Bootcamp Full Stack

CSS avanzado

Transformaciones

Transformaciones 2D

Transform

Las transformaciones, permiten rotar, escalar y trasladar los elementos del **HTML, para luego hacer un buen complemento con animaciones y transiciones de CSS3**.

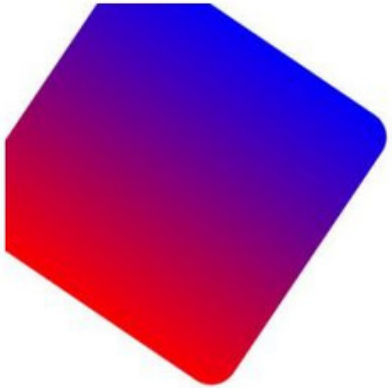
A la derecha, veamos un ejemplo **de sintaxis para comprender mejor las transformaciones**.

```
<style>
div {
   transform: tipoTransformacion(parametro);
    }


                                    </style>
```

Rotate

Rotamos el objeto que puede ser de línea o de bloque, tanto a favor como en el sentido contrario a las agujas del reloj. Los valores posibles son **0 a 360deg o en el sentido contrario: 0 a -360deg**



Resultado

```
<style>
div {
   width: 200px;
   height: 200px;
   border-radius: 20px;
   padding: 20px;
   background: linear-gradient(blue,red);
   transform: rotate(34deg);
    }

                              </style>



    <div>
     mi contenedor

       </div>
```

Transform-origin

Se pueden utilizar **medidas de longitud, px, em, %** (en referencia al alto o al ancho del elemento que estoy transformando), **así como palabras (center, top, bottom, etc)**, es decir, aquellos valores que por ejemplo **utilizamos en la `propiedad background-position`.**

```
<style>
div {
   width: 200px;
   height: 200px;
   border-radius: 20px;
   padding: 20px;
   background: linear-gradient(blue,red);
   transform: rotate(34deg);
   transform-origin: 20px top;
    }

</style>
```

Transformaciones 3D

Rotación 3D

La rotación 3D va desde **0 a 180deg y desde 0 a -180deg**, y se puede realizar **con *x, y* ó *z:***

```
<style>
div {
    width: 200px;
    height: 200px;
    border-radius: 20px;
    padding: 20px;
    background: linear-gradient(blue,red);
    transform: rotateY(-34deg);

    }

</style>
```

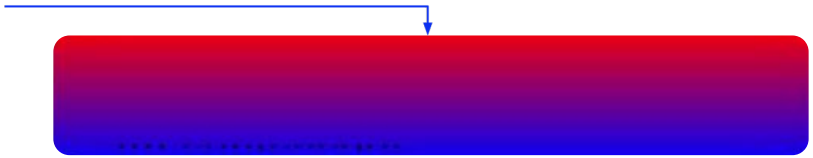
Esta es otra forma:

```
<style>
div {
    width: 200px;
    height: 200px;
    border-radius: 20px;
    padding: 20px;
    background: linear-gradient(blue,red);
    transform: rotateX(100deg);

     }

                                </style>
```

Resultado



¿Cómo empezar a trabajar de forma más eficiente?

Podemos favorecer la cantidad de líneas dentro de nuestro CSS utilizando **un shorthand o declaración de una única línea:**

```
div {
    width: 200px;
    height: 200px;
    border-radius: 20px;
    padding: 20px;
    background: linear-gradient(blue,red);
    transform: rotateY(160deg) rotateX(-34deg);


    }
```

¿Cómo empezar a trabajar con valores negativos?

También podemos trabajar con valores negativos como en el ejemplo anterior. **Por supuesto también podemos sumar una rotación en 2D, veamos un ejemplo.**

Resultado

```
<style>
div {
    width: 200px;
    height: 200px;
    border-radius: 20px;
    padding: 20px;
    background: linear-gradient(blue,red);
    transform: rotate(34deg) rotateY(-30deg);

    }

</style>
```

Perspective

La propiedad perspective se aplica al padre del elemento que contiene la rotación en 3D.

