

# CSS AVANZADO

LIMA UD2 – Tema 1

IES Plurilingüe Antón Losada Diéguez

## Tabla de contenido

1. Introducción.....	3
2. Inclusión por CDN .....	3
2.1. Descarga .....	3
2.2. CDN.....	3
3. Unidades avanzadas.....	3
3.1. ems .....	4
3.2. rems.....	4
4. !Important .....	4
5. Cursor .....	5
6. Selectores de descendencia avanzados .....	5
6.1. Hijo directo .....	5
6.2. Hermano .....	6
7. Pseudoelementos .....	7
7.1. Antes y después .....	7
7.2. Texto.....	8
7.3. Viñeta .....	8
7.4. Selección .....	8
8. Flexbox .....	9
9. Variables.....	9
9.1. Valor de respaldo .....	10
10. Funciones matemáticas.....	10
10.1. Min/max .....	10
10.2. Calc .....	11
11. Contador.....	11
11.1. Inicializar/reiniciar .....	11
11.2. Aumentar el contador .....	11
11.3. Uso.....	11
12. Iconos .....	12
12.1. Image Sprite .....	13
12.1.1. Imagen transparente + fondo .....	13
12.1.2. Contenedor + imagen relativa.....	14
12.1.3. Uso desde servidor .....	14
12.2. Iconos SVG .....	15

12.3. Uso combinado .....	15
13. Degradados.....	15
13.1. Lineal.....	16
13.2. Radial .....	16
13.3. Cónico .....	16
14. Transformaciones.....	17
14.1. Traslación .....	17
14.2. Rotación .....	17
14.3. Inclinação.....	18
14.4. Escalado .....	18
14.5. Matrix .....	18
14.6. Transformaciones 3D.....	18
15. Transiciones y animaciones .....	18

# CSS Avanzado

---

## 1. Introducción

Existen infinidad de propiedades de CSS, por lo que se irán dividiendo en documentos en función de su nivel de conocimientos necesarios, utilidad o complejidad.

En este tema se verán los avanzados. Algunos forman parte de los anteriores o los complementan, mientras que otros son totalmente nuevos.

## 2. Inclusión por CDN

A la hora de incluir el archivo CSS de una fuente externa, existen dos formas fundamentales de incluirlo, mediante la descarga e inclusión o mediante CDN (Content Delivery Network).

### 2.1. Descarga

En este caso, su inclusión es igual que con un archivo CSS propio, se descarga en el servidor y se enlaza.

Esto evita que el usuario tenga que realizar una consulta externa y permite pruebas en local sin conexión a internet, pero sobrecarga el servidor con peticiones, ya que tendrá que enviarle el archivo CSS a todo cliente que se conecte.

### 2.2. CDN

En el segundo caso, se referencia a un repositorio externo, de tal forma que el cliente descarga el CSS de dicho repositorio.

```
<link rel="stylesheet" href="https://.../file.css">
```

Esto hace que el servidor no se colapse con peticiones del archivo CSS y hace que el cliente guarde en caché el CSS y pueda ser utilizado múltiples veces por páginas distintas.

Además, los CDN pueden estar formados por una red de servidores, de tal forma que la petición sea respondida por el más cercano al usuario, reduciendo el tiempo de respuesta de la página.

## 3. Unidades avanzadas

Además de las unidades absolutas px y la relativa porcentual %, existen otras dos unidades relativas, los ems y los rems.

Ambas toman como referencia el tamaño de fuente, pero con distintos ámbitos.

Son ampliamente utilizadas porque permiten un mayor control de la proporcionalidad de la página, pues el texto suele ser la parte fundamental de una web.

### 3.1. ems

Los *ems* son la unidad básica en tipografía, la habitual en editores de texto, impresión, etc.

En CSS, se usa para escalar tamaños con respecto al tamaño de fuente del elemento o del ancestro.

```
div {
    font-size: 30px;
}
span {
    font-size: 0.5em;
}

<div>
    El tamaño de fuente es de 30px
    <span>El span tiene un tamaño de fuente de 0.5em, por tanto 0.5x30 = 15px</span>
    .
</div>
```

El tamaño de fuente es de 30px El span tiene un  
tamaño de fuente de 0.5em, por tanto 0.5x30 = 15px.

