO ENTUSIASTA, QUE LOGRA CONTAGIAR HASTA EL MAS
FRIÓ Y ESCÉPTICO INDIVIDUO, ARRASTRÁNDOLO DE UN POLO A OTRO EN SU FORMA DE
OBSERVAR LA VIDA.</p></div>

</body>
</html>
```

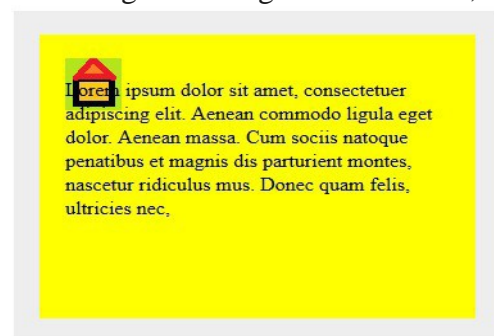
Con background-origin: border-box; el resultado es:



Con background-origin: padding-box; el resultado es:



Con background-origin: content-box; el resultado es:



23) Transiciones de una propiedad (transition)

Las transiciones en css3 permiten modificar el valor de una propiedad de un elemento HTML en forma gradual durante un tiempo determinado de un estado inicial a un estado final.

La sintaxis más simple para definir una transición de una propiedad es:

Elemento {


```
    transition: [nombre de propiedad] [duración de la
transición] ;
}
```

Por ejemplo crearemos dos recuadros y cuando pasemos la flecha del mouse sobre el mismo cambiaremos su tamaño. El primer recuadro lo haremos sin transición:

```
#recuadro1{
  border-radius: 20px;
  background-color:#ddd;
  width:200px;
  padding:10px;
}

#recuadro1:hover{
  width:300px;
}
```

Cuando probamos la página veremos que el recuadro1 cambia el ancho del recuadro de 200 píxeles a 300 píxeles en forma instantanea al pasar la flecha del mouse.

Ahora si definimos la propiedad transition indicando que actúe sobre la propiedad width y que el cambio lo realice en 1 segundo:

```
#recuadro2{
  border-radius: 20px;
  background-color:#ff0;
  width:200px;
  padding:10px;
  transition:width 1s;
  -moz-transition:width 1s;
  -ms-transition:width 1s;
  -webkit-transition:width 1s;
  -o-transition:width 1s;
}

#recuadro2:hover{
  width:300px;
}
```

Tenemos la siguiente sintaxis:

```
transition:width 1s;
```

Como vemos indicamos que la transición afecta a la propiedad width y que el cambio se realice en 1 segundo. Debemos agregar la propiedad con cada prefijo de navegador hasta que se estabilice su uso.

24) Transiciones de múltiples propiedades (transition)

También podemos hacer transiciones de múltiples propiedades, para ello indicamos cada transición separada por coma:

```
Elemento {
    transition: [nombre de propiedad] [duración de la
transición],
               [nombre de propiedad] [duración de la
transición],
```

[nombre de propiedad] [duración de la transición]

```
;  
}
```

Por ejemplo si queremos que el recuadro modifique su ancho y cambie de color el código es el siguiente:

```
#recuadro1{  
  border-radius: 20px;  
  background-color:#ff0;  
  width:200px;  
  padding:10px;  
  transition:width 1s,  
              background-color 8s;  
  -moz-transition:width 1s,  
                  background-color 8s;  
  -ms-transition:width 1s,  
                  background-color 8s;  
  -webkit-transition:width 1s,  
                      background-color 8s;  
  -o-transition:width 1s,  
                  background-color 8s;  
}  
  
#recuadro1:hover{  
  width:300px; background-  
  color: #f00;  
}
```

Como vemos en la propiedad transition indicamos la propiedad width con una duración de un segundo y la propiedad background-color con un valor de 8 segundos. Esto significa que cuando dispongamos la flecha del mouse dentro del div se lanzarán ambas transiciones que tienen duraciones distintas (cambiará de 200 píxeles a 300 píxeles en el lapso de un segundo y también cambiará del color amarillo al rojo en forma gradual en un lapso de 8 segundos)

Transiciones (funciones de transición)

Un tercer parámetro opcional de la propiedad transition es indicar una "función de transición" que nos permite seleccionar la velocidad durante la transición:

```
Elemento {  
  transition: [nombre de propiedad] [duración de la  
transición] [función de transición];  
}
```

Los valores posibles que podemos especificar son:

✈ease : Define un efecto de transición con un comienzo lento, luego rápido y finalmente termina lento (cuando no definimos la función de transición elige esta por defecto)

✈linear : Define un efecto de transición con la misma velocidad de inicio a fin.

✈ease-in : Define un efecto de transición con un comienzo lento.

✈ease-out : Define un efecto de transición con un final lento.

✈ease-in-out : Define un efecto de transición con un comienzo lento y un final lento.

La sintaxis si queremos aplicar la función de transición linear será:

```
#recuadro2{
  margin: 5px;
  border-radius: 20px;
  background-color:#eee;
  width:200px;
  padding:10px;
  transition:width 2s linear;
  -moz-transition:width 2s linear;
  -ms-transition:width 2s linear;
  -webkit-transition:width 2s linear;
  -o-transition:width 2s linear;
}
```

Tiempo de demora en iniciar la transición)

El cuarto parámetro opcional de la propiedad transition es indicar **un tiempo de espera hasta que se inicie la transición:**

```
Elemento {
  transition: [nombre de propiedad] [duración de la
transición]
              [función de transición] [tiempo de inicio];
}
```

Es decir indicamos la cantidad de milisegundos o segundos hasta que se inicia el proceso de transición.

Por ejemplo si queremos que la transición comience en 1 segundo la sintaxis será la siguiente:

```
#recuadro1{
  border-radius: 20px;
  background-color:#eee;
  width:200px;
  padding:10px;
  transition:width 1s ease 1s;
  -moz-transition:width 1s ease 1s;
  -ms-transition:width 1s ease 1s;
  -webkit-transition:width 1s ease 1s;
  -o-transition:width 1s ease 1s;
}
```

Puede ser muy útil si queremos encadenar transiciones, por ejemplo si queremos que el elemento modifique su ancho y luego al finalizar el cambio del ancho modifique su color:

```
#recuadro2{
  margin:5px;
  border-radius: 20px;
  background-color:#eee;
  width:200px;
  padding:10px;
  transition:width 1s ease,
              background-color 1s ease 1s;
  -moz-transition:width 1s ease,
                  background-color 1s ease 1s;
  -ms-transition:width 1s ease,
                  background-color 1s ease 1s;
  -webkit-transition:width 1s ease,
```

```

        background-color 1s ease 1s;
    -o-transition:width 1s ease,
        background-color 1s ease 1s;
}

#recuadro2:hover{
    width:300px; background-
    color:#ff0;
}

```

25) Animaciones (sintaxis básica)

Las animaciones en css3 nos permiten hacer cosas que con las transiciones no se pueden.

La sintaxis básica para una animación:

```

Elemento {
    animation-name: [nombre de la animación];
    animation-duration: [tiempo de duración];
}

@ keyframes [nombre de la animación] {
    from {
        [propiedades y valores del estado inicial de la animación]
    }
    to {
        [propiedades y valores del estado final de la animación]
    }
}

```

Veamos un ejemplo donde emplearemos la sintaxis expuesta, en este momento solo el FireFox, Chrome y Opera implementan las animaciones. Mostraremos un recuadro con un texto que parta de la columna 30px y avance hasta la columna 300px y que cambie del color gris al color amarillo:

```

#recuadro1{
    left:30px;
    position:relative;
    border-radius: 20px;
    background-color:#ddd;
    width:200px;
    height:100px;
    padding:4px;
    -moz-animation-name: animacion1;
    -moz-animation-duration: 4s;
    -webkit-animation-name: animacion1;
    -webkit-animation-duration: 4s;
    -o-animation-name: animacion1;
    -o-animation-duration: 4s;
}

@-moz-keyframes animacion1 {
    from {
        left:30px;
        background-color:#ddd;
    }
}

```

