Módulo 2

HTML y CSS

CODE SPACE ACADEMY

Luis Guzmán

Tema 5
Propiedades intermedias CSS

CODE
SPACE
ACADEMY

CSS Custom Properties

Las `CSS Custom Properties` (muchas veces conocidas por variables CSS) son un mecanismo de CSS que permite dar un valor personalizado a las propiedades. El objetivo principal suele ser evitar escribir múltiples veces ese valor, y en su lugar, ponerle un nombre más lógico y fácil de recordar, que hará referencia al valor real.

Para definir una `custom property` haremos uso de los dos guiones `--` previos al nombre que queramos utilizar. Además, debemos fijarnos en el elemento que definimos la variable, en este ejemplo la pseudoclase `:root`:

```
:root {
  --background-color: black;
}
```

La pseudoclase :root hace referencia al elemento raíz del documento, o lo que es lo mismo, al elemento `<html>`. La diferencia de utilizar html o `:root` como selector es que este último tiene algo más de especificidad CSS. Mientras que `html` tiene `001`, `:root` tendría `010`.

Al colocarla en `:root` estamos definiendo que la custom property estará definida para el ámbito de esa etiqueta `<html>` (o cualquier elemento hijo), es decir, a todo el documento. Sin embargo, ya veremos que podemos aplicar estas variables sólo a partes concretas del `DOM` de nuestra página.

Utilizar una variable CSS

A la hora de utilizar una `custom property`, hay que utilizarla dentro de la expresión `var()`.

```css
.element {
  background: var(--background-color);
}
```

```css
.element {
  background: var(--background-color, blue);
}
```

En este caso estamos aplicando a la propiedad `background` el valor que contiene `--background-color` para el elemento `.element`. Esto último es muy importante entenderlo, ya que una `custom property` puede tener diferentes valores dependiendo del ámbito en el que se encuentra.

Además, es muy recomendable que la expresión `var()` tenga dos parámetros. El primero de ellos, la `custom property` en cuestión, el segundo de ellos, el valor por defecto en el caso de que esa propiedad no esté definida en el ámbito actual.

Ámbito de las custom properties

Las custom properties no se pueden declarar únicamente en el :root, también tenemos la posibilidad de hacerlo en diferentes ámbitos:

```
<div class="parent">
  <div class="first child">First child</div>
  <div class="second child">Second child</div>
</div>
<div class="third child">Third child</div>

<style>
.parent {
  --background-color: black;

  color: white;
}

.first {
  --background-color: purple;
}

.child {
  background: var(--background-color, blue);
}
</style>
```

- Los dos primeros elementos `.child` tomarán color negro, ya que se le aplica a `.parent` (e hijos).
- El primer elemento `.child` se sobrescribe con color púrpura, ya que se le aplica a `.first`.
- El tercer elemento no tendrá ninguna variable definida, por lo que tomará color azul.

Esto nos permite mucha flexibilidad y potencia a la hora de utilizar `custom properties` en CSS, ya que son tremendamente útiles y versátiles, permitiéndonos utilizar incluso la cascada de CSS a nuestro favor.

Unidades del viewport

Con estas unidades podemos hacer referencia a un porcentaje concreto del tamaño específico que tengamos en la ventana del navegador, es decir, cada vez que hagamos referencia a una unidad precedida por v estamos haciendo referencia a un % del tamaño total de la ventana. De esta forma, podemos hacer que elementos concretos tengan valores proporcionales al tamaño de la ventana del navegador.

Unidad	Significado	Media aproximada
`vw`	viewport width	`1vw` = 1% del ancho del navegador
`vh`	viewport height	`1vh` = 1% del alto del navegador
`vmin`	viewport minimum	`1vmin` = 1% del alto o ancho (el mínimo)
`vmax`	viewport maximum	`1vmax` = 1% del alto o ancho (el máximo)
`vi`	viewport inline	Versión lógica inline de `vw` y/o `vh`
`vb`	viewport block	Versión lógica en bloque de `vw` y/o `vh`

https://lenguajecss.com/css/unidades-css/relativas-viewport/#la-unidad-vw-viewport-width

Funciones matemáticas

Son funciones de apoyo que podemos utilizar en CSS para realizar cálculos u operaciones sencillas de una forma fácil y sencilla, sin tener que abandonar CSS y/o recurrir a Javascript.

Función `calc()`

Permite calcular operaciones con unidades CSS como px, %, vw, vh u otras. se pueden usar operaciones como sumas (+), restas (-), multiplicaciones (*) o divisiones (/) que utilicen alguna de las unidades soportadas por CSS, como por ejemplo, números, dimensiones, porcentajes, tiempos, ángulos, etc...

```css
.elemento {
  width: calc(200px + 1em);
  height: 300px;
  background: blue;
}
```

```css
:root {
  --width: 550px;
}

header {
  --header-width: calc(var(--width, 500px) / 2);

  width: var(--header-width);
}
```

Funciones matemáticas

Función `min()` / `max()`

Permite calcular el valor mínimo o máximo de las unidades indicadas. En algunas ocasiones, necesitaremos que en una propiedad se aplique un valor entre varias posibilidades, y lo que te interesa es el valor más pequeño o el mayor.