En este ejemplo, el *div* tiene un tamaño de fuente 30px, por lo que el *span* hereda dicho tamaño. Con *em* se referencia a esos 30px, por lo que estableciendo su tamaño en 0.5em, se aplicará 15px.

Hay que tener en cuenta que los *ems* referencian al tamaño que le afecte a él directamente. Si un elemento dentro del *span* del ejemplo utiliza *ems*, referenciará a los 15px.

Los *ems* pueden usarse de cualquier forma que el resto de unidades, no solo para los tamaños de fuente.

### 3.2. rems

Los *rems* son muy similares a los *ems*, pero hacen referencia al tamaño de fuente de la página y no del elemento. Más concretamente, hace referencia al tamaño de fuente del elemento *html*.

Por defecto, el elemento *html* tiene 16px de tamaño de fuente, pero puede modificarse como cualquier otro elemento.

## 4. !Important

En el caso de que se necesite hacer que una propiedad se aplique por encima de las demás, independiente de la especificidad de elementos, se estipula poniendo *!important* tras el valor de la propiedad.

```
#myid {
  background-color: blue;
}
.priebac {
  background-color: gray;
}
p {
  background-color: red !important;
}

<p id="myid">Texto</p>
```

Aunque por especificidad, debería aparecer en azul, dado que el rojo está estipulado como *!important*, tiene prioridad sobre los demás.

El problema de usar *!important* es que ignora toda la cascada de prioridades, por lo que puede llegar a romper la estructura de la página.

La única forma de superar un *!important* es con otro, por lo que su uso puede conllevar su uso en reiteradas ocasiones.

Por todo esto, **es una mala práctica y su utilización ha de quedar reservada para casos muy concretos en los que sea imposible establecer la propiedad de otra forma.**

## 5. Cursor

Con CSS también es posible cambiar el puntero al ponerlo encima del elemento. Para ello, se usa la propiedad `cursor` y uno de los múltiples valores disponibles.

```
.pointer {
  cursor: pointer;
}
```

En este caso, al poner el ratón sobre el elemento, aparecerá la mano de hacer clic.

Esta propiedad es muy utilizada para señalar al usuario la funcionalidad de algún elemento recreado, como botones, menús etc.

## 6. Selectores de descendencia avanzados

En el primer tema se vio el selector de descendencia simple (espacio) que implicaba a todos los hijos directos y todos los descendientes de estos, pero existen selectores más avanzados.

### 6.1. Hijo directo

Este selector selecciona solo los hijos directos de un elemento, sin seguir la cascada.

```
div > p {
  background-color: yellow;
}
```

<code>&lt;div&gt;</code>	<b>Hijo_1 h2</b>
<code>  &lt;h2&gt;Hijo_1 h2&lt;/h2&gt;</code>	
<code>  &lt;p&gt;Párrafo hijo_2&lt;/p&gt;</code>	<b>Párrafo hijo_2</b>
<code>&lt;/div&gt;</code>	
<code>&lt;div&gt;</code>	
<code>  &lt;span&gt;&lt;p&gt;Párrafo nieto&lt;/p&gt;&lt;/span&gt;</code>	Párrafo nieto
<code>&lt;/div&gt;</code>	
<code>&lt;p&gt;Párrafo suelto&lt;/p&gt;</code>	Párrafo suelto

En este caso, la propiedad se aplica únicamente a aquel hijo directo de un *div* y no al hijo del *span*, aunque sea descendiente del elemento.

## 6.2. Hermano

Existen dos selectores para aplicar reglas a los elementos hermanos de otro elemento. Dos elementos son hermanos si descienden de un mismo elemento y están al mismo nivel.

El primero de los selectores es la virguitilla (~) y permite seleccionar a todos los hermanos de un elemento.

```
div ~ p {
  background-color: yellow;
}
```

### Hijo 1 de div

<code>&lt;div&gt;</code>	
<code>  &lt;h2&gt;Hijo 1 de div&lt;/h2&gt;</code>	Hijo 2 de div
<code>  &lt;p&gt;Hijo 2 de div&lt;/p&gt;</code>	
<code>&lt;/div&gt;</code>	<b>Primer hermano</b>
<code>&lt;p&gt;Primer hermano&lt;/p&gt;</code>	
<code>&lt;p&gt;Segundo hermano&lt;/p&gt;</code>	<b>Segundo hermano</b>

Por otro lado, el más (+) selecciona únicamente al hermano adyacente.

```
div + p {
  background-color: yellow;
}
```

### Hijo 1 de div

Hijo 2 de div

**Primer hermano**

Segundo hermano

En este caso, solo seleccionará al elemento si está inmediatamente después, si hay otro elemento entre ellos, aunque no cumpla la condición, no lo seleccionará.