```

    to {
      left:300px;
      background-color:#ff0;
    }
  }
}

@-webkit-keyframes animacion1 {
  from {
    left:30px;
    background-color:#ddd;
  }
  to {
    left:300px;
    background-color:#ff0;
  }
}

@-o-keyframes animacion1 {
  from {
    left:30px;
    background-color:#ddd;
  }
  to {
    left:300px;
    background-color:#ff0;
  }
}

```

Como podemos ver inicializamos las propiedades animation-name con el nombre de la animación:

```

-moz-animation-name: animacion1;
-webkit-animation-name: animacion1;
-o-animation-name: animacion1;

```

Y la propiedad animation-duration con el tiempo duración de la animación:

```

-moz-animation-duration: 4s;
-webkit-animation-duration: 4s;
-o-animation-duration: 4s;

```

La sintaxis para especificar la animación propiamente dicha es idéntica para cada navegador pero como debemos utilizar el prefijo creamos tres:

```

@-moz-keyframes animacion1 {
  from {
    left:30px;
    background-color:#ddd;
  }
  to {
    left:300px;
    background-color:#ff0;
  }
}

```

En la parte del from indicamos el estado inicial del elemento (en este caso 30 píxeles respecto a la izquierda y de color gris (#ddd)), luego en la sección del to indicamos los valores finales que debe tomar el elemento (300 píxeles respecto a la izquierda y de color amarillo(#ff0))

26) Animaciones (animation-iteration-count y animation-direction)

Otras dos propiedades muy importantes para crear animaciones con css3 son animation-iteration-count y animation-direction:

```
Elemento {  
    animation-iteration-count: [cantidad de veces a repetir la animación o "infinite"];  
    animation-direction: ["normal" o "alternate"];  
}
```

Si queremos que la animación se repita solo tres veces indicamos en la propiedad animation-iteration-count dicho valor:

```
-moz-animation-iteration-count: 3;
```

En cambio si queremos que se repita siempre luego especificamos el valor "infinite":

```
-moz-animation-iteration-count: infinite;
```

La propiedad animation-direction indica como debe repetirse la animación, puede tomar el valor "normal" con lo que la animación cada vez que finaliza comienza desde el principio. Pero si inicializamos con el valor "alternate" cuando finaliza la animación comienza desde el final hasta el principio:

```
-moz-animation-direction: alternate;
```

El ejemplo del concepto anterior que muestra un recuadro con un texto que se desplaza de izquierda a derecha cambiando su color, pero ahora indicando que se repita en forma indefinida y con el valor "alternate" en la propiedad animation-direction:

```
#recuadro1{  
    left:30px;  
    position:relative;  
    border-radius: 20px;  
    background-color:#ddd;  
    width:200px;  
    height:100px;  
    padding:4px;  
    -moz-animation-name: animacion1;  
    -moz-animation-duration: 4s;  
    -moz-animation-iteration-count: infinite;  
    -moz-animation-direction: alternate;  
    -webkit-animation-name: animacion1;  
    -webkit-animation-duration: 4s;  
    -webkit-animation-iteration-count: infinite;  
    -webkit-animation-direction: alternate;  
    -o-animation-name: animacion1;  
    -o-animation-duration: 4s;  
    -o-animation-iteration-count: infinite;  
    -o-animation-direction: alternate;  
}
```

```
@-moz-keyframes animacion1 {  
    from {  
        left:30px;  
        background-color:#ddd;
```

```

    }
    to {
        left:300px;
        background-color:#ff0;
    }
}

@-webkit-keyframes animacion1 {
    from {
        left:30px;
        background-color:#ddd;
    }
    to {
        left:300px;
        background-color:#ff0;
    }
}

@-o-keyframes animacion1 {
    from {
        left:30px;
        background-color:#ddd;
    }
    to {
        left:300px;
        background-color:#ff0;
    }
}

```

26) Animaciones (animation-timing-function y animation-delay)

Similar a las transiciones disponemos de dos propiedades para definir la función de transición y el tiempo que debe esperar para comenzar la animación:

```

Elemento {
    animation-timing-function: [función de transición];
    animation-delay: [tiempo de demora para iniciar la animación];
}

```

Los valores posibles para la propiedad animation-timing-function:

- ✈ease : Define un efecto de animación con un comienzo lento, luego rápido y finalmente termina lento (cuando no definimos la función elige esta por defecto)
- ✈linear : Define un efecto de animación con la misma velocidad de inicio a fin.
- ✈ease-in : Define un efecto de animación con un comienzo lento.
- ✈ease-out : Define un efecto de animación con un final lento.
- ✈ease-in-out : Define un efecto de animación con un comienzo lento y un final lento.

Definimos la función de animación "linear" y un tiempo de retardo de 2 segundos:

```

#recuadro1{

```

```
left:30px;
position:relative;
border-radius: 20px;
background-color:#ddd;
width:200px;
height:100px;
padding:4px;
-moz-animation-name: animacion1;
-moz-animation-duration: 4s;
-moz-animation-iteration-count: infinite;
-moz-animation-direction: alternate;
-moz-animation-timing-function: linear;
-moz-animation-delay: 2s;
-webkit-animation-name: animacion1;
-webkit-animation-duration: 4s;
-webkit-animation-iteration-count: infinite;
-webkit-animation-direction: alternate;
-webkit-animation-timing-function: linear;
-webkit-animation-delay: 2s;
-o-animation-name: animacion1;
-o-animation-duration: 4s;
-o-animation-iteration-count: infinite;
-o-animation-direction: alternate;
-o-animation-timing-function: linear;
-o-animation-delay: 2s;
}
```

```
@-moz-keyframes animacion1 {
  from {
    left:30px;
    background-color:#ddd;
  }
  to {
    left:300px;
    background-color:#ff0;
  }
}
```

```
@-webkit-keyframes animacion1 {
  from {
    left:30px;
    background-color:#ddd;
  }
  to {
    left:300px;
    background-color:#ff0;
  }
}
```

```
@-o-keyframes animacion1 {
  from {
    left:30px;
    background-color:#ddd;
```

```

}
to {
  left:300px;
  background-color:#ff0;
}
}

```

28) Animaciones (animation-play-state)

La propiedad animation-play-state nos permite cambiar el estado de la animación, los valores posibles son "running" y "paused", es decir ejecutándose o pausada. La sintaxis luego es:

```

Elemento {
  animation-play-state: [running o paused];
}

```

Por defecto una animación comienza con esta propiedad con el valor "running".

El siguiente ejemplo muestra un recuadro que cuando ponemos la flecha del mouse en su interior se activa la animación cambiando por esquinas redondeadas y el color interior:

```

#recuadro1{
  border-radius: 20px;
  background-color:#ddd;
  width:200px;
  height:100px;
  padding:8px;
  -moz-animation-play-state:paused;
  -moz-animation-name: animacion1;
  -moz-animation-duration: 4s;
  -moz-animation-iteration-count: infinite;
  -moz-animation-direction: alternate;
  -webkit-animation-play-state:paused;
  -webkit-animation-name: animacion1;
  -webkit-animation-duration: 4s;
  -webkit-animation-iteration-count: infinite;
  -webkit-animation-direction: alternate;
  -o-animation-play-state:paused;
  -o-animation-name: animacion1;
  -o-animation-duration: 4s;
  -o-animation-iteration-count: infinite;
  -o-animation-direction: alternate;
}

#recuadro1:hover {
  -moz-animation-play-state:running;
  -webkit-animation-play-state:running;
  -o-animation-play-state:running;
}

@-moz-keyframes animacion1 {
  from {
    border-radius:0px;
    background-color:#ddd;
  }
  to {

```

```

    border-radius:40px;
    background-color:#ff0;
  }
}

@-webkit-keyframes animacion1 {
  from {
    border-radius:0px;
    background-color:#ddd;
  }
  to {
    border-radius:40px;
    background-color:#ff0;
  }
}