CSS permite realizar esto utilizando la función `min()` que nos permite elegir el valor más pequeño de 2 o más posibilidades aplicadas por parámetro, y la función `max()` el mayor de los dos.

```css
.elemento {
  width: min(200px, 25%);
  height: 200px;
  background: red;
}
```

```css
.elemento {
  width: max(200px - 100px, 25%, 14vw);
  height: 200px;
  background: red;
}
```

Funciones matemáticas

Función `clamp()`

Permite calcular valores «ajustados». Así podemos realizar, de forma concisa la operación `max(MIN, min(VAL, MAX))` y crear valores flexibles pero con un mínimo y un máximo establecido.

Hay dos líneas con un `width` que son equivalentes. El primer valor de `clamp()` sería el valor mínimo, mientras que el tercero es el valor máximo. De esta forma, el navegador realiza lo siguiente:

- Obtiene el valor mínimo entre el segundo y tercer parámetro.
- Obtiene el valor máximo entre el primer parámetro y el resultado anterior.
- Utiliza el resultado de la operación anterior en el `width`.

```css
.element {
  width: max(100px, min(25%, 300px));
  width: clamp(100px, 25%, 300px);
  height: 200px;
  background: red;
}
```

Muy útil para calcular tipografías fluidas
https://www.fluid-type-scale.com/

Bordes con imágenes

Si queremos hacer algo un poco más complejo con los bordes, CSS3 incorporó en su momento un sistema para crear bordes extensibles basados en una imagen de molde.

Propiedad	Descripción
`border-image-width`	Especifica el grosor de la imagen utilizada para el borde.
`border-image-source`	Imagen a utilizar para los bordes con imágenes mediante la técnica 9-slice.
`border-image-slice`	Distancia donde se hace el punto de corte en la imagen.
`border-image-outset`	Distancia donde se hace el punto de corte en la imagen.

https://lenguajecss.com/css/modelo-de-cajas/bordes-imagenes/

Gradientes lineales

El gradiente lineal permite crear fondos degradados que van en una dirección y cambian de un color a otro, dependiendo de la lista de colores indicada por parámetro.

Propiedad	Descripción
`linear-gradient(color, color)`	Gradiente de colores (hacia abajo).
`linear-gradient(direction/angle, color, color)`	Gradiente con dirección específica.
`linear-gradient(direction/angle, color size, color size)`	Gradiente indicando donde comienza a cambiar el color.
`linear-gradient(direction/angle, color size size, color size size)`	Gradiente indicando inicio y fin de cada color.
`repeating-linear-gradient()`	Gradiente estableciendo un patrón repetitivo

https://lenguajecss.com/css/gradientes/linear-gradient/#linear-gradient

Imágenes y objetos

Al insertar en un documento HTML algunos objetos como imágenes a través de la etiqueta `<img>`, elementos multimedia a través de `<video>` u otros como `<textarea>` o `<input>`, dichos elementos tienen su propia forma de mostrarse en pantalla ya que tienen características ajenas a CSS. Por ello tienen un tamaño inicial que no encaja con el uso que queremos darle, no se adapta a las cajas o contenedores que usamos o no funciona como tenemos previsto que lo haga.

Propiedad	Valores	Descripción
`object-fit`	`fill | contain | cover | none | scale-down`	Modo de relleno del elemento.
`object-position`	`PERCENT PERCENT`	Posición (x,y) del elemento.
`object-view-box`	`SHAPE`	Región del elemento se visualizará.
`image-rendering`	`auto | smooth | high-quality | crisp-edges | pixelated`	Algoritmo de escalado a utilizar.
`image-orientation`	`from-image | none | ANGLE [flip]`	Orientación de la imagen.
`aspect-ratio`	`auto | NUMBER / NUMBER | NUMBER`	Define la proporción de aspecto.

https://lenguajecss.com/css/imagenes-y-fondos/imagenes-css/#la-propiedad-object-fit

Combinadores lógicos

En ciertas situaciones, es posible que queramos crear grupos con diferentes selectores con el objetivo de escribir menos código, o reutilizar bloques de código CSS en más situaciones, de forma que sean más potentes y flexibles.

La forma más sencilla de conseguir esto es separando selectores por comas, pero también disponemos de combinadores lógicos que nos permiten seleccionar elementos con ciertas restricciones. Estos combinadores funcionan como una pseudoclase, sólo que se le pueden pasar parámetros, ya que son de tipo pseudoclase funcional.