## Hijo 1 de div

<pre>&lt;div&gt;   &lt;h2&gt;Hijo 1 de div&lt;/h2&gt;   &lt;p&gt;Hijo 2 de div&lt;/p&gt; &lt;/div&gt; &lt;span&gt;En el medio&lt;/span&gt; &lt;p&gt;Primer hermano&lt;/p&gt; &lt;p&gt;Segundo hermano&lt;/p&gt;</pre>	<p>Hijo 2 de div</p> <p>En el medio</p> <p>Primer hermano</p> <p>Segundo hermano</p>
---	--

## 7. Pseudoelementos

Los pseudoelementos se añaden a los selectores de una forma similar a las pseudoclases, pero permiten añadir estilos a una parte concreta del elemento y no en función del estado como estos.

Los pseudoelementos se escriben con dos veces dos puntos (::) antes del nombre del mismo.

Al igual que con las pseudoclases, los pseudoelementos pueden combinarse, aplicarse varios a un mismo elemento, pero han de estipularse como reglas independientes.


### 7.1. Antes y después

Estos dos permiten añadir contenido antes y después del elemento.

Ambos tienen el mismo funcionamiento, añaden contenido antes (::before) o después (::after) como texto o una imagen. En cualquier caso, se le añade con la propiedad *content*.

<pre>h1::before {   content: "~="; } h1::after {   content: "=~"; }</pre>	<p>~=<b>Cabecera</b>=~</p>
---	----------------------------

En este caso se añade texto antes y después del elemento.

<pre>h1::before {   content: url(smiley.gif); }</pre>	<p> <b>Cabecera</b></p>
---	--

En este otro, mediante el uso de url(), se añade una imagen antes del elemento.

<pre>h1::before {   content: "~=";   color: red; }</pre>	<p>~=<b>Cabecera</b>=~</p>
--	----------------------------



Como se puede observar en este otro ejemplo, a estos añadidos se le pueden aplicar formatos propios.

En ambos casos, lo añadido forma parte del bloque de contenido, teniendo el *padding* y el borde añadido a continuación. Sin embargo, el usuario no puede seleccionar ese extra, pues no es considerado parte del texto real, simplemente un añadido.

## 7.2. Texto

Existen dos pseudoelementos propios del texto para modificar partes concretas.

El primero, *::first-letter* permite modificar la primera letra de un elemento.

```
p::first-letter {  
  color: red;  
  font-size: 3em;  
}
```

**S**e puede usar el pseudoelemento *::first-letter* para modificar el texto

El segundo, *::first-line* permite modificar la primera línea de texto de un elemento. Esta se adapta dinámicamente al tamaño del elemento, incluso si se modifica dinámicamente.

```
p::first-line {  
  color: red;  
  font-family: sans-serif;  
}
```

You can use the *::first-line* pseudo-element to add a special effect to the first line of a text. Some more text. And even more, and more, and more, and more, and more, and more, and more, and more, and more, and more, and more, and more.

## 7.3. Viñeta

El pseudoelemento *::marker* permite modificar la viñeta de las listas aplicándoles un formato propio.

```
li::marker {  
  color: tomato;  
  font-size: 23px;  
}
```

1. Primero
2. Segundo
3. Tercero

Está en estado experimental y no está soportado en todos los navegadores, teniendo funcionalidades limitadas en otros.

## 7.4. Selección

El pseudoelemento *::selection* permite modificar el aspecto de los elementos que el usuario haya seleccionado con el cursor.

```
::selection {  
  color: red;  
  background: yellow;  
}
```

Párrafo  
Texto en un div

## 8. Flexbox

Flexbox es un modelo de caja flexible para el posicionamiento vertical y horizontal de los elementos. Es una alternativa a float y actualmente uno de los modelos más usados para la estructuración de la página.