@-o-keyframes animacion1 {
  from {
    border-radius:0px;
    background-color:#ddd;
  }
  to {
    border-radius:40px;
    background-color:#ff0;
  }
}

```

29) Animaciones (definición de más de 2 keyframes)

Hasta ahora hemos indicado en la animación solo 2 keyframes (el inicial y el final), pero para animaciones más complejas es posible que necesitemos más de 2 keyframes, para esto tenemos la siguiente sintaxis:

```

@-moz-keyframes [nombre de la animación] {
  0%{
    [propiedades y valores]
  }
  [valor en porcentaje]%{
    [propiedades y valores]
  }
  [valor en porcentaje]%{
    [propiedades y valores]
  }
  100%{
    [propiedades y valores]
  }
}

```

Por ejemplo si queremos 5 frames claves que nos permitan desplazar un recuadrado sobre un perímetro de un rectángulo tenemos que especificar los siguientes frames:

```

#recuadro1{
  left:30px;
  top:30px;
  position:relative;
}

```



```
border-radius: 20px;
background-color:#ddd;
width:100px;
height:100px;
padding:4px;
-moz-animation-name: animacion1;
-moz-animation-duration: 4s;
-moz-animation-iteration-count: infinite;
-webkit-animation-name: animacion1;
-webkit-animation-duration: 4s;
-webkit-animation-iteration-count: infinite;
-o-animation-name: animacion1;
-o-animation-duration: 4s;
-o-animation-iteration-count: infinite;
}
```

```
@-moz-keyframes animacion1 {
  0%{
    left:30px;
    top:30px;
  }
  25%{
    left:300px;
    top:30px;
  }
  50%{
    left:300px;
    top:200px;
  }
  75%{
    left:30px;
    top:200px;
  }
  100%{
    left:30px;
    top:30px;
  }
}
```

```
@-webkit-keyframes animacion1 {
  0%{
    left:30px;
    top:30px;
  }
  25%{
    left:300px;
    top:30px;
  }
  50%{
    left:300px;
    top:200px;
  }
  75%{
```

```

    left:30px;
    top:200px;
  }
  100%{
    left:30px;
    top:30px;
  }
}

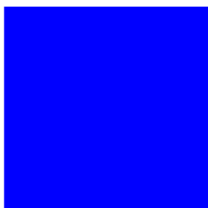
@-o-keyframes animacion1 {
  0%{
    left:30px;
    top:30px;
  }
  25%{
    left:300px;
    top:30px;
  }
  50%{
    left:300px;
    top:200px;
  }
  75%{
    left:30px;
    top:200px;
  }
  100%{
    left:30px;
    top:30px;
  }
}

```

CSS3 Shapes

Here you'll find a range off shapes all coded with just pure CSS3 code. Unfortunately not all shapes will render correctly in all browsers, currently only web browsers that support CSS3 will produce the correct geometric shapes.

Square



```

#square {
  width: 140px;
  height: 140px;
  background: blue;
}

```

Circle



```
#circle {  
  width: 140px;  
  height: 140px;  
  background: blue;  
  -moz-border-radius: 70px;  
  -webkit-border-radius: 70px;  
  border-radius: 70px;  
}
```

Oval



```
#oval {  
  width: 200px;  
  height: 100px;  
  background: blue;  
  -moz-border-radius: 100px / 50px;  
  -webkit-border-radius: 100px / 50px;  
  border-radius: 100px / 50px;  
}
```

Up Triangle



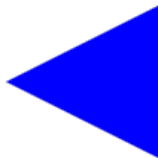
```
#up-triangle {  
  width: 0;  
  height: 0;  
  border-bottom: 120px solid blue;  
  border-left: 60px solid transparent;  
  border-right: 60px solid transparent;  
}
```

Down Triangle



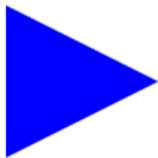
```
#down-triangle {  
  width: 0;  
  height: 0;  
  border-top: 120px solid blue;  
  border-left: 60px solid transparent;  
  border-right: 60px solid transparent;  
}
```

Left Triangle



```
#left-triangle {  
  width: 0;  
  height: 0;  
  border-right: 100px solid blue;  
  border-top: 50px solid transparent;  
  border-bottom: 50px solid transparent;  
}
```

Right Triangle



```
#right-triangle {  
  width: 0;  
  height: 0;  
  border-left: 100px solid blue;  
  border-top: 50px solid transparent;  
  border-bottom: 50px solid transparent;  
}
```

Triangle Top Left



```
#triangle-topleft {  
  width: 0;  
  height: 0;  
  border-top: 100px solid blue;  
  border-right: 100px solid transparent;  
}
```

Triangle Top Right



```
#triangle-topright {  
  width: 0;  
  height: 0;  
  border-top: 100px solid blue;  
  border-left: 100px solid transparent;  
}
```

Triangle Bottom Left



```
#triangle-bottomleft {  
  width: 0;  
  height: 0;  
  border-bottom: 100px solid blue;  
  border-right: 100px solid transparent;  
}
```

Triangle Bottom Right



```
#triangle-bottomright {  
  width: 0;  
  height: 0;  
  border-bottom: 100px solid blue;  
  border-left: 100px solid transparent;  
}
```

Trapezium



```
#trapezium {  
  height: 0;  
  width: 80px;  
  border-bottom: 80px solid blue;  
  border-left: 40px solid transparent;  
  border-right: 40px solid transparent;  
}
```

Diamond



```
#diamond {  
  width: 80px;  
  height: 80px;  
  background: blue;  
  margin: 3px 0 0 30px;  
  /* Rotate */  
  -webkit-transform: rotate(-45deg);  
  -moz-transform: rotate(-45deg);  
  -ms-transform: rotate(-45deg);  
  -o-transform: rotate(-45deg);  
  transform: rotate(-45deg);  
  /* Rotate Origin */  
  -webkit-transform-origin: 0 100%;  
  -moz-transform-origin: 0 100%;  
  -ms-transform-origin: 0 100%;  
  -o-transform-origin: 0 100%;  
  transform-origin: 0 100%;  
}
```

Rectangle



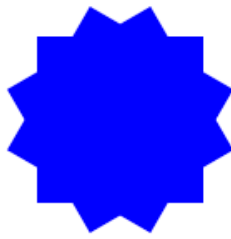
```
#rectangle {  
  width: 140px;  
  height: 80px;  
  background: blue;  
}
```

Parallelogram



```
#parallelogram {  
  width: 130px;  
  height: 75px;  
  background: blue;  
  /* Skew */  
  -webkit-transform: skew(20deg);  
  -moz-transform: skew(20deg);  
  -o-transform: skew(20deg);  
  transform: skew(20deg);  
}
```


Twelve Point Star



```
#twelve-point-star {  
  height: 100px;  
  width: 100px;  
  background: blue;  
  position: absolute;  
}  
#twelve-point-star:before {  
  height: 100px;  
  width: 100px;  
  background: blue;  
  content:"";  
  position: absolute;  
  /* Rotate */  
  -moz-transform: rotate(30deg);  
  -webkit-transform: rotate(30deg);  
  -ms-transform: rotate(30deg);  
  -o-transform: rotate(30deg);  
  transform: rotate(30deg);  
}
```

```
#twelve-point-star:after {  
  height: 100px;  
  width: 100px;  
  background: blue;  
  content:"";  
  position: absolute;  
  /* Rotate */  
  -moz-transform: rotate(-30deg);  
  -webkit-transform: rotate(-30deg);  
  -ms-transform: rotate(-30deg);  
  -o-transform: rotate(-30deg);  
  transform: rotate(-30deg);  
}
```

Six Point Star



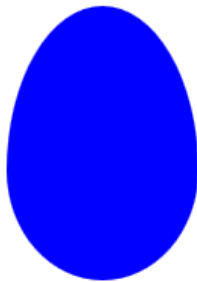
```
#six-point-star {  
  position: absolute;  
  width: 0;  
  height: 0;  
  border-left: 50px solid transparent;  
  border-right: 50px solid transparent;  
  border-bottom: 80px solid blue;  
}  
#six-point-star:after {  
  content:"";  
  position: absolute;  
  width: 0;  
  height: 0;  
  border-left: 50px solid transparent;  
  border-right: 50px solid transparent;  
  border-top: 80px solid blue;  
  margin: 30px 0 0 -50px;  
}
```

Speech Bubble



