

Hojas de Estilo CSS

En el tema anterior aprendimos a crear archivos HTML que aglutinaban toda la estructura y el contenido de una página web. Sin embargo, no hemos modificado ni personalizado la forma en la que el contenido se muestra en pantalla. Desde la introducción de HTML5 esta tarea es responsabilidad de CSS. Se trata de un lenguaje que ofrece instrucciones que podemos usar para asignar estilos a los diferentes elementos HTML. Algunos ejemplos básicos de estilo serían los colores, tipos y tamaños de letra, bordes de párrafos o fondos, entre muchos otros.

Por compatibilidad, los navegadores asignan un estilo por defecto (normalmente muy simple) a todos los elementos. Para que nuestra página web se presente como nosotros deseamos en pantalla deberemos modificar esos estilos por defecto usando hojas de estilo y el lenguaje CSS. En la actualidad usamos la tercera versión del lenguaje CSS, también denominada CSS3.

En CSS los estilos personalizados se definen usando propiedades. Los estilos declaran el nombre de la propiedad en cuestión y le asignan un valor, normalmente separado del nombre por el símbolo de los dos puntos. Casi todas las propiedades modifican un solo aspecto del elemento (por ejemplo el tipo de letra). Para realizar varios cambios de estilo habitualmente se utilizan múltiples propiedades. CSS posee un mecanismo que simplifica la tarea de asignar múltiples propiedades a un mismo elemento; se trata de las reglas CSS.

Reglas CSS

En CSS una regla es un conjunto de propiedades encerradas entre llaves, e identificadas por un selector. Los selectores delimitan los elementos HTML a los cuales se les va a aplicar el conjunto de propiedades contenidas en la regla.

Estructura de una regla CSS:

```
selector{  
    propiedad1: valor;  
    propiedad2 : valor;  
    ...  
}
```

Aplicación de estilos

Hay tres maneras de aplicar estilos en una página web:

- Estilos en línea. Utilizan el atributo global `style` directamente dentro del elemento. No se recomienda usar este método. Ej: `<p style="font-size: 20px">`
- Estilos en la cabecera. Dentro de la etiqueta `<head>` del archivo HTML podemos añadir la etiqueta `<style>...</style>` y añadir en ella todas las reglas CSS que se aplicarían en el archivo. Tampoco es un método muy recomendable porque en cualquier proyecto HTML con varias páginas habrá estilos que se repetirán, y con esta técnica tendríamos que repetir el mismo código en todas las páginas, con la dificultad de mantenimiento y la poca eficiencia que ello conlleva.
- Hojas de estilo. Es la solución preferible para cualquier proyecto. Agruparemos todas las reglas en un archivo externo al documento HTML, normalmente con extensión `.css`, y que se vinculará a la página web mediante la etiqueta `<link>` dentro de `<head>` (Ejemplo: `<link rel="stylesheet" href=".//css/archivo.css">`). Usar hojas de estilo permite aplicar las mismas reglas a múltiples archivos de manera muy sencilla. Además permite reemplazar todos los estilos a la vez simplemente cambiando el archivo `.css` asociado.

En caso de que se use más de una técnica para aplicar estilos y aparezcan valores distintos para las mismas propiedades, se aplicarán las que estén definidas de manera más específica. Esto quiere decir que si las propiedades entran en conflicto se aplicarán antes los estilos en línea que los estilos en la cabecera, y la opción con menos prioridad será la hoja de estilo.

CSS significa Cascading Style Sheets, u “hojas de estilo en cascada”. Por su estructura esto implica que los elementos en los niveles más bajos de la jerarquía heredan los estilos de los elementos en los niveles más altos.

Ejemplo:

Archivo HTML

```
<main>
...
<p>
...
</p>
...
</main>
```

Archivo CSS

```
main{
    font-size: 20pt;
}
```

En este caso el elemento `<p>` hereda todas las reglas del elemento padre `<main>`. Pero los estilos heredados pueden modificarse. Por ejemplo, para el mismo HTML podemos escribir:

CSS

```
main{
    font-size: 20pt;
}
p{
    font-size: 14pt;
}
```

En este caso el elemento `<p>` no heredará el tamaño de fuente de `<main>`: los párrafos se mostrarán con tamaño 14 puntos mientras que los elementos de `<main>` que no sean párrafos se mostrarán con tamaño de 20 puntos.

Selectores CSS

En CSS, un selector indica sobre qué elementos se aplicará un conjunto de propiedades descritas en una regla CSS. La forma más básica de usar un selector es escribir el nombre de la etiqueta HTML; de este modo las

propiedades encerradas entre llaves se aplicarán a todas las apariciones de la etiqueta en el documento HTML vinculado al archivo CSS.

Ejemplo: este código define un color concreto para el texto de todas las etiquetas `<h1>` del documento HTML asociado.

```
h1{  
    color: #C79B26 ;  
}
```

Sin embargo CSS permite una amplísima variedad de selectores para que nos ajustemos a las necesidades específicas de cada proyecto. Así podremos aplicar los estilos deseados de forma mucho más práctica y flexible. A continuación vamos a repasar los selectores más habituales.

- Selector universal (*). Se aplica a todos los elementos de una página. Se puede usar en combinación con otros selectores, como el descendiente.
- Selector de etiqueta. Es el más utilizado. Se aplicará a todas las apariciones de la etiqueta HTML que aparezca en la regla. Si queremos aplicar el mismo estilo a varias etiquetas diferentes podemos encadenarlas separadas por comas.

Ejemplo: este selector afecta a todas las etiquetas `<h1>`, `<h2>` y `<h3>`

```
h1,h2,h3{  
    ...  
}
```

- Selector descendiente (espacio en blanco). También es muy utilizado. En él definiremos estilos que se aplicarán solamente a elementos que se encuentran en un nivel inferior a otros en la estructura del documento HTML. Por ejemplo, si queremos aplicar un estilo a todos los elementos `` que estén dentro de una etiqueta `` lo expresaremos así:

```
ul li{  
    ...  
}
```

Es importante que no confundamos el selector de descendientes con la combinación de selectores. Así, un selector “`p a`” se aplica solamente a los elementos `<a>` que estén dentro de un elemento `<p>`, mientras que el selector “`p,a`” se aplica a todos los elementos `<p>` y a todos los elementos `<a>`.