Por esa razón debemos seguir la siguiente estructura en nuestro ***archivo.html***

```
<padre>
<hijo> </hijo> <!--elemento es el que va a tener la transformación en 3D-->
</padre> <!--elemento que va a tener la perspectiva-->
```

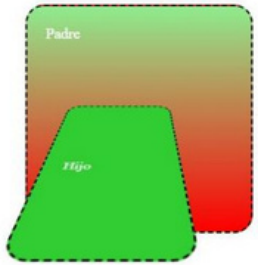
Ahora trabajamos en nuestros ***estilos.css:***

```
<style>
div {
   width: 200px;
   height: 200px;
   border-radius: 20px;
   padding: 20px;
   background: linear-gradient(lightgreen,red);
   color: white;
   border: 2px dashed black;    }
#padre {   margin-left: 100px; perspective: 200px;
   perspective-origin: center bottom;}
 #hijo { background: limegreen; width: 100px;  transform: rotateX(34deg); line-height: 230px;}

                              </style>


<div id="padre">Padre  <div id="hijo">  Hijo </div> </div>
```

Resultado



Más transformaciones

Scale

Implementación

Me permite **escalar un elemento haciéndolo más pequeño o más grande**. Veamos un ejemplo:

```
<style>
div {
    width: 200px;
    height: 200px;
    border-radius: 20px;
    padding: 20px;
    background: linear-gradient(lightgreen,red);
    color: white;
    border: 2px dashed black;
    transform: scale(2);    }


                                </style>



<div>contenido</div>
```

Las **alternativas para trabajar** varían. Estas son todas las posibles:

```
transform: scale(width,height)
/*escalamos tanto el ancho como el alto*/
transform: scaleX(n)
/*ancho del elemento*/
transform: scaleY(n)
/*alto del elemento*/
```

Ahora veamos un ejemplo aplicado en el archivo ***estilos.css***:

```
<style>
div {
    width: 200px;
    height: 200px;
    border-radius: 20px;
    padding: 20px;
    background: linear-gradient(lightgreen,red);
    color: white;
    border: 2px dashed black;
    transform: scale(0.5, 2);    }


                                </style>



<div>contenido</div>
```

Translate

La diferencia entre trasladar un elemento en una animación, en una transición con **translate**, a hacerlo **con position, es que es mucho más fluido el movimiento**.

Te permite **trasladar un elemento a través de las transformaciones**. Las posibles variantes son:

```
transform: translate(x,y);
transform: translateX();
transform: translateY();
```

Ejemplo

```
<!DOCTYPE html>
<head>
<style>
div {
   width: 200px;
   height: 200px;
   border-radius: 20px;
   padding: 20px;
   background: linear-gradient(lightgreen,red);
   color: white;
   border: 2px dashed black;
   transform: translate(100px, 200px);    }

                                    </style>
```

Skew

Esto sería lo que, por ejemplo, en *Photoshop* o *Illustrator* llamamos ***sesgar.***

Las posibles variantes son:

```
transform: skew(x,y)
transform: skewX()
transform: skewY()
```

```
<style>
div {
    width: 200px;
    height: 200px;
    border-radius: 20px;
    padding: 20px;
    background: linear-gradient(lightgreen,red);
    color: white;
    border: 2px dashed black;
    transform: skewX(20deg)    }

                                </style>

<div> transformaciones </div>
```

No tiene sentido trabajar ni con 0 ni con 180deg porque ese sería el valor predeterminado, el elemento no tiene cambios es decir no está sesgado, **la idea es siempre trabajar con valores entre el rango antes referenciado**.

En todos los casos anteriores, es decir, en el **skew y en el translate podemos hacerlo también sobre el eje Z, por ejemplo skewZ() o `translateZ()`.**