```
#speech-bubble {
  width: 120px;
  height: 80px;
  background: blue;
  position: absolute;
  -moz-border-radius: 10px;
  -webkit-border-radius: 10px;
  border-radius: 10px;
}
#speech-bubble:before {
  content: "";
  position: absolute;
  width: 0;
  height: 0;
  border-top: 13px solid transparent;
  border-right: 26px solid blue;
  border-bottom: 13px solid transparent;
  margin: 13px 0 0 -25px;
}
```

Egg



```
#egg {
  display: block;
  width: 126px;
  /* width has to be 70% of height */
  /* could use width: 70%; in a square container */
  height: 180px;
  background-color: blue;

  /* beware that Safari needs actual
  px for border radius (bug) */
  -webkit-border-radius: 63px 63px 63px 63px /
  108px 108px 72px 72px;
  /* fails in Safari, but overwrites in Chrome */
  border-radius: 50% 50% 50% 50% / 60% 60% 40% 40%;
}
```

EQ Triangle



```
#eq-triangle {
  width: 0;
  height: 0;
  border-bottom: 104px solid blue;
  /* 104 = 120 * 0.866 */
  border-left: 60px solid transparent;
  border-right: 60px solid transparent;
}
```

Pacman



```
#pacman {  
  width: 0px;  
  height: 0px;  
  border-right: 60px solid transparent;  
  border-top: 60px solid blue;  
  border-left: 60px solid blue;  
  border-bottom: 60px solid blue;  
  border-top-left-radius: 60px;  
  border-top-right-radius: 60px;  
  border-bottom-left-radius: 60px;  
  border-bottom-right-radius: 60px;  
}
```

Biohazard



```
#biohazard {  
  width: 0;  
  height: 0;  
  border-bottom: 60px solid black;  
  border-top: 60px solid black;  
  border-left: 60px solid yellow;  
  border-right: 60px solid yellow;  
  -moz-border-radius: 60px;  
  -webkit-border-radius: 60px;  
  border-radius: 60px;  
}
```

Heart



```
#heart {  
  position: relative;  
}  
#heart:before, #heart:after {  
  position: absolute;  
  content: "";  
  left: 70px; top: 0;  
  width: 70px;  
  height: 115px;  
  background: blue;  
  -moz-border-radius: 50px 50px 0 0;  
  border-radius: 50px 50px 0 0;  
  -webkit-transform: rotate(-45deg);  
  -moz-transform: rotate(-45deg);  
  -ms-transform: rotate(-45deg);  
  -o-transform: rotate(-45deg);  
  transform: rotate(-45deg);  
  -webkit-transform-origin: 0 100%;  
  -moz-transform-origin: 0 100%;  
  -ms-transform-origin: 0 100%;  
  -o-transform-origin: 0 100%;  
  transform-origin: 0 100%;  
}
```

```

}
#heart:after {
left: 0;
-webkit-transform: rotate(45deg);
-moz-transform: rotate(45deg);
-ms-transform: rotate(45deg);
-o-transform: rotate(45deg);
transform: rotate(45deg);
-webkit-transform-origin: 100% 100%;
-moz-transform-origin: 100% 100%;
-ms-transform-origin: 100% 100%;
-o-transform-origin: 100% 100%;
transform-origin :100% 100%;
}

```

Yin Yang



```

#yin-yang {
width: 96px;
height: 48px;
background: #eee;
border-color: red;
border-style: solid;
border-width: 2px 2px 50px 2px;
border-radius: 100%;
position: relative;
}
#yin-yang:before {
content: "";
position: absolute;
top: 50%;
left: 0;
background: #eee;
border: 18px solid red;
border-radius: 100%;
width: 12px;
height: 12px;
}

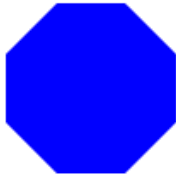
```

```

#yin-yang:after {
content: "";
position: absolute;
top: 50%;
left: 50%;
background: red;
border: 18px solid #eee;
border-radius: 100%;
width: 12px;
height: 12px;
}

```

Octagon



```
#octagon {  
  width: 100px;  
  height: 100px;  
  background: blue;  
  position: relative;  
}  
#octagon:before {  
  height: 0;  
  width: 40px;  
  content: "";  
  position: absolute;  
  border-bottom: 30px solid blue;  
  border-left: 30px solid white;  
  border-right: 30px solid white;  
}
```

```
#octagon:after {  
  height: 0;  
  width: 40px;  
  content: "";  
  position: absolute;  
  border-top: 30px solid blue;  
  border-left: 30px solid white;  
  border-right: 30px solid white;  
  margin: 70px 0 0 0;  
}
```

➤ Font Face:

Especifica un nombre para una fuente y la URL donde se puede encontrar.

Example

```
@font-face {  
  font-family: myFirstFont;  
  src: url(sansation_light.woff);  
}
```

Se usa:

```
div {  
  font-family: myFirstFont;  
}
```

www.dafont.com/

Dirección para bajar fuentes.

El diseño web adaptativo:

En la actualidad la mayoría de los sitios web adaptan su diseño y su formato al tamaño de la pantalla en la que se muestran. La página se va a adaptar automáticamente en función de si se visualiza en la pantalla de un ordenador, en una tableta o en un teléfono inteligente. La expresión inglesa para designar esta arquitectura web es **Responsive Web Design**, que podemos traducir por **diseño web adaptativo**, aunque también se usa **diseño web reactivo**. Para ello, debemos utilizar los **Media Queries**.

Los Media Queries

- **La recomendación del W3C**

Para crear un diseño adaptado a los diferentes tamaños de pantalla, el W3C facilita el módulo **Media Queries**. Con los Media Queries puede crear hojas de estilo que se adapten a los diferentes tamaños de la pantalla. La detección del tamaño de la pantalla se lleva a cabo mediante la «consulta» al medio, tras la cual el navegador utiliza los estilos adaptados.

Los criterios de los Media Queries

El valor que devuelve un Media Query es de tipo booleano: verdadero o falso. El navegador escoge la hoja de estilos adaptada en función de las respuestas a los Media Queries.

Estos son los criterios que podemos utilizar en los Media Queries:

- La anchura de visualización: `width`. Podemos testar la anchura del área de visualización del navegador. Por ejemplo: `width: 780px`.
- La altura de visualización: `height`. Podemos testar la altura del área de visualización del navegador.
- La anchura física: `device-width`. Podemos testar la anchura física de la pantalla de difusión.
- La altura física: `device-height`. Podemos testar la altura física de la pantalla de difusión.
- La orientación de la pantalla: `orientation`. Por ejemplo: `orientation: portrait` u `orientation: landscape`. Muy práctico para testar si el usuario usa su tableta táctil en posición vertical (`portrait`) u horizontal (`landscape`).
- La ratio: `aspect-ratio`. Para testar el valor de la proporción anchura/altura. Por ejemplo: `aspect-ratio: 16/9`.
- La ratio física: `device-aspect-ratio`. Para testar el valor de la proporción física anchura/altura de la pantalla.
- El color: `color`. Podemos testar si el soporte de difusión usa colores (valor por defecto si no se usan), es en blanco y negro o en escalas de gris. Por ejemplo: `min-color: 8`.
- El número de colores en la tabla de colores: `color-index`.
- El número de niveles de gris en los aparatos monocromos: `monochrome`.
- La resolución de la pantalla de salida: `resolution`. Se expresa en dpi.
- El tipo de barrido para las pantallas de televisión: `scan`.
- Utilice `grid` para testar si la pantalla de difusión emplea una cuadrícula con un solo tamaño de fuente.

La sintaxis de los Media Queries

Veamos ahora un ejemplo concreto. Queremos detectar los periféricos que tengan una anchura exacta de 780 píxeles.

En una página HTML, en el <head>, tenemos el siguiente Query con el elemento <link>:

```
<link rel="stylesheet" href="styles780.css" />
```

En el archivo CSS llamado **styles780.css**, este es el Query con la regla @:

```
@media screen and (width:780px) {  
    ...  
}
```

Indicamos que el tipo de medio es una pantalla: `screen`.

Indicamos que hay un segundo criterio: `and`.

Indicamos que la anchura de estas pantallas debe ser igual a 780 píxeles: `width: 780px`.

Todos los estilos que se han de usar se ponen entre llaves.

Los valores mínimos y máximos

Todos los criterios que acabamos de ver, excepto `orientation`, `scan` y `grid`, pueden utilizar los prefijos `min-` y `max-` junto a sus valores (de los criterios).

Por ejemplo, vamos a testar las pantallas cuya anchura sea como máximo de 780 píxeles.