Este selector puede usarse combinándolo con otros. Por ejemplo, este código aplicará estilos a todos los elementos `<a>` que estén dentro de un elemento `<p>`, independientemente de los elementos de nivel intermedio que aparezcan:

```
p * a{  
    ...  
}
```

- Selector de clase (.): Se aplica a todos los elementos con el mismo atributo global "class" que aparezcan en el HTML.
- Selector de identificador (#): Permite seleccionar un único elemento del archivo HTML a través de su nombre, que debe estar definido en el atributo global "id". No debemos confundirlo con el selector de clase. En cualquier página web bien construida no deberían aparecer elementos con un `id` repetido, mientras que no hay ningún problema en repetir la misma clase para muchos elementos que queramos que comparten formato. Por supuesto podemos combinar el selector de clase y el de identificador con otros. Por ejemplo, si queremos seleccionar todos los elementos con la etiqueta `<a>` que además pertenezcan a la clase "menu", podemos escribirlo así:

```
a.menu{  
    ...  
}
```

Más ejemplos:

- Todos los elementos `<p>` pertenecientes a la clase "algo": `p.algo`
- Todos los elementos de la clase "algo" que estén contenidos en un elemento `<p>`: `p .algo`
- Todos los elementos de tipo `<p>` y todos los elementos de la clase "algo": `p,algo`

Hay que señalar que los selectores de `id` no son muy recomendables según algunos desarrolladores. Como argumento indican que podemos incluir cualquier elemento en múltiples clases, lo cual es más flexible, además de que las clases pueden reutilizarse. Esto es más una recomendación que una norma obligatoria, basada en la experiencia y en el objetivo de construir código que sea lo más fácil de mantener que se pueda.

- Selector de hijo directo (>): Se usa para referirse a elementos que están exactamente en el nivel inmediatamente inferior respecto a otro. Es más restrictivo que el selector descendiente. Vemos la diferencia con un ejemplo:

```
...
<div>
...
  <p>
    ...
      <span class="s1">...</span>
    ...
  </p>
  <span class="s2">...</span>
</div>
...
```

En este contexto, un selector descendiente del tipo "div span" se aplicaría a los elementos `` s1 y s2, mientras que el selector "div>span" se aplicaría solamente a "s2", porque "s1" no es hijo directo de `<div>`.

- Selector de hermanos (~): Se aplica a todos los elementos que comparten un padre común y en los que el segundo elemento está precedido por el primero. Ejemplo:

```
...
<span class="s1">...</span>
<p>...</p>
<h1>...</h1>
<span class="s2">...</span>
...
```

Con este código, el selector "p~span" se aplicaría al `` "s2", pero no al "s1", porque, aunque está al mismo nivel, aparece antes de `<p>`.

- Selector de hermanos adyacentes (+): Es muy similar al anterior, aunque un poco más restrictivo. Solamente se aplicará a elementos que tengan el mismo parente y que además aparezcan consecutivamente en el código HTML. En el código de ejemplo anterior, el selector "p+span" no se

aplicaría a "s1" ni a "s2". No se aplicaría a "s2" específicamente porque hay un elemento `<h1>` entre `<p>` y ``. Otro ejemplo:

```
...
<h1>...</h1>
<h2 class="h2a">...</h2>
<h2 class="h2b">...</h2>
```

...

En este caso el selector "`h1+h2`" se aplicaría al elemento "h2a" pero no al "h2b".

- Selector de atributos ([]): Se utiliza en todos los elementos que presenten un cierto atributo, o que tengan un valor concreto en alguno de sus atributos. Permite el uso de comodines. Algunos ejemplos de uso:

- `input [required]`. Afectaría a todas las etiquetas `<input>` en las que esté presente el atributo "required".
- `a[href="https://www.google.es"]`. Se aplica a todos los enlaces que lleven a esta página web en concreto.
- `div[class ~= "principal"]`. Para todos los elementos con la etiqueta `<div>` cuyo nombre de clase incluye "principal". No serviría para clases como "principal1" o "miprincipal".
- `div[class *= "principal"]`. Incluirá a todos los elementos `<div>` con un atributo `class` que contenga la cadena de caracteres "principal" en cualquier posición.
- `div[class ^= "principal"]`. Afecta a todos los elementos `<div>` que tengan alguna clase que comience con la cadena de caracteres "principal".
- `div[class $= "principal"]`. Afecta a todas las etiquetas `<div>` que tengan alguna clase que termine con la cadena de caracteres "principal".

Pseudoclases CSS

En CSS, las pseudoclases son herramientas especiales que se utilizan para referenciar elementos HTML por medio de sus características, como su posición concreta en el código o sus condiciones actuales. Las más importantes son las siguientes:

- `:first-child`. Selecciona el primer elemento descendiente de otro.
- `:last-child`. Selecciona el último elemento descendiente de otro.
- `:only-child`. Selecciona un elemento si es el único hijo de otro.
- `:nth-child(num)`. Selecciona el enésimo hijo de otro elemento, que se especifica entre los paréntesis. Para esta pseudoclase podemos tener el caso particular de seleccionar únicamente los hijos impares escribiendo `:nth-child(odd)`, o los hijos pares con `:nth-child(even)`.
- `:nth-last-child(num)`. Es análogo a `:nth-child`, pero comenzando a contar los hijos del elemento por el final.
- `:empty`. Se aplica a los elementos que estén vacíos de contenido.

Ejemplo:

```
...
<article>
  <p class="p1">
    ...
  </p>
  <p class="p2">
    ...
      <span class="s1">...</span>
    ...
  </p>
  <p class="p3">
    ...
  </p>
  <h3>
    ...
  </h3>
```

Para este código podríamos usar las pseudoclases de la siguiente manera:

- ➔ `p:first-child`. Selecciona a "p1"
- ➔ `p:last-child`. Selecciona a "p3"
- ➔ `p:nth-child(2)` . Selecciona a "p2"
- ➔ `p:nth-child(odd)`. Selecciona a "p1" y "p3"
- ➔ `span:only-child`. Selecciona a "s1"

- `:first-of-type`. Se aplica al primer elemento de su tipo de entre un grupo de elementos hermanos.
- `:last-of-type`. Análogamente, se aplica al último elemento de su tipo de entre un grupo de hermanos.
- `:nth-of-type(n)`. Enésimo elemento de su tipo, comenzando a contar por el principio.
- `:nth-last-of-type(n)`. Enésimo elemento de su tipo, comenzando a contar por el final de un grupo de hermanos.