## 9. Variables

Las variables son una utilidad propia de los lenguajes de alto nivel que tomó CSS en las últimas versiones.

Estas ahorran mucho tiempo, permiten una rápida modificación y alto nivel de reutilización, ya que guardan valores que se podrán usar posteriormente.

Las variables se declaran como una propiedad, pero con un nombre personalizado y anteponiéndole dos guiones (--).

El valor que toman puede ser una dirección con *url*, un texto a usar como valores, o un valor propio como un color *rgb*.

Luego, puede ser utilizada mediante la función *var(nombre)*.

```
:root {
  --back-img: url('../img/bg.png');
  --gold-color: 240,230,140;
  --black-color: 50,50,50;
  --text-color: rgb(var(--black-color));
  --back-color: rgba(var(--gold-color),0.1);
}
body {
  background: var(--back-image) repeat;
  color: var(--text-color);
}
```

En este ejemplo puede verse su uso y combinación de unas con otras.

Existe una pseudoclase *:root* utilizada casi en exclusiva para las variables que hace referencia a la raíz del documento, es decir, la etiqueta *html*. Las variables declaradas en esta pseudoclase, afectarán a todo elemento, por tanto se denominarían globales.

Las variables pueden sobrescribirse o declararse en cualquier nivel, afectando a ese elemento y descendientes.

Si una variable se sobrescribe en un elemento, todas las reglas que le afecten y que conlleven el uso de esa variable, usarán el valor de ese elemento, aunque la regla sea heredada de un nivel superior.

```
body {
  background: var(--back-image) repeat;
  color: var(--text-color);
}
div {
  --text-color: wheat;
}
```

En este ejemplo, todos los *divs* tendrán el color establecido como *wheat*, ya que tienen la variable que lo estipula establecida a dicho valor, independientemente de que la regla que establece el color venga heredada del *body*.

Estas variables también pueden ser estipuladas en la etiqueta *style* del elemento, por ejemplo.

Al igual que cualquier otra propiedad, en caso de solapamiento, se seguirá la cascada de reglas de CSS para determinar el valor real.

Esto permite un nivel de personalización extremadamente potente, sobre todo si estos valores se modifican dinámicamente con JavaScript (por ejemplo) o se generan dinámicamente del lado del servidor.

### 9.1. Valor de respaldo

La función *var* dispone de un segundo parámetro para establecer un valor por defecto en caso de que la variable no tome ningún valor.

```
color: var(--text-color, red);
```

Si la variable no tiene valor y no tiene valor por defecto, la propiedad será ignorada.

Hay que tener en cuenta que el valor por defecto también ha de tener las unidades acordes al valor necesario.

## 10. Funciones matemáticas

CSS dispone de varias funciones matemáticas que permiten realizar pequeños cálculos.

De por sí, son muy potentes, pero unidas a las variables, abren la puerta a una personalización y adaptación mucho mayor.

### 10.1. Min/max

Estas dos funciones permiten seleccionar el mínimo o máximo valor de una serie de valores.

```
label {
  width: min(40%, 400px);
}
```

En este ejemplo, el ancho de los campos del formulario será el 40% del contenedor o 400px, lo que sea menor.

```
label {
  width: max(var(--elem-w), 400px);
}
```

En este caso, unido al poder de las variables, el ancho será el mayor entre 400px o el de la variable en cuestión.

## 10.2. Calc

La función *calc* permite realizar un cálculo matemático.

```
#div1 {  
    width: calc(100% - 100px);  
}
```

En este ejemplo se ve uno de los usos más habituales de la función *calc*, el aumento o disminución de un valor relativo por uno absoluto.

## 11. Contador

Los contadores son un tipo de “variable” que permite contar el número de veces que se cumple una regla.

Su uso más común es el de incluir el número de sección de forma automática en un título o generar de forma manual una lista de elementos numerados.

### 11.1. Inicializar/reiniciar

Para crear el contador, se usa la propiedad *counter-reset*, cuyo valor es el nombre del contador.

Esta propiedad también reinicia el contador si se necesita usar de nuevo.

```
div {  
    counter-reset: section;  
}
```

En este ejemplo, el contador *section* se reiniciará cada vez que encuentre un elemento *div*.