```
@media screen and (max-width:780px) {  
    ...  
}
```

Esto es muy útil, ya que resulta raro testar un tamaño fijo para las pantallas de ordenador, por ejemplo.

Los operadores lógicos

Los Media Queries ofrecen operadores lógicos a fin de precisar y combinar los diferentes criterios.

El operador `and` permite utilizar el operador lógico **Y**.

Primer ejemplo: queremos determinar las pantallas que tienen una anchura de 780 píxeles y que son monocromas:

```
@media screen and (width:780px) and monochrome {  
    ...  
}
```

Segundo ejemplo: queremos determinar las pantallas que tienen una resolución comprendida entre un mínimo de 1024 píxeles y un máximo de 1280 píxeles:

```
@media screen and (min-width: 1024px) and (max-width: 1280px) {  
    ...  
}
```

El operador `not` permite utilizar el operador lógico **NO**.

Por ejemplo: queremos determinar las pantallas cuya resolución no es de 780 píxeles:

```
@media screen and (not width:780px) {  
    ...  
}
```

La coma (,) permite usar el operador lógico **O**.

Por ejemplo, queremos determinar las pantallas de 780 píxeles de ancho o aquellas que son monocromas:

```
@media screen and (width:780px), monochrome {  
    ...  
}
```

La palabra clave `only` permite precisar que el Query debe aplicarse únicamente sobre los criterios indicados. Esto permite ocultar los estilos para navegadores antiguos.

```
@media only screen and (width:780px) {  
    ...  
}
```

El tamaño de las pantallas

El tamaño físico y la visualización

En la actualidad, cada tipo de teléfono inteligente y de tableta tiene sus propias características técnicas relativas al tamaño físico de las pantallas y a la superficie realmente disponible para la visualización de un sitio en la versión móvil de un navegador.

Un segundo problema se deriva de lo que realmente se muestra en función de las características de la pantalla. Cuando salió el iPhone de Apple en 2007, el tamaño de su pantalla se había fijado a 320 x 480 píxeles. Pero Safari Mobile muestra los sitios con una anchura de 980 píxeles, frente a los 320 píxeles de la pantalla, lo que implica una disminución sustancial del área de visualización.

Con el iPhone, Apple ha introducido un nuevo valor para el atributo `name: viewport`. Este permite controlar la ratio anchura del sitio (980 píxeles)/anchura de visualización (320 píxeles) aplicada por Safari en su versión móvil.

Si queremos que Safari para iPhone muestre los sitios con una anchura de 320 píxeles (y no con 980 píxeles), debemos utilizar el valor `viewport` para el atributo `name`:

```
<meta name="viewport" content="width=320" />
```

En la actualidad, este atributo ya ha sido adoptado por todos los fabricantes de teléfonos inteligentes y de tabletas. Se ha convertido en un estándar *de facto*.

El problema que aún queda pendiente es que el tamaño de las pantallas no está en absoluto armonizado entre los distintos teléfonos inteligentes y otras tabletas. Debemos adaptar la visualización en función de las características propias de cada aparato usando el valor `device-width`, que determina la anchura física de la pantalla.

```
<meta name="viewport" content="width=device-width" />
```

De este modo, la visualización se adaptará a la superficie y las capacidades de cada navegador para aparatos móviles.

Los zooms en pantalla

En `viewport`, dentro del atributo `content`, puede agregar otros valores relacionados con el zoom en pantalla. Estos otros valores se separan de los anteriores con una coma.

El valor `initial-scale` especifica el nivel de zoom inicial de la página cuando esta se carga. Ello permite que los aparatos «respeten» la anchura especificada en los Media Queries.

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

El valor `user-scalable` define el nivel de zoom autorizado por la acción del usuario. Un valor de 0 prohíbe al usuario el uso del zoom. Un valor de 1 autoriza la utilización del zoom.

```
<meta name="viewport" content="width=device-width, user-scalable=0" />
```

Los valores `minimum-scale` y `maximum-scale` permiten definir los zooms mínimos y máximos autorizados.

```
<meta name="viewport" content="width=device-width,
minimum-scale=0.5, maximum-scale=3.0, initial-scale=1.0,
user-scalable=1" />
```

El sitio **mydevice.io** (<http://mydevice.io/devices/>) proporciona los valores `width` y `device-width` de los teléfonos inteligentes, las *phablets* (teléfonos inteligentes con pantallas muy grandes) y las tabletas más corrientes.

El sitio **screensiz.es** (<http://screensiz.es/phone>) ofrece el mismo servicio.

Imágenes adaptativas

Las imágenes también pueden comportar problemas de tamaño en relación con su elemento padre. Si la imagen es más ancha que su contenedor, se desborda.

La solución para evitar este problema (y el de los cambios consecutivos de tamaño en los Media Queries) consiste en imponer una anchura máxima en relación con el elemento contenedor, el elemento padre de la imagen. Para ello, es preciso utilizar la propiedad `max-width`.

Añadamos este estilo:

```
#miCaja img {
    max-width: 100%;
}
```

Lo que pedimos es que la anchura máxima de las imágenes insertadas en la caja `<div id="miCaja">` sea del 100 %. De este modo, las imágenes nunca serán más anchas que su elemento padre.

Las cajas flexibles

El principio en el que se basa la utilización de las cajas flexibles es muy simple: basta con definir un contenedor padre como caja flexible para que todos los elementos que contiene, los elementos hijos, adopten su comportamiento.

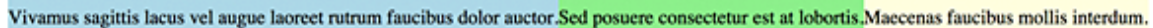
La propiedad `display` acepta dos valores: `flex` para una visualización flexible en bloque e `inline-flex` para una visualización en línea.

Veamos a continuación un primer ejemplo muy sencillo. Definimos una caja contenedor padre `<div id="caja">` con una visualización en caja flexible: `display: flex;`. Automáticamente, todos sus elementos hijos, tres cajas `<div>` con fondo de color en este ejemplo, se muestran unas al lado de otras.

He aquí el código utilizado:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Cajas flexibles</title>
  <meta charset="UTF-8" />
  <style>
    #caja {
      display: flex;
    }
    .azul {
      background-color: lightblue;
    }
    .verde {
      background-color: lightgreen;
    }
    .amarillo {
      background-color: lightyellow;
    }
  </style>
</head>
<body>
<div id="caja">
  <div class="azul">
    <p>Vivamus sagittis...</p>
  </div>
  <div class="verde">
    <p>Sed posuere consectetur...</p>
  </div>
  <div class="amarillo">
    <p>Maecenas faucibus...</p>
  </div>
</div>
</body>
</html>
```

Esto es lo que se obtiene:



Vivamus sagittis lacus vel augue laoreet rutrum faucibus dolor auctor.Sed posuere consectetur est at lobortis.Maecenas faucibus mollis interdum.

La orientación y el sentido de las cajas

Por defecto, las cajas hijas se muestran unas al lado de otras, en una línea. Es la propiedad `flex-direction` la que determina esta orientación de las cajas hijas con los valores `row` (comportamiento por defecto) y `column`.