Ejemplo:

```
...
<main>
    <p class="p1">...</p>
    <p class="p2">...</p>
    <h1 class="h1a">...</h1>
    <p class="p3">...</p>
    <h1 class="h1b">...</h1>
    <p class="p4">...</p>
</main>
...
```

Para el código anterior, podemos escribir los siguientes selectores:

- `p:first-of-type`. Afecta a "p1"
- `h1:last-of-type`. Afecta a "h1b"
- `p:nth-of-type(3)`. Afecta a "p3"
- `p:nth-last-of-type(2)`. Afecta a "p3"
- `h1:nth-last-of-type(2)`. Afecta a "h1a"

- `:not()`. Selecciona todos los elementos que no coinciden con un selector. Suele usarse combinándolo con otros selectores. Por ejemplo, el selector `div:not(p)` se aplicaría a todos los elementos descendientes de `<div>` excepto a los `<p>`.

- `:is()`. Toma como argumento una lista de selectores, y como resultado se aplicará a todos los elementos que puedan seleccionarse por alguno de los elementos de esa lista. La especificidad de este selector será la del argumento más específico que esté contenido en la lista (comentaremos algo sobre especificidad más adelante). Básicamente sirve para generar código más legible y breve. Así, por ejemplo, escribir "`:is(ol,ul) li`" sería totalmente equivalente a "`ol li, ul li`".
- `:where()`. Es muy similar a `:is()`, y también acepta una lista de argumentos del mismo modo y con el mismo resultado. La única diferencia entre `:is()` y `:where()` es que este último tiene la mínima especificidad posible, de modo que cualquier selector declarado en otra parte del archivo CSS que también coincida con alguno de los selectores de `:where()` siempre lo sobrescribirá.

Pseudoclases dinámicas en CSS

Son pseudoclases que se centran en el estilo de los elementos cuando estamos interactuando con ellos.

- `:hover`. Especifica el estilo del elemento cuando el usuario pasa el ratón por encima.
- `:active`. Especifica cómo se verá el elemento cuando hacemos clic sobre él, y será visible hasta el momento en que dejemos de pulsar el botón.
- `:focus`. Define el estilo del elemento cuando el cursor está sobre él. Se utiliza sobre todo en elementos `<input>` dentro de formularios, y en raras ocasiones en enlaces.

Pseudoclases de enlaces CSS

Están diseñadas específicamente para utilizarse dentro de la etiqueta `<a>`.

- `a:link`. Especifica el estilo de los enlaces que todavía no han sido visitados por el usuario.
- `a:visited`. Para los enlaces que ya han sido visitados por el usuario.

Pseudoclases de formularios CSS

Todas estas pseudoclases tienen sentido cuando se utilizan en elementos dentro de un formulario, como por ejemplo la etiqueta `<input>`, entre otras.

- `:checked`. Para botones de radio, casillas de verificación o elementos `<option>`, cuando están activados.
- `:default`. Especifica el estilo de las opciones que aparecen marcadas por defecto en un formulario, como en los botones de radio, casillas de verificación y `<option>`. Será necesario combinarlo con el atributo "checked" dentro de `<input>` en el HTML.
- `:focus`. Para los elementos del formulario que tienen el cursor activo.
- `:in-range`. En elementos `<input>` de tipo numérico cuyo valor actual se encuentra comprendido entre los valores especificados en los atributos "min" y "max".
- `:out-of-range`. Se aplica a los elementos `<input>` numéricos cuyo valor actual está fuera del rango especificado en "min" y "max".
- `:valid`. Estilo para campos `<input>` cuyo contenido se valide correctamente, por ejemplo porque cumplan las condiciones de "pattern", o en campos que se validan de forma automática como los de tipo "url" o "email".
- `:invalid`. Se aplica a los estilos en los campos `<input>` cuyo contenido no se haya validado con éxito.
- `:optional`. Este estilo se aplicará a todos los elementos `<input>`, `<select>` o `<textarea>` que no sean obligatorios (porque no tienen activo el atributo booleano "required").
- `:required`. Estilo aplicable a los elementos obligatorios del formulario (todos aquellos que tienen activo el atributo "required").
- `:readonly`. Se aplica a todos los elementos que no pueden modificarse por el usuario. Estos elementos, normalmente de tipo `<input>`, tendrán el atributo booleano `readonly` activo.
- `:readwrite`. Se aplica a los elementos editables por los usuarios.

Pseudoelementos en CSS

En CSS los pseudoelementos se emplean para definir el estilo de partes de un elemento. Es frecuente usarlos para insertar algún contenido antes o después de determinados elementos, o para formatear líneas concretas dentro de un párrafo.

- `::first-letter`. Se usa para definir el estilo del primer carácter de un texto. Solamente puede aplicarse sobre elementos de bloque en HTML.
- `::selection`. Modifica la apariencia del texto que el usuario seleccione con el ratón. Para este pseudoelemento solamente tendrán efecto las propiedades "color", "background", "cursor" y "outline".
- `::first-line`. Aplica estilos únicamente a la primera línea de un párrafo o texto.
- `::before`. Con este pseudoelemento CSS crea un elemento que no está en el código HTML y que será el primer hijo del elemento que se selecciona. Habitualmente se utiliza para añadir contenido estático mediante la propiedad "content". Dicho contenido se mostrará por defecto en línea con el texto, aunque se puede cambiar.
- `::after`. El mismo efecto que `::before`, pero el contenido se mostrará justo después del elemento.

Ejemplo:

```
blockquote::before{  
    content: "<<" ;  
    color: "orangered";  
}
```

```
blockquote::after{  
    content: ">>" ;  
    color: "orangered";  
}
```

Propiedades CSS

Las propiedades son la piedra angular de CSS. todos los estilos que podemos aplicar a un elemento vienen definidos mediante alguna propiedad. Hay cientos de propiedades válidas en CSS, por lo que en este tema nos vamos a centrar en conocer y estudiar únicamente las más importantes. A la hora de clasificar las propiedades CSS podemos establecer dos tipos fundamentales:

- Propiedades de formato: Dan forma a los elementos y a su contenido.
- Propiedades de diseño: Están enfocadas en determinar el tamaño, posición u otras propiedades de los elementos en pantalla.