### 11.2. Aumentar el contador

Una vez el contador está creado, se incrementa con la propiedad *counter-increment*, cuyo valor es el contador a incrementar.

```
h2 {  
    counter-increment: section;  
}
```

En este ejemplo, cada vez que encuentra un *h2*, se incrementará en 1 el valor del contador *section*.

### 11.3. Uso

Una vez creado el contador y aumentado su valor, se puede recoger y usar dicho valor con *counter(nombre)*, siendo nombre el previamente establecido.

```

<div>
  <h2>HTML</h2>
  <h2>CSS</h2>
</div>
<div>
  <h2>JavaScript</h2>
  <h2>SQL</h2>
</div>
div {
  counter-reset: section;
}
h2::before {
  counter-increment: section;
  content: "Sección " counter(section) ": ";
}

```

## Sección 1: HTML

## Sección 2: CSS

## Sección 1: JavaScript

## Sección 2: SQL

En este ejemplo, se crea/reinicia el contador en cada *div* y se incrementa en cada *h2*, añadiendo el valor en el *::before* de dicho elemento.

Se pueden combinar tantos contadores como sean necesarios, siempre que tengan un nombre distinto.

```

body {
  counter-reset: section;
}
div {
  counter-increment: section;
  counter-reset: sub_section;
}
h2::before {
  counter-increment: sub_section;
  content: counter(section) "."
  counter(sub_section) ". ";
}

```

### 1.1. HTML

### 1.2. CSS

### 2.1. JavaScript

### 2.2. SQL

## 12. Iconos

Los iconos se utilizan para mejorar la experiencia del usuario, ya sea en botones o menús para hacerlos más representativos, representar algo o como decoración de textos.

Aunque se podría usar una imagen por cada icono, esto acarrea el problema de que, cada vez que un icono apareciese en la página, el cliente tendría que solicitar y descargar la imagen correspondiente, consumiendo recursos del servidor y del cliente.

A día de hoy hay dos alternativas principales, el uso de una *image sprite* y el uso de iconos *svg*.

## 12.1. Image Sprite

Una *image sprite* es una única imagen con todos los iconos de la página en forma de tabla. De esta forma, solo se solicitará y descargará la imagen una única vez y se usará para todos los iconos de la página.

La imagen puede ser vertical u horizontal, tener una única fila/columna o ser una tabla. Es recomendable, eso sí, que todos los iconos tengan el mismo tamaño reservado en la imagen.



Para este ejemplo, se usarán cuatro iconos de 16x16 px cada uno agrupados de forma vertical.

Existen dos formas fundamentales de utilizarla, mediante una imagen transparente y la *image sprite* como fondo desplazada o mediante el uso de un contenedor y una imagen con posición relativa.

### 12.1.1. Imagen transparente + fondo

En este método, se utiliza cada uno de los iconos mediante la propiedad *background*, indicando la *url* de la imagen y los valores de desfase en x e y y el tamaño de cada icono.

```
  

```

Las imágenes representan elementos distintos, pero comparten *src*, que sería una imagen transparente para poder ver su fondo.

```
.icon {  
    width: 16px;  
    height: 16px;  
}  
#primavera {  
    background: url(estaciones.png) 0px 0px;  
}  
#otono {  
    background: url(estaciones.png) 0px -32px;  
}
```

Ambos iconos son del mismo tamaño, 16x16 px y comparten la imagen, pero cada uno tiene un desplazamiento dependiendo de la posición en la que se encuentre en el sprite.

La primavera no tiene desplazamiento, ya que se encuentra en la posición 0, 0. El otoño, se encuentra en la tercera posición, por tanto habrá que desplazarse dos iconos hacia abajo,  $16 \times 2 = 32$ .

Hay que tener en cuenta que el desplazamiento hacia abajo y hacia la derecha es negativo y el pixel superior izquierdo de toda imagen es el 0x0.

El anterior ejemplo usa la versión agrupada de la propiedad *background*, pero también se puede usar por separado, con *background-image* y *background-position*.

```
.icon {
    width: 16px;
    height: 16px;
    background-image: url(estaciones.png);
}
#primavera {
    background-position: 0px 0px;
}
#otono {
    background-position: 0px -32px;
}
```

En este ejemplo se estipula la imagen común en la clase de los iconos y en los ids, la posición.