Este ejemplo (**10_06.html**) muestra las cajas hijas unas debajo de otras:

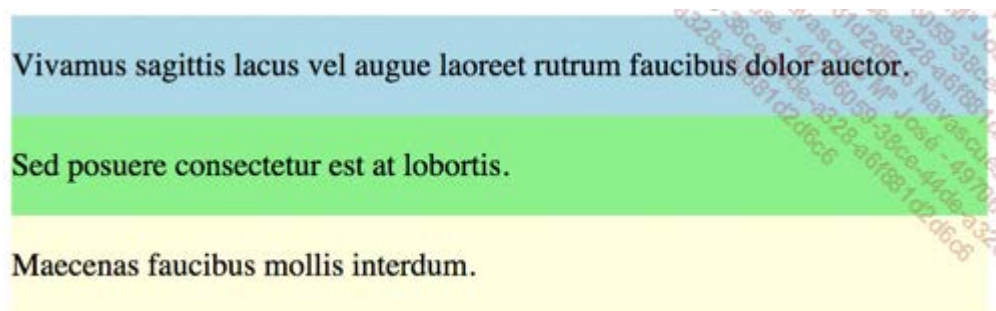
```
<!DOCTYPE html>
<html lang="es">
<head>
```

```

<title>Cajas flexibles</title>
<meta charset="UTF-8" />
<link href="#" rel="stylesheet" />
<style>
#caja {
    display: flex;
    flex-direction: column;
}
.azul {
    background-color: lightblue;
}
.verde {
    background-color: lightgreen;
}
.amarillo {
    background-color: lightyellow;
}
</style>
</head>
<body>
<div id="caja">
    <div class="azul">
        <p>Vivamus sagittis lacus vel augue laoreet rutrum
        faucibus dolor auctor.</p>
    </div>
    <div class="verde">
        <p>Sed posuere consectetur est at lobortis.</p>
    </div>
    <div class="amarillo">
        <p>Maecenas faucibus mollis interdum.</p>
    </div>
</div>
</body>
</html>

```

Esto es lo que se obtiene:



Los valores `column-reverse` y `row-reverse` permiten invertir el sentido en que se visualizan los elementos.

El desborde de las cajas

Podemos gestionar el comportamiento de las cajas hijas cuando estas sobrepasan o desbordan la caja flexible padre con la propiedad `flex-wrap`. El valor `nowrap` impone una visualización en una sola línea aun cuando los elementos hijos se desborden, incluso aunque sean más anchos que la caja flexible padre. La anchura de las cajas hijas se modificará para que estas quepan en la línea de la caja padre.

Este es el código utilizado:

```

<!DOCTYPE html>
<html lang="es">
<head>
    <title>Cajas flexibles</title>
    <meta charset="UTF-8" />
    <style>
    #caja {
        width: 600px;
        display: flex;
        flex-direction: row;
        flex-wrap: nowrap;
    }
    </style>
</head>
</html>

```

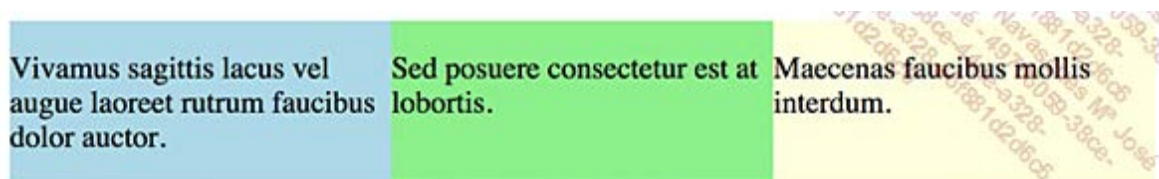


```

    }
    .azul {
        width: 250px;
        background-color: lightblue;
    }
    .verde {
        width: 250px;
        background-color: lightgreen;
    }
    .amarillo {
        width: 250px;
        background-color: lightyellow;
    }
</style>
</head>
<body>
<div id="caja">
    <div class="azul">
        <p>Vivamus sagittis lacus vel...</p>
    </div>
    <div class="verde">
        <p>Sed posuere consectetur...</p>
    </div>
    <div class="amarillo">
        <p>Maecenas faucibus mollis...</p>
    </div>
</div>
</body>
</html>

```

Esto es lo que se obtiene:



Con valor wrap, las cajas hijas conservan la anchura asignada y se encajan en la línea para que su visualización sea correcta.

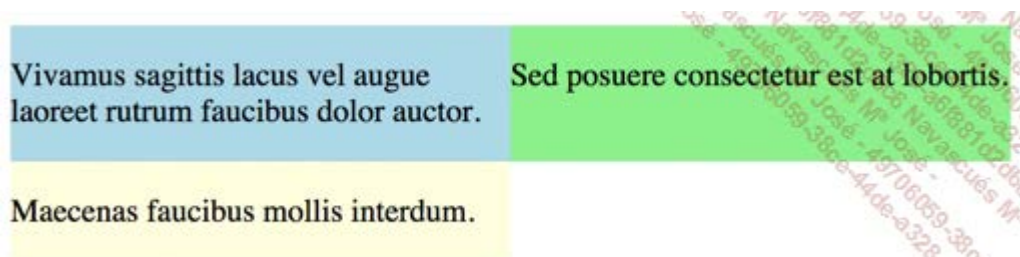
Este es el selector padre modificado:

```

#caja {
    width: 600px;
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
}

```

Esto es lo que se obtiene:



La alineación horizontal de las cajas

Podemos gestionar la alineación horizontal de las cajas hijas con la propiedad `justify-content`. Esta propiedad acepta como valor:

- `flex-start`: los elementos hijos se sitúan a la izquierda dentro del elemento padre. Es el valor por defecto.
- `flex-end`: los elementos hijos se sitúan a la derecha dentro del elemento padre.
- `center`: los elementos hijos se centran horizontalmente dentro del elemento padre.
- `space-between`: los elementos hijos se justifican dentro del elemento padre usando un espacio idéntico entre ellos.
- `space-around`: los elementos hijos se justifican dentro del elemento padre usando un espacio idéntico entre ellos y medio espacio a la izquierda del primero y a la derecha del último.

He aquí un ejemplo con elementos hijos centrados:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Cajas flexibles</title>
  <meta charset="UTF-8" />
  <style>
    #caja {
      width: 900px;
      background-color: #dedede;
      display: flex;
      flex-direction: row;
      flex-wrap: nowrap;
      justify-content: center;
    }
    .azul {
      width: 250px;
      background-color: lightblue;
    }
    .verde {
      width: 250px;
      background-color: lightgreen;
    }
    .amarillo {
      width: 250px;
      background-color: lightyellow;
    }
  </style>
</head>
<body>