Orden de aplicación de las propiedades CSS

En una hoja de estilo ideal no debería haber colisiones entre las diferentes propiedades que se aplicasen. Sin embargo, si se producen colisiones (propiedades a las que se les aplican valores diferentes en la misma hoja de estilo) existe una jerarquía bien definida que determina cuál se aplicará finalmente. El orden de aplicación, de mayor a menor prioridad, es el siguiente:

- Importancia. El modificador `!important` sirve para que una determinada regla se salte la cadena de prioridades y se aplique independientemente de la prioridad que le correspondería originalmente.
- Especificidad. Se refiere al número de etiquetas que es capaz de abarcar un selector. De mayor a menor especificidad tenemos:
 - Estilos declarados dentro de esa etiqueta, que solo afectan a dicha etiqueta
 - Selectores `id (#)`
 - Selectores de clase (`.`)
- Orden del código. Si con los criterios anteriores no se hubiera resuelto la colisión, simplemente se aplicará el selector que esté definido más tarde, es decir, aquel que aparezca el último en el código de la hoja de estilo.

Uso de colores en CSS

Para especificar el color de un elemento en CSS podemos tener varias opciones:

- Código hexadecimal RGB. En este caso se usan 6 dígitos hexadecimales para especificar el color. Los dos primeros representan el nivel de rojo, los dos centrales el nivel de verde y los dos últimos el nivel de azul. ejemplo: #1E09EE. Si además queremos especificar el nivel de transparencia usaremos 8 dígitos hexadecimales, siendo los dos últimos el nivel de opacidad del elemento (valores más bajos representan más transparencia). Ejemplo: #1E09EE77.
- Valor RGB. En este caso especificaremos la cantidad de rojo, verde y azul con un valor numérico entre 0 y 255. Ejemplo: `rgb(105,109,204)`. Si queremos añadir transparencia con este método, lo haremos especificando el valor “alfa” entre 0 (totalmente transparente) y 1 (totalmente opaco) para representar el valor de transparencia. Igual que en el caso anterior, valores más bajos representan mayor transparencia. Ejemplo: `rgba(173,59,191,0.2)`.
- Valor HSL (Hue, Saturation, Value). Este es otro espacio de color análogo al RGB, que supuestamente está más en consonancia con cómo percibimos el color los seres humanos. Para ello se introduce un valor de H (matiz) entre 0 y 360, seguido de los valores S (saturación) y L (luminosidad) expresados en porcentajes. Ejemplo: `hsl(210, 75%, 50%)`. Si queremos añadir transparencia se añade un valor al final entre 0 y 1, igual que en el espacio RGB. Ejemplo: `hsla(125, 60%, 75%, 0.3)`.
- Nombre. Todos los navegadores soportan códigos identificativos para 140 colores frecuentes. Usar un nombre para el color nos permitirá identificarlo fácilmente sobre el código, aunque quizás no nos permita expresar todos los colores que nos suelen hacer falta en un proyecto. Algunos ejemplos: red, black, pink, olive, plum, forestgreen, darkorange, mediumaquamarine, gold,...

Variables en CSS

Si hay valores que se van a utilizar múltiples veces en nuestro código, como por ejemplo para definir un color que se va a referenciar múltiples veces, o el valor de una propiedad o atributo, una buena idea es usar variables globales. Tienen la ventaja de facilitar la lectura del código y además permiten que cambiando su valor solamente en la declaración de la variable se cambie en todas las referencias a la variable en otras zonas del código. Para usar variables globales

podemos declararlas dentro del selector `:root`, que hace referencia al elemento raíz del documento HTML, que en condiciones normales debería ser la etiqueta `<html>`. Ejemplo:

- Definición de variables globales

```
:root{  
    --rosa: #DD68B0;  
    --morado: #811181;  
}
```

- Uso de variables globales

```
body{  
    background-color: var(--rosa);  
}  
h1{  
    color: var(--morado);  
}
```

También es posible crear variables con un ámbito más reducido si las declaramos dentro de un selector, si bien esto es menos frecuente.

Unidades de medida en CSS

Muchas propiedades CSS necesitan la especificación de un tamaño o longitud para tener sentido. Esto es particularmente relevante en las propiedades de diseño de una página web. En CSS hay dos tipos de unidades de medida:

- Absolutas. Tienen siempre el mismo tamaño.
- Relativas. Están vinculadas a otro elemento de referencia (como por ejemplo el tamaño de la ventana, o el tamaño definido en un elemento padre), por lo que su tamaño será diferente según las circunstancias. Si queremos usar un diseño adaptable será necesario el uso de unidades relativas.

Unidades absolutas de medida:

- **px**: Píxeles. Equivale a un punto en la pantalla.
- **mm**: Milímetros.
- **cm**: Centímetros.
- **in**: Pulgadas. una pulgada equivale a 2,54 centímetros.
- **pt**: Puntos. Un punto equivale a 1/72 de pulgada.

Unidades relativas de medida:

- **%**: Porcentaje.
- **em**: En el caso de propiedades tipográficas, **1em** se refiere al valor por defecto del tamaño de un elemento (en anchura) en el dispositivo actual. Por ejemplo, escribir **1.2em** significaría ampliar el tamaño de la letra un 20% respecto al tamaño por defecto.
- **rem**: Significa “root em”, y es bastante parecido a **em**, pero con una diferencia. Mientras que **em** podría tener un valor diferente para cada tipo de elemento al que se refiera (no solamente para propiedades tipográficas), **rem** siempre tiene el mismo tamaño por defecto: el del elemento raíz del código HTML. Esto sucede porque diferentes elementos tienen por defecto tamaños predefinidos diferentes, y con **rem** se podrían armonizar todos. En cualquier caso es mucho más frecuente usar **em** que **rem**.
- **ex**: Si **em** se refiere a la anchura de la fuente por defecto, **ex** se refiere a la altura en vertical (**x-height**) de la fuente por defecto. Se usa muy poco.
- **vh** (viewport height): **1vh** se refiere al 1% de la altura de la ventana en la pantalla. Así, por ejemplo, un tamaño de **50vh** equivale a la mitad del espacio de la pantalla medido verticalmente.
- **vw** (viewport width): **1vw** se refiere al 1% de la anchura de la ventana en pantalla. Al igual que **vh**, cuando se modifica el tamaño de la ventana su valor cambia.