### 12.1.2. Contenedor + imagen relativa

En esta opción se usa un elemento contenedor, en este ejemplo un *span*, que tendrá el tamaño del icono y cortará la imagen, que se desplaza de forma relativa.

```
<span class="icon">
    
</span>

.icon {
    width: 16px;
    height: 16px;
    overflow: hidden;
    display: inline-block;
}
#otono {
    position: relative;
    top: -32px;
}
```

En este caso, el contenedor tiene que ser *inline-block* para permitir tamaños, tener el mismo tamaño que los iconos y que su contenido se oculte si sale del tamaño.

Por otro lado, la imagen tiene una posición relativa y su *top/left* (dependiendo de si es vertical u horizontal) será el desplazamiento.

### 12.1.3. Uso desde servidor

Aunque esto pueda parecer engorroso, su uso se simplifica y facilita mediante su inclusión desde el lado del servidor, ya que permite crear una pequeña función que calcula el valor en base al índice.

```

/*
Función iconTileSelector
  Selecciona el icono en un sprite/tile vertical de 16px en base a la posición que ocupa

Parameters:
  $img: La ruta de la imagen
  $pos: La posición
*/
if(!function_exists('iconTileSelector'))
{
    function iconTileSelector($img, $pos)
    {
        $pos = 16 * $pos * -1;
        $icon = '<span class="tileicon">';
        $icon .= '';
        $icon .= '</span>';

        return $icon;
    }
}

```

Este ejemplo representa una posible función en PHP para la inclusión de iconos mediante el segundo método.

## 12.2. Iconos SVG

El uso de *image sprite*, ahorra en recursos y tiempo pero, aun así, el cliente seguirá necesitando descargar la imagen. Además, el uso de imágenes, impide la adaptación de las mismas al color de texto o escalar su tamaño.

Por todo esto, la opción más viable es el uso de SVG, el cual puede escalarse, incluso con el propio tamaño de la fuente, darle formato con CSS y el peso es mucho menor que el de una imagen.

Hay que tener en cuenta que los iconos tienen una serie de limitaciones:

- Tienen un color uniforme, habitualmente el del propio texto.
- Su creación es más compleja, por lo que se suelen utilizar librerías y no siempre tienen los iconos que se puedan necesitar.
- Se pueden seleccionar como texto y no se pueden guardar ni tratar como una imagen.

## 12.3. Uso combinado

Lo más habitual es el uso combinado de iconos *svg* e *image sprites*, usando los primeros para textos, dar un aspecto más visual y representativo a los botones, incluso sustituyendo todo el texto del mismo, y los segundos para cosas más personalizadas, como representaciones de algo en concreto o menús propios.

También es habitual el uso de varios *image sprite* para agrupar iconos por tipo o sección.

## 13. Degradados

Hasta ahora se han visto colores planos, un único color para una propiedad, pero es posible establecer degradados de colores para los fondos de los elementos.

Aunque pueda resultar llamativo, hay que tener en cuenta la elección de color y el uso que se haga de ellos, pues puede llegar a ser molesto.

Aunque sea un color, para ponerlo se estipula en la propiedad *background-image*.



### 13.1. Lineal

El más sencillo es el degradado lineal, que va de un lado a otro. Se estipula con *linear-gradient*.



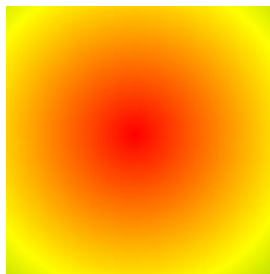
```
background-image: linear-gradient(to bottom right, red , yellow);
```

El primer parámetro es la dirección, estipulada de forma textual o mediante grados 0-360°.

A continuación, los colores separados por comas. Aunque en el ejemplo son dos, se pueden incluir un número indeterminado, incluso repetirse.

### 13.2. Radial

El segundo tipo son los circulares, empezando en el centro hacia fuera. Se estipula con *radial-gradient*.



```
background-image: radial-gradient(red, yellow, green);
```

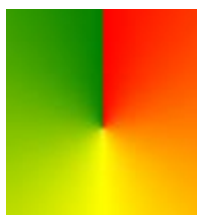
Dado que van desde el centro, solo requiere los colores a utilizar.

```
background-image: radial-gradient(circle, red, yellow, green);
```

Por defecto, la forma del degradado es elíptico, adaptándose a la forma del contenedor. Para hacer que siempre sea circular, se incluye *circle* como primer parámetro.