  <div id="caja">

    <div class="azul">

      <p>Cras mattis consectetur...</p>

    </div>

    <div class="verde">

      <p>Nulla vitae elit libero...</p>

    </div>

    <div class="amarillo">

      <p>Curabitur blandit...</p>

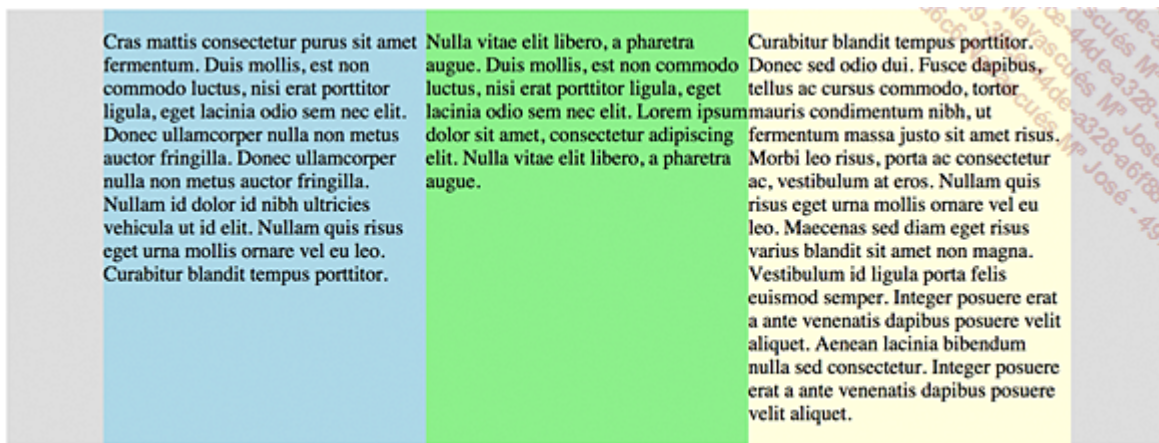
    </div>

  </div>

</body>

</html>
```

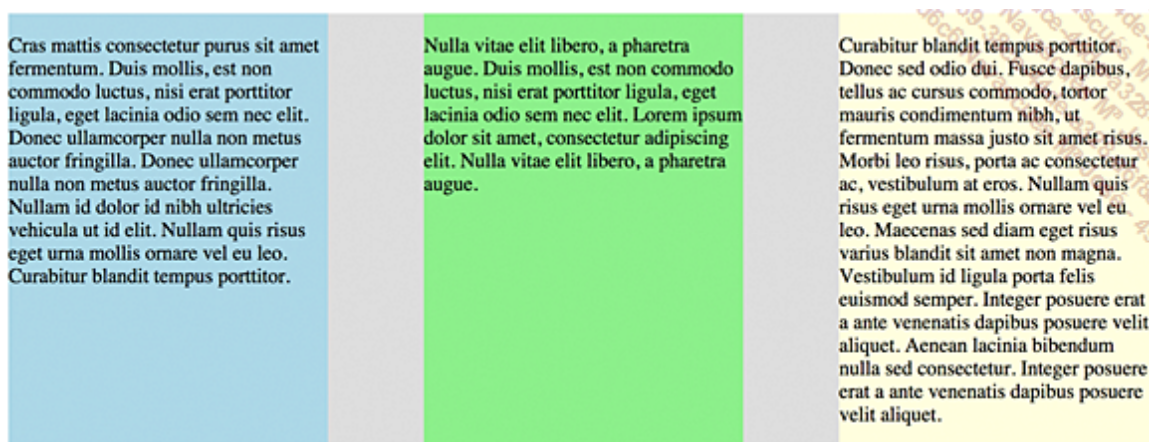
Esto es lo que se obtiene:



Para que se muestren justificadas, usaremos el valor `space-between`:

```
#caja {
  width: 900px;
  background-color: #dedede;
  display: flex;
  flex-direction: row;
  flex-wrap: nowrap;
  justify-content: space-between;
}
```

Esto es lo que se obtiene:



La alineación vertical de las cajas

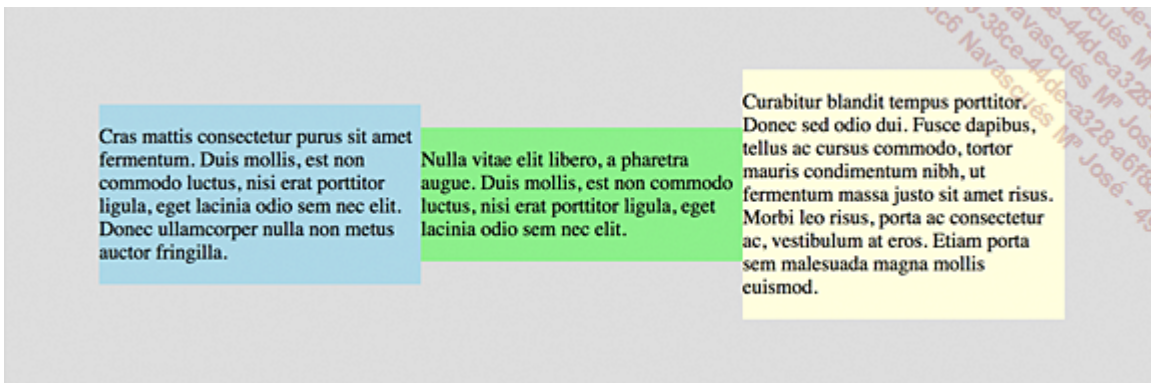
La propiedad `align-items` es la que permite gestionar la alineación vertical de los elementos hijos dentro de la caja padre. Esta propiedad acepta como valor:

- `flex-start`: los elementos hijos se colocan en la parte superior dentro del elemento padre.
- `flex-end`: los elementos hijos se colocan en la parte inferior dentro del elemento padre.
- `center`: los elementos hijos se centran verticalmente dentro del elemento padre.
- `stretch`: los elementos hijos se «estiran» dentro del elemento padre, de forma que utilizan toda la altura de este. Es el valor por defecto.
- `baseline`: los elementos hijos se alinean en la parte superior del padre, sobre cada primera línea.

Este es el selector de la caja padre para elementos hijos centrados horizontal y verticalmente:

```
#caja {
  width: 900px;
  height: 300px;
  background-color: #dedede;
  display: flex;
  flex-direction: row;
  flex-wrap: nowrap;
  justify-content: center;
  align-items: center;
}
```

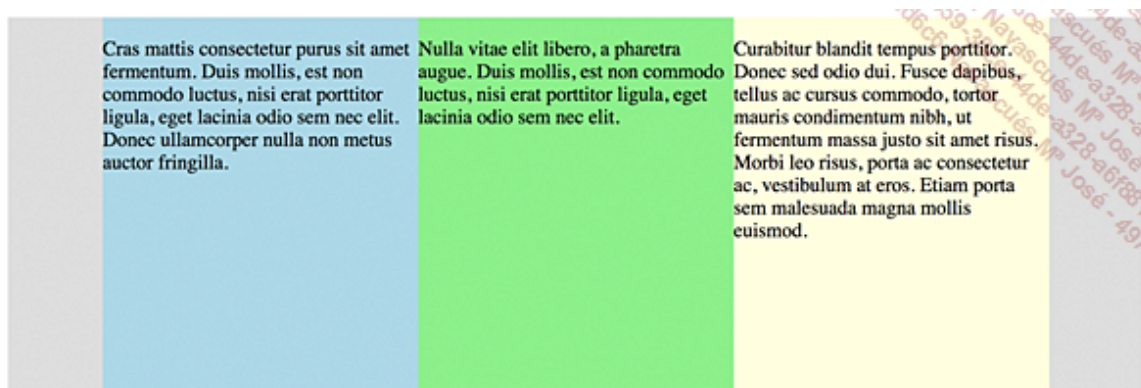
Esto es lo que se obtiene:



Y este es el selector de la caja padre para una visualización vertical justificada:

```
#caja {
  width: 900px;
  height: 300px;
  background-color: #dedede;
  display: flex;
  flex-direction: row;
  flex-wrap: nowrap;
  justify-content: center;
  align-items: stretch;
}
```

Esto es lo que se obtiene:



Las propiedades de flexibilidad

Recordemos que este módulo se denomina **Flexbox Layout**; la noción de flexibilidad está incorporada en el nombre. Esta noción de flexibilidad se aplica con la propiedad `flex`, que es la sintaxis corta de tres propiedades: `flex-grow`, `flex-shrink` y `flex-basis`.

Esta propiedad determina el comportamiento de los elementos hijos en lo que respecta a la adaptación al espacio disponible dentro del elemento padre. Estos elementos serán flexibles según el espacio disponible.

La propiedad `flex` permite definir el factor con el que se «estiran» los elementos, es decir, la facultad de ocupar el espacio disponible dentro del elemento padre. El valor 1 permite indicar que el elemento ocupa **1 parte** del total de partes disponibles, 2 indica que el elemento ocupa **2 partes**.

Veamos un ejemplo sencillo: en el contenedor (`#caja`) tenemos tres cajas que van a ocupar, respectivamente:

- 1 parte (`flex: 1;`),
- 2 partes (`flex: 2;`),
- 1 parte (`flex: 1;`).

Obtendremos, por lo tanto, $1 + 2 + 1 = 4$ partes en el contenedor.

Ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Cajas flexibles</title>
  <meta charset="UTF-8" />
  <style>
    #caja {
      width: 800px;
      display: flex;
      flex-direction: row;
      flex-wrap: nowrap;
    }
    #caja > div {
      padding: 5px;
    }
    .azul {
      background-color: lightblue;
      flex: 1;
    }
    .verde {
      background-color: lightgreen;
      flex: 2;
    }
    .amarillo {
      background-color: lightyellow;
      flex: 1;
    }
    p {
      margin: 0;
    }
  </style>
</head>
<body>
<div id="caja">
  <div class="azul">
    <p>Cras mattis consectetur...</p>
  </div>
  <div class="verde">
    <p>Nulla vitae elit libero...</p>
  </div>
  <div class="amarillo">
    <p>Curabitur blandit...</p>
  </div>
</div> </body> </html>
```

El contenedor padre mide 800 píxeles de ancho; la primera y la tercera cajas ocuparán cada una un 25 % (1/4), es decir, 200 píxeles, y la segunda, el 50 % (2/4), es decir, 400 píxeles.