Propiedades abreviadas (shorthand)

Son propiedades que permiten asignar el valor de múltiples propiedades CSS al mismo tiempo. Facilitan la legibilidad del código CSS generado y pueden ahorrar

tiempo a los desarrolladores. Suelen agrupar varias propiedades relativas al mismo tema, que habitualmente se usan juntas. Por ejemplo, la propiedad abreviada "font" define todas las opciones de estilo más frecuentes relacionadas con la tipografía.

Ejemplo:

```
p{  
    margin-top : 20px;  
    margin-right: 10px;  
    margin-bottom: 15px;  
    margin-left: 10px;  
}
```

es equivalente a

```
p{  
    margin: 20px 10px 15px 10px;  
}
```

Otro ejemplo:

```
div{  
    border-width: 5px;  
    border-style: dotted;  
    border-color: #A74329;  
}
```

es equivalente a

```
div{  
    border: 5px dotted #A74329;  
}
```

Propiedades CSS: Tipografía

- **font-family:** Especificaremos familias de fuentes. Si ponemos más de una significa que, en el caso de que la primera no estuviera disponible en el sistema, será sustituida por la segunda, y así sucesivamente.
 - `font-family: "Arial Black", Verdana, sans-serif;`
- **font-size:** Tamaño de fuente (absoluto o relativo)
- **font-weight:** Grosor de la fuente. Podemos poner nombres para el grosor como "normal" o "bold", o poner un valor numérico para el grosor si la fuente concreta lo admite. En este último caso el valor irá desde 100 (letra más fina) hasta 900 (letra más gruesa), siendo 400 el grosor equivalente a "normal".

- `font-style: normal / italic / oblique`. El valor `italic` pone el texto en cursiva, mientras que el valor `oblique` "distorsiona" el texto inclinándolo un poco hacia la derecha.
- `font-variant: normal / small-caps`. Para texto en formato versalita.
- `line-height`: Especifica la distancia entre líneas del texto.
- Propiedad shorthand `font`. Equivale a: "`font-style font-variant font-weight font-size / line-height font-family`". En esta propiedad solamente son obligatorios `font-size` y `font-family`. Los tres primeros parámetros (`font-style`, `font-variant` y `font-weight`) pueden aparecer desordenados, pero el resto, si aparecen, deben hacerlo en el orden especificado.
 - `font: italic 600 1.4em "courier new";`
 - `font: 1.5em / 1.2em "Arial";`
 - `font: bold 16pt / 18pt "Verdana";`

Para evitar depender de las fuentes que cada usuario tenga instaladas en sus dispositivos, es frecuente importarlas desde algún tipo de repositorio. Por ejemplo desde Google Fonts podemos seleccionar las fuentes que queramos y copiar un código en la cabecera de nuestro HTML para que sean accesibles. Si queremos utilizar fuentes más específicas que no estén en un repositorio podemos utilizar la regla `@font-face`. Esto no es una propiedad CSS como tal, y suele ubicarse al final del archivo CSS. Especificaremos el nombre que vamos a utilizar para la fuente junto con la ruta a los archivos necesarios alojados en nuestro servidor.

Ejemplo:

```
@font-face{  
    font-family: 'Alef';  
    font-style: normal;  
    font-weight: 400;  
    src: url ('../fonts/alef.woff') format ('woff');  
}
```

Propiedades CSS: Texto

- `text-align: left/right/center/justify`. Alineación del texto.
- `letter-spacing <medida>`: Especifica el espacio entre caracteres.

- direction: ltr / rtl. Si el texto se presenta de izquierda a derecha, o al contrario.
- word-wrap: normal / break-word. Con "normal" solamente se dividen las palabras de un párrafo en los espacios designados para ello (etiqueta <wbr> o ­ en HTML), y con "break-word" el navegador romperá automáticamente palabras largas cuando lo considere oportuno, pudiendo romperlas por cualquier sitio.
- word-spacing <medida> : Anchura del espacio entre las palabras del texto.
- text-indent <medida> : Tamaño de la sangría de primera línea en un párrafo.
- vertical-align. Se usa normalmente con imágenes que van mezcladas con texto (en línea) o para pequeños fragmentos en un texto más largo, y especifica la alineación vertical. Algunos valores posibles: baseline / top / bottom / text-top / sub / sup / text-bottom / baseline.
- line-height <medida>: Espacio entre líneas.
- text-align-last: left / right / center / justify. Alineación específica para la última línea de un párrafo.
- text-decoration: underline / overline / line-through / none. Resalta texto mediante una línea, o tachándolo en el caso de line-through.

Propiedades CSS: Color

Ya hemos comentado un poco sobre cómo se utilizan los colores en CSS. Habitualmente se asignan mediante la propiedad "color". En esta propiedad podremos especificar dígitos hexadecimales, nombres de color o las funciones rgb() o hsl() para definir el color de un elemento. Para especificar la transparencia de un elemento podemos hacerlo mediante el uso de 8 dígitos hexadecimales o las funciones rgba() y hsla(), pero también podemos utilizar la propiedad "opacity", con un valor entre 0 (transparencia total) y 1 (opacidad absoluta), o un porcentaje. La diferencia entre usar un color transparente o la propiedad "opacity" es que esta última por defecto se hereda por todos los elementos hijos, lo cual no ocurre con "color". Adicionalmente podríamos usar la propiedad "visibility: hidden;" para ocultar un elemento.