### 13.3. Cónico

El tercer tipo son los cónicos, empezando en una línea y degradando de forma circular. Se estipula con *conic-gradient*.



```
background-image: conic-gradient(from 90deg, red, yellow, green);
```

En este degradado, todo se indica en grados, incluyendo el punto inicial. Si no se estipula, el valor es 0deg.

En los colores también se puede estipular el inicio y el final del color.

```
conic-gradient(red 0deg 90deg,  
              yellow 90deg 180deg,  
              green 180deg 270deg,  
              blue 270deg);
```

En este ejemplo, se estipulan cuatro colores y el punto en el que empieza y termina cada uno, creando un efecto similar al de una gráfica circular.



## 14. Transformaciones

Las transformaciones permiten mover, rotar, inclinar o escalar un elemento con respecto a su posición original.

Estas transformaciones se realizan después de que todos los elementos estén creados y movidos a sus posiciones.

Todas ellas están diseñadas y optimizadas para su uso con animaciones y adaptaciones posteriores.

### 14.1. Traslación

La traslación mueve el elemento unas coordenadas x e y con respecto al lugar en el que se encontraba.

```
transform: translate(50px, 100px);
```

Hay que tener en cuenta que va después del posicionamiento relativo o absoluto, por lo que su origen es el que estas otras propiedades estipulen.

Para realizar animaciones, es recomendable utilizar esto antes que los posicionamientos, pues está diseñado para ello.

### 14.2. Rotación

La rotación gira el elemento una cantidad de grados. Si estos son positivos girará en sentido de las agujas del reloj, mientras que si son negativos, en sentido contrario.

```
transform: rotate(20deg);
```

### 14.3. Inclinación

La inclinación transforma el elemento inclinándolo unos grados en base a un eje de coordenadas.

```
transform: skew(20deg, 30deg);  
transform: skewX(20deg);  
transform: skewY(30deg);
```

Existen cuatro formas, indicando la inclinación en ambos ejes, en el eje horizontal o en el eje vertical.

### 14.4. Escalado

El escalado aumenta o disminuye el tamaño del elemento y todo su contenido en base a un factor.

```
transform: scale(2, 3);  
transform: scaleX(2);  
transform: scaleY(3);  
transform: scale(3);
```

Existen cuatro formas, la primera indica el escalado de ancho y alto, el segundo el ancho solo, el tercero solo el alto y el cuarto ambos por el mismo factor.

### 14.5. Matrix

Con *matrix* se pueden estipular todas las anteriores transformaciones en una sola.

```
transform: matrix(1, -0.3, 0, 1, 0, 0);
```

Dado que está diseñado como una matriz, el orden puede no resultar obvio.

*matrix( scaleX(), skewY(), skewX(), scaleY(), translateX(), translateY() )*

### 14.6. Transformaciones 3D

Las anteriores son transformaciones en 2D, pero el estándar define una serie de transformaciones en 3D, aunque actualmente solo están implementadas las rotaciones.

Su uso es igual que el de la rotación, pero usando *rotateX*, *rotateY* o *rotateZ* para los distintos ejes.

## 15. Transiciones y animaciones

Las transiciones y las animaciones permiten crear efectos visuales continuos o que se activan bajo ciertas iteraciones del usuario, por ejemplo.

La forma más sencilla de lograr esto son las transiciones, que indican un tiempo de retardo en un cambio de propiedad. Su uso se combina con pseudoclases como *:hover* para lograr un efecto cuando el usuario interacciona con el elemento.

```
div {  
  width: 100px;  
  height: 100px;  
  background: red;  
  transition: width 2s;  
}  
  
div:hover {  
  width: 300px;  
}
```

En este ejemplo, cuando el usuario pasa el ratón por encima del elemento, su tamaño cambia a 300px. Al indicar la posición *transition* con un valor de 2s para la propiedad *width*, se le indica que, cuando esa propiedad cambie, lo hará a lo largo de 2s.

Si no se estipula una propiedad en concreto, ese tiempo de transición afectará a cualquier cambio.

La propiedad *transition-timing-function* permite establecer la fluidez de esta transición entre múltiples valores posibles, incluso se puede crear una transición personalizada.