Esto es lo que se obtiene:

| | | |
|--|---|---|
| Cras mattis consectetur purus sit amet fermentum. Duis mollis, est non commodo luctus, nisi erat porttitor ligula, eget lacinia odio sem nec elit. Donec ullamcorper nulla non metus auctor fringilla. | Nulla vitae elit libero, a pharetra augue. Duis mollis, est non commodo luctus, nisi erat porttitor ligula, eget lacinia odio sem nec elit. | Curabitur blandit tempus porttitor. Donec sed odio dui. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Morbi leo risus, porta ac consectetur ac, vestibulum at eros. Etiam porta sem malesuada magna mollis euismod. |
|--|---|---|

Destacar que la propiedad `column`, `float` y `clear` no tienen sentido en una caja flexible.

Composición de página en tabla

Crear una composición de página similar a la anterior, pero esta vez estructurando la página en una tabla con objeto de evitar los inconvenientes que acabamos de ver. no se trata de utilizar la «antigua» técnica desfasada de diseño de página con los elementos HTML de las tablas (`<table>`, `<tr>`, `<td>`...). Lo que vamos a utilizar es la propiedad `display`, con los valores `table`, `table-cell`...

Los valores de la propiedad `display` retoman su equivalente en la estructura HTML de las tablas. Estos son los más usuales:

- `display: table`: visualización de la tabla.
- `display: table-row`: visualización de una fila.
- `display: table-column`: visualización de una columna.
- `display: table-cell`: visualización de una celda.
- `display: table-caption`: visualización de la leyenda.
- Cuando crea una composición de página con la propiedad `display`, no es necesario crear un contenedor para las filas con el valor `table-row`. En efecto, los navegadores crean automáticamente este elemento con una caja anónima. Del mismo modo, si una «fila» de esta tabla de diseño de página no contiene más que una sola celda, no es necesario declarar la tabla; bastará con la propiedad `display: table-cell`.
- Ahora bien; en función de la composición y de su complejidad, puede resultar más sencillo declarar un `display: table` para cada fila de la composición.

La estructura de la tabla

Para la estructura de la tabla, vamos a tomar como hilo conductor el uso de la propiedad `display: table` por fila. Tendremos, por lo tanto, cuatro filas: `<header>`, `<nav>`, `<main>` y `<footer>`. Cada uno de estos elementos utilizará la clase dedicada `.tabla`.

Ejemplo

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi empresa 2</title>
  <meta charset="UTF-8" />
  <style>
    /*Estructura de las tablas*/
    .tabla {
      display: table;
      width: 960px;
      margin: 0 auto;
    }
  </style>
</head>
<body>
  <header class="tabla">
    ...
  </header>
  <nav class="tabla">
    ...
  </nav>
  <main class="tabla">
    ...
  </main>
  <footer class="tabla">
    ...
  </footer>
</body>
</html>
```

Las celdas de la tabla

Cada contenido se ubicará en una celda, y cada celda usará la clase dedicada `.celda`, con `display: table-cell;`.

Para gestionar de forma ordenada la alineación vertical en las celdas, crearemos la clase `.align-vert`, con `vertical-align: middle;`.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi empresa 2</title>
  <meta charset="UTF-8" />
  <style>
    /*Estructura de las tablas */
```

```

        .tabla {
display: table;

        width: 960px;

        margin: 0 auto;

    }

    .celda {

        display: table-cell;

    }

    .align-vert {

        vertical-align: middle;

        padding: 0 0 0 10px;

    }

</style>
</head>
<body>

    <header class="tabla">

        <div id="logo" class="celda align-vert">

            ...

        </div>

        <div id="busqueda" class="celda align-vert">

            ...

        </div>

    </header>

    <nav class="tabla">

        <ul id="menu" class="celda align-vert">

            ...

        </ul>

    </nav>

    <main class="tabla">

        <div id="content" class="celda">

            ...

        </div>

        <aside class="celda">

            ...

        </aside>

    </main>

    <footer class="tabla">

        <p class="celda align-vert">Sitio creado...</p>

    </footer>

</body>
</html>

```

Los contenidos de la tabla

Aquí retomamos los contenidos anteriores, con clases dedicadas al formato específico de los contenidos.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Mi empresa 2</title>
  <meta charset="UTF-8" />
  <style>
    /* Estructura de las tablas */
    .tabla {
      display: table;
      width: 960px;
      margin: 0 auto;
    }
    .celda {
      display: table-cell;
    }
    .align-vert {
      vertical-align: middle;
      padding: 0 0 0 10px;
    }
    /* El encabezado */
    header {
      height: 150px;
    }
    #logo, #busqueda {
      width: 480px;
      padding: 0 20px;
      background-color: #000;
      color: #fff;
    }
    #busqueda {
      text-align: right;
    }
    #logo p, #logo h1 {
      margin: 0;
    }
    #logo h1 {
      font-family: 'Oswald', sans-serif;
      font-size: 2em;
    }
    /* La barra de navegación */
    nav {
```

```

        background-color: lightblue;
        height: 30px;
    }
    #menu li {
        display: inline-block;
        margin-right: 20px;
    }
    #menu a {
        text-decoration: none;
    }
    /* La parte central */
    #content {
        width: 640px;
        padding: 20px;
        background-color: lightyellow;
    }
    aside {
        width: 320px;
        padding: 10px;
        background-color: lightgreen;
    }
    /* El pie de página */
    footer {
        height: 30px;
        background-color: gray;
        font-size: small;
        color: #fff;
    }
</style>
</head>
</head>
<body>
    <header class="tabla">
        <div id="logo" class="celda align-vert">
            <p></p>
            <h1>Planeta azul</h1>
        </div>
        <div id="busqueda" class="celda align-vert">
            <form method="post" action="#">
                <input type="text" name="search"
placeholder="buscar" value="" />
            </form>
        </div>
    </header>
    <nav class="tabla">

```

```

        <ul id="menu" class="celda align-vert">
            <li><a href="#">Inicio</a></li>
            <li><a href="#">Proyectos</a></li>
            <li><a href="#">Acciones</a></li>
            <li><a href="#">Reportajes</a></li>
        </ul>
    </nav>
<main class="tabla">
    <div id="content" class="celda">
        <h2>Venenatis Etiam Inceptos</h2>
        <p>Sed posuere consectetur...</p>
        <h2>Ipsum Sollicitudin Ornare</h2>
        <p>Duis mollis, est non commodo..</p>
    </div>
    <aside class="celda">
        <h3>Magna Fermentum Condimentum</h3>
        <p>Curabitur blandit tempus...</p>
        <h3>Lorem Parturient</h3>
        <p>Nulla vitae elit libero...</p>
    </aside>
</main>
<footer class="tabla">
    <p class="celda align-vert">Sitio creado...</p>
</footer>
</body>
</html>

```

Visualización y ventajas

Con esta técnica eliminamos todos los inconvenientes que presentaba la técnica anterior:

- La alineación vertical requiere simplemente el uso de la propiedad `vertical-align: middle;` en una sola clase.
- Las celdas de una fila tienen siempre la misma altura.
- No son necesarias imágenes de «falsas columnas» para obtener fondos de color.
- Menos líneas de código.