Propiedades CSS: Fondos

- **background-size:** Tamaño del fondo (si es una imagen).
 - **auto:** tamaño original de la imagen.
 - **<medida>:** Especificar el tamaño de la imagen. Si ponemos dos valores el primero será el ancho y el segundo el alto de la imagen. Está permitido utilizar medidas relativas. También podemos poner una medida y la otra dejarla en "auto".
 - **Porcentaje:** Tamaño relativo (en tanto por ciento) al de su contenedor.
 - **cover:** Redimensiona la imagen para ocupar todo su contenedor, sin importar que la imagen se tenga que pixelar o recortar en el proceso.
 - **contain:** Redimensiona la imagen de modo que sea completamente visible; si es más grande que el contenedor, se reducirá.
- **background-image: none / url:** Define la ubicación de la imagen de fondo si la vamos a usar. Podemos especificar varias imágenes separadas por comas.
- **background-repeat: no-repeat / repeat / repeat-x / repeat-y.** Si queremos que la imagen se repita en mosaico en caso de haber espacio para ello en el fondo. Podemos hacer que solamente se repita en horizontal (**repeat-x**), o solamente en vertical (**repeat-y**).
- **background-attachment: fixed / scroll.** Con "fixed", el fondo permanece estático en la ventana, mientras que con "scroll" se desplazará al hacer "scrolling" en la pantalla.
- **background-color: transparent / <color> .** Fondo de color sólido o transparente.
- **background-position:** Indica la posición desde la cual empieza a mostrarse la imagen de fondo dentro de su contenedor. Por defecto será desde la esquina superior izquierda. Otros valores posibles:
 - **left top / left center / left bottom**
 - **right top / right center / right bottom**

- center top / center center / center bottom
 - <medida x> <medida y> Posiciones absolutas en horizontal y en vertical.
 - x% y% Porcentajes respecto al tamaño del contenedor, en horizontal y en vertical.
- **background-origin:** Define el punto desde el cual comenzará a pintarse el fondo del elemento. Posibles valores:
- padding-box (valor por defecto). Comenzará en el borde superior izquierdo del padding, o relleno. Esta propiedad padding define un espacio vacío en el borde interior del elemento.
 - border-box. Empezará en la esquina superior izquierda del borde del elemento.
 - content-box. Empezará en el borde superior izquierdo del contenido, por lo que en caso de haber "padding", este no tendrá fondo.
- **background-clip:** Determina cómo se recorta la imagen de fondo si se extiende más allá de los límites de su contenedor. Es muy similar a la propiedad **background-origin** y tiene las mismas opciones.
- Propiedad shorthand **background**: Permite especificar en una sola propiedad CSS muchas de las opciones relativas a fondos que acabamos de ver.

Propiedades CSS: Bordes

- **border-width:** <medida> / thin / thick / medium. Grosor del borde.
- **border-style:** none / dashed / dotted / solid / double / inset / outset / ridge. Estilo de la línea que conforma el borde. Suele usarse "solid".
- **border-left-style**, **border-right-style**, **border-top-style**, **border-bottom-style**: Propiedades para particularizar el estilo de los bordes.
- **border-color:** <color>. Color del borde.

- `border-left-color`, `border-right-color`, `border-top-color`, `border-bottom-color`: Propiedades para particularizar colores de los bordes.
- `border-radius: <medida>`. Especifica el radio de las esquinas, con lo que se crean bordes redondeados.
- `border-top-left-radius`, `border-top-right-radius`, `border-bottom-left-radius`, `border-bottom-right-radius`: Propiedades para personalizar cada uno de los radios de las esquinas.
- Propiedad shorthand `border`. Equivale a “`border-width border-style border-color`”. Solamente “`border-style`” es obligatoria.
 - `border: 3px solid pink;`
- Análogamente tenemos las propiedades shorthand `border-top`, `border-bottom`, `border-left` y `border-right`, que se aplican igual que “`border`” en un solo lado.

Propiedades CSS: Sombras

- `text-shadow`. Aplica sombra a un texto. Acepta hasta 4 parámetros, que son el color de la sombra, la distancia en horizontal (positiva o negativa) de la sombra, la distancia en vertical y el radio de difuminado de la sombra (cuanto mayor sea, menos nítida será la sombra). Esta propiedad solamente puede aplicarse sobre elementos que tengan texto. Para aplicar sombras sobre otros tipos de elementos se usa la propiedad `box-shadow`. Se pueden mezclar varias sombras diferentes si las separamos por comas.
 - `text-shadow 3px -2px 5px blue;`
- `box-shadow`. Es muy similar a `text-shadow`, pero en este caso la sombra se aplica a la caja que rodea a un elemento de bloque, como por ejemplo un párrafo o un título. Los parámetros son los mismos que para `text-shadow`. Como curiosidad, si omitimos especificar el color de la sombra se heredará del color del texto dentro del elemento, y si omitimos el radio de desenfoque se mostrará una sombra completamente nítida.
 - `box-shadow: 5px -5px 3px red, -10px 2px 5px green;`

Tanto en `box-shadow` como en `text-shadow` podemos añadir el parámetro `inset` al final para hacer que la sombra sea interna en lugar de externa.

- Ejemplo: `box-shadow #777777 5px 5px inset;` . Propiedades en otro orden, sin radio, sombra interna.

Propiedades CSS: Gradientes

Un gradiente o degradado se forma con colores que van pasando paulatinamente de un color a otro. Los gradientes se crean como imágenes y se añaden como fondos de otros elementos, por ejemplo usando propiedades como `background-image` o `background`.

- `linear-gradient` (posición, ángulo, colores). Degradados lineales.
 - `posición`: Dónde empieza el gradiente. Puede ser `top, bottom, left, right, un porcentaje o un valor de medida absoluta`.
 - `ángulo`: Dirección del gradiente. Se especifica en grados con la unidad `deg`. Por ejemplo, `180deg`.
 - `colores`: Conjunto de colores que conforman el gradiente. Pueden incluir transparencia. Cuando hay varios colores se separan por comas, y se suele incluir un valor en porcentaje que indica hasta dónde llega ese color en el conjunto del degradado.

Ejemplo:

```
body{  
background: linear-gradient (180 deg, rgba(64,10,117)  
10%, rgba(11,105,122) 40%, rgb(173,140,27) 75%, rgb  
(255,178,35) 100% );  
}
```

- `radial-gradient` (posición, forma, colores, extensión). Degradados radiales / elípticos.
 - `posición`: Dónde empieza el gradiente.
 - `forma`: Puede ser "circle" o "ellipse"
 - `colores`: Lista de colores que forman el gradiente separados por coma con su correspondiente porcentaje, igual que se hace en `linear-gradient`.
 - `extensión`: Determina la forma que va a adquirir el degradado.

- closest-side. El gradiente acaba cuando llega al lateral más próximo de su contenedor.
- farthest-side. El gradiente acaba al llegar al lateral más lejano a su contenedor.
- closest-corner, farthest-corner. Lo mismo que las opciones anteriores, pero aplicado a las esquinas en lugar de los laterales del contenedor.

Por defecto los degradados radiales empiezan en el centro de su contenedor, pero eso se puede modificar con el parámetro at:

- top right / top center / top left
- bottom right / bottom center / bottom left
- center right / center center / center left

Ejemplo:

```
body{  
    background: radial-gradient (circle at top right,  
        #BCD632 0%, #1EBF4E 40%, #23390E 100% ) ;  
}
```

También se pueden crear patrones de rayas o círculos que se repiten usando repeating-linear-gradient() y repeating-radial-gradient() respectivamente. Para terminar con los gradientes simplemente mencionaremos que existen los gradientes cónicos, definidos por la propiedad conic-gradient(), si bien su uso es poco común.

Como la sintaxis de estos degradados es un poco compleja, existen numerosas páginas web que nos generan el código para un gradiente de forma automática, facilitándonos el proceso.

Propiedades CSS: listas

Este conjunto de propiedades permite modificar cómo se presentan los ítems de listas, definidos en HTML con o bien con .

- **list-style-type**. Define el tipo de viñeta de la lista.
 - En listas no ordenadas, podemos usar `discs` (valor por defecto), `none`, `asterisks`, `circle` o `square`, entre otras opciones.
 - En listas ordenadas, algunos de los valores posibles son `none`, `decimal`, `decimal-leading-zero`, `lower-roman`, `upper-roman`, `lower-alpha`, `upper-alpha` o `lower-greek`.
- **list-style-position**: Determina cómo se posiciona la viñeta respecto al texto.
 - `outside`. Valor por defecto. La viñeta está “fuera” del texto.
 - `inside`: Las viñetas forman parte del texto y lo “empujan” hacia la derecha.
- **list-style-image: <URL>** . Reemplaza la viñeta por defecto con una imagen de nuestra elección.
- Propiedad shorthand **list-style**: Permite establecer los valores para las tres propiedades anteriores en una única declaración. Ejemplo: `list-style square inside;`

Filtros CSS: propiedad “filter”

Se usan sobre todo en imágenes, aunque pueden aplicarse sobre muchos elementos. Los filtros agregan efectos visuales a los elementos y su contenido. Para usarlos hay que escribir la propiedad **filter**, seguida del filtro o conjunto de filtros que queramos aplicar.

- `blur(valor)`. Difumina el elemento; cuanto mayor sea el valor, más difuminado estará.
- `grayscale(valor)`. Convierte los colores de una imagen a escala de grises. Se pone en porcentaje, con 100% siendo la imagen totalmente en blanco y negro.
- `drop-shadow(x y tamaño color)`. Genera una sombra arrojada plana alrededor del elemento. Con `x` e `y` se especifica el desplazamiento en horizontal y vertical de dicha sombra.
- `sepia(valor)`. Convierte la imagen a tonos sepia. Se usa con porcentajes igual que el filtro `grayscale()`.

- `opacity(valor)`. Modifica la transparencia del objeto. Un valor de 100% equivale a un elemento totalmente opaco, y un 0% es totalmente transparente.
- `brightness(valor)`, `contrast(valor)`. Modifican el brillo y contraste del elemento respectivamente. También se expresan en porcentajes, pudiendo ir más allá del 100%. En este caso un valor del 100% dejaría el elemento sin cambios.
- `hue-rotate(grados)`. Rotación del valor de matiz de los colores de la imagen en grados. Podemos usar valores entre `0deg` y `360deg`.
- `invert(valor)`. Invierte los colores de la imagen. Un valor del 100% la presentaría con todos los colores reemplazados por el opuesto.
- `saturate(valor)` Modifica la saturación (intensidad de los colores) en una imagen. El valor de 100% representa la imagen en su estado original, un 0% representaría la imagen en escala de grises y valores por encima del 100% presentarían una imagen más saturada que la original.

Transformaciones CSS: propiedad “transform”

Con las transformaciones podemos modificar tamaño, posición o apariencia de los elementos HTML. Podemos usar varias transformaciones sobre un mismo elemento, que se aplicarán en el orden en que se declaran.

- `scale(valor-x,valor-y)` / `scaleX(valor)` / `scaleY(valor)`. Operaciones para modificar el tamaño del elemento. Si solamente ponemos un valor dentro de `scale()` se sobreentiende que será el mismo para horizontal y vertical. Los valores por debajo de 1 reducen el elemento, mientras que los valores mayores lo aumentan respecto al tamaño original.
 - `scale(1.5)` Aumenta un 50% el tamaño en horizontal y vertical.
 - `scale(1.25,2)` Aumenta el tamaño un 25% en horizontal y lo duplica en vertical.
 - `scaleX(1.5)` Equivale a `scale(1.5,1)`
- `rotate(ángulo)`, `rotateX(ángulo)`, `rotateY(ángulo)`, `rotateZ(ángulo)`: Rota el elemento el número de grados especificado,

que puede ser positivo o negativo. La transformación `rotate()` es bidimensional, mientras que las demás son tridimensionales.

- `translate(x, y)` Desplaza el elemento a la posición determinada por los atributos `x` e `y`.
- `skew(ánguloX,ánguloY) / skewX(ángulo) / skewY(ángulo)`. Inclinan el elemento generando una distorsión, en horizontal y/o vertical.

Ejemplo:

```
img:hover{  
    transform: scale(1.2);  
    opacity: 75%;  
}  
header{  
    ...  
    transform: translateY(50px) rotate (20deg);  
    ...  
}
```

Además de las transformaciones anteriores, podemos tener sus versiones en tres dimensiones:

- `scale3d(x,y,z)` Asigna una nueva escala al elemento en un espacio tridimensional.
- `rotate3d(x,y,z,ángulo)` Rota el elemento sobre un ángulo y un eje específicos. Los valores `x`, `y`, `z` definirán el vector que será el eje sobre el cual se realizará el giro determinado por el ángulo especificado.
- `rotateX(ángulo), rotateY(ángulo), rotateZ(ángulo)`. Simplificación de `rotate3d` para girar un elemento sobre un solo eje. Se suele usar sobre todo en elementos con animaciones o transiciones.
- `translate3d(x,y,z)` Mueve un elemento hacia una nueva posición en un espacio tridimensional.
- `perspective(valor)` Añade un efecto 3D de perspectiva a un elemento. Cuanto mayor sea, más lejos se presentará el objeto sobre el eje Z. Suele combinarse con rotaciones 3D y con estas otras propiedades:

- `perspective-origin(x,y)`: Indica la posición desde la que "miramos" el elemento.
- `backface-visibility`: Indica si el reverso del elemento será visible en caso de que la transformación lo deje expuesto. Si queremos ocultarlo escribiremos "hidden" y en caso contrario, "visible".

En el uso de transformaciones puede ser muy interesante emplear la propiedad `transform-origin`. Sirve para definir desde qué punto servirá de origen para aplicar las transformaciones. Tiene sentido sobre todo para transformaciones tridimensionales. Por defecto las transformaciones parten desde el centro del elemento, pero podemos personalizar el comportamiento según el efecto que se quiera conseguir.

Ejemplos:

```
transform-origin: top left;  
transform-origin: 20% 40%;  
transform-origin: 50px 75px;
```

Transiciones CSS

Las transiciones son muy importantes en CSS. Habitualmente se combinan con pseudoclases dinámicas como `:hover` para animar de forma simple los elementos. Simplificando podemos decir que las transiciones crean una animación con el cambio de valor de alguna propiedad de un elemento. Para crear transiciones debemos definir a qué propiedades afecta y cuánto dura la transición. Las principales propiedades que gobiernan las transiciones en CSS son:

- `transition-property`: Especifica qué propiedades del elemento participan en la transición. Si queremos que todas las propiedades se vean afectadas, podemos poner el valor "all".
- `transition-duration`: Especifica la duración de la transición en segundos. Ej: 2s.
- `transition-delay`: Determina el tiempo en segundos que el navegador esperará antes de iniciar la transición desde que se carga la página.

- **transition-timing-function:** Determina cómo cambia la velocidad de la transición durante su reproducción. Posibles valores:
 - **ease.** Comportamiento por defecto. Inicio lento, parte central a velocidad rápida y final lento otra vez.
 - **linear:** Velocidad constante desde el inicio hasta el final de la reproducción.
 - **ease-in:** Inicio lento, luego velocidad normal hasta el final.
 - **ease-out:** Velocidad de reproducción normal y lento al final.
 - **ease-in-out.** Inicio y final de la reproducción lentos, con la parte central a velocidad normal.
 - **cubic-bezier(valor1, valor2, valor3, valor4).** Define una curva personalizada para la velocidad de reproducción, que podría incluso hacer que el elemento avanzara hacia atrás en partes de la animación.
- Propiedad shorthand **transition:** Combina todas las propiedades anteriores en una. Es obligatorio especificar al menos una propiedad sobre la que hacer la transición y un tiempo de transición. Se pueden especificar varias propiedades separadas por comas. Es la forma más habitual de realizar transiciones en CSS.

Ejemplo:

```
div.tr{  
    width: 200px;  
    height: 200px;  
    background: #939;  
    transition-property:all;  
    transition-duration:3s;  
}
```

Equivale a:

```
div.tr{  
    width: 200px;  
    height: 200px;  
    background: #939;  
    transition: all 3s;  
}
```

Otro ejemplo:

```
...
transition-property: width, height;
transition-duration: 2s,3s;
transition-timing-function: linear;
```

Es equivalente a:

```
...
transition: width 2s, height 3s linear;
...
```

Animaciones CSS

Si queremos insertar animaciones más complejas que las que permiten las transiciones, CSS dispone de animaciones, en las cuales definiremos varios estados (normalmente más de dos) que servirán como fotogramas clave de una película. Algunas de las propiedades implicadas en las animaciones son:

- **animation-name:** Nombre con el que se identifican los fotogramas clave de una animación.
- **animation-duration:** Duración en segundos de cada ciclo de la animación.
- **animation-timing-function:** Variaciones de velocidad en la animación. Se pueden poner las mismas opciones que en la propiedad `transition-timing-function`.
- **animation-delay:** Cuánto tiempo se espera desde que se carga la página para comenzar a reproducir la animación.
- **animation-iteration-count:** Cuántas veces se reproducirá la animación. Podemos poner cualquier número, o "infinite" si queremos que se reproduzca sin parar.
- **animation-direction:** En qué orden se reproduce la animación.
 - **normal:** Por defecto, del principio al final.
 - **reverse:** Empieza por el final y va hacia el principio.
 - **alternate:** Del principio al final y luego de vuelta al principio.
 - **alternate-reverse:** Del final al principio y después otra vez hasta el final.

- **animation-fill-mode:** Cómo se quedan los elementos animados cuando la animación no se está reproduciendo, ya sea porque no ha empezado o porque ha terminado.
 - **forwards:** Se quedan como estaban al final de la animación.
 - **backwards:** Los elementos se quedan como estaban antes de comenzar la animación.
 - **both:** Mezcla los estilos iniciales y finales de la animación.
 - **none:** Es el valor por defecto. No se aplican estilos incluidos de la animación si no se está reproduciendo.
- Propiedad shorthand **animation:** Permite definir todos los parámetros de una animación en una única propiedad. Es el método habitual para definir animaciones.

Además de la configuración general de la animación es obligatorio declarar los fotogramas clave de la animación con la regla `@keyframes`. Esta regla debe preceder al mismo nombre que se haya definido en la propiedad `animation-name`, y debe incluir las propiedades que se modifican en cada fotograma clave, junto con sus valores en cada uno de ellos. Las posiciones en la línea temporal de los fotogramas clave se representan con porcentajes, siendo 0% el fotograma inicial y 100% el fotograma final. El navegador se encargará de generar la animación para que se produzcan transiciones suaves entre un fotograma clave y el siguiente.

Ejemplo de animación CSS:

```
h1{  
    text-align:center;  
    font-family: "Arial", sans-serif;  
    font-size: 2em;  
    background-color: darkblue;  
    border: 5px solid black;  
    animation anim 10s ease-in-out 0s infinite  
    alternate none;  
}
```

Lo escrito en la propiedad animation equivale a:

```
...
    animation-name: anim;
    animation-duration: 10s;
    animation-timing-function: ease-in-out;
    animation-delay: 0s;
    animation-iteration-count: infinite;
    animation-direction: alternate;
    animation-fill-mode: none;
...
```

En la regla @keyframes especificamos los fotogramas clave de nuestra animación de ejemplo:

```
@keyframes anim{
    20%{
        background-color: deepskyblue;
    }
    40%{
        background-color: powderblue;
    }
    60%{
        background-color: midnightblue;
    }
    80%{
        background-color: azure;
    }
    100%{
        background-color: darkblue;
    }
}
```