Selector	Descripción
`div, button, p`	Agrupaciones. Seleccionamos varios elementos separándolos por comas.
`:is()`	Agrupaciones. Idem al anterior, pero permite combinar con otros selectores.
`:where()`	Agrupaciones. Idem al anterior, pero con menor especificidad CSS.
`:has()`	Permite seleccionar elementos padre que tengan ciertas características en sus hijos.
`:not()`	Permite seleccionar elementos que no cumplan ciertas características.

Combinadores `:is()` y `:where()`

Son un reemplazo práctico de la agrupación de selectores mediante comas, que permite reescribir selectores complejos de una forma mucho más práctica y compacta, ya que permite combinar y acumular con otros selectores anteriores o posteriores.

```css
.container .list,   /* Especificidad (0,2,0) */
.container .menu,   /* Especificidad (0,2,0) */
.container ul {     /* Especificidad (0,1,1) */
  /* ... */
}

.container :is(.list, .element, .menu) {  /* Especificidad (0,2,0) */
  /* ... */
}

.container :where(.list, .element, .menu) {  /* Especificidad (0,1,0) */
  /* ... */
}
```

Mucho cuidado con la especificidad CSS, ya que no es igual. Con la pseudoclase `:is()`, se calcula la especificidad sumando la especificidad más alta de sus parámetros. Mientras que con `:is()`, la especificidad es el valor más alto de la lista de parámetros, en el caso de `:where()` es siempre cero.

Combinador `:has()`

La pseudoclase `:has()` permite seleccionar el elemento precedido, si sus elementos hijos cumplen los criterios indicados por parámetro. Esto puede resultar un poco confuso, pero se ve claramente en el siguiente ejemplo:

En este caso, la propiedad `text-decoration: none` se aplica sobre el enlace `<a>`, sólo si en el interior del enlace existe una etiqueta `<img>`. Este ejemplo podría ser muy útil para eliminar estilos sobre imágenes que son enlaces.

```
a:has(> img) {
  text-decoration: none;
}
```

Algunos detalles interesantes sobre la pseudoclase funcional `:has()`:

- La pseudoclase `:has()` no se puede anidar dentro de otra `:has()`.
- Los pseudoelementos como `::before` o `::after` no funcionan dentro de `:has()`.
- La especificidad de `:has()` es el valor más alto de los selectores indicados por parámetro.

Combinador `:not()`

La pseudoclase de negación `:not()` es muy útil, ya que permite seleccionar todos los elementos que no cumplan los criterios indicados en sus parámetros entre paréntesis.

Este pequeño fragmento de código nos indica que todos los párrafos `<p>` que no pertenezcan a la clase `.general`, se les aplique el estilo especificado.

```css
p:not(.general) {
  border: 1px solid #DDD;
  padding: 8px;
  background: #FFF;
}
```

Algunos detalles adicionales sobre la pseudoclase funcional `:not()`:

- Se puede indicar una lista de criterios por parámetro, y no uno solo (como en el ejemplo anterior).
- La pseudoclase `:not()` no acepta pseudoelementos como `::before` o `::after` por parámetro.
- Al igual que con `:is()` y `:has()`, la especificidad de `:not()` es el valor más alto de sus parámetros.

Pseudoclases de formularios

Estas pseudoclases permiten seleccionar elementos para darle estilo dependiendo de temas relacionados con los formularios o los campos que están en el interior de un formulario.

Esencialmente, existen tres categorías:

- Pseudoclases de interacción: Seleccionar elementos cuando cambia el estado de un elemento.

Pseudoclase	Descripción
`:checked`	Selecciona el elemento cuando el campo está seleccionado.
`:indeterminade`	Selecciona el elemento cuando la casilla está en un estado indeterminado.

https://lenguajecss.com/css/selectores/pseudoclases-formularios/#pseudoclases-de-interacci%C3%B3n

Pseudoclases de formularios

- Pseudoclases de estado: Seleccionar elementos cuando se encuentran en un estado concreto.

Pseudoclase	Descripción
`:enabled`	Selecciona cuando el campo del formulario está activado.
`:disabled`	Selecciona cuando el campo del formulario está desactivado.
`:read-only`	Selecciona cuando el campo es de sólo lectura.
`:read-write`	Selecciona cuando el campo es editable por el usuario.
`:placeholder-shown`	Selecciona cuando el campo está mostrando un placeholder.
`:default`	Selecciona cuando el elemento tiene el valor por defecto.

https://lenguajecss.com/css/selectores/pseudoclases-formularios/#pseudoclases-de-estado

- Pseudoclases de validación: Seleccionar elementos si cumplen o no un cierto criterio de validación.

Pseudoclases de formularios

- Pseudoclases de validación: Seleccionar elementos si cumplen o no un criterio de validación.

Pseudoclase	Descripción
`:required`	Cuando el campo es obligatorio, o sea, tiene el atributo `required`.
`:optional`	Cuando el campo es opcional (por defecto, todos los campos).
`:valid`	Cuando los campos cumplen la validación HTML5.
`:invalid`	Cuando los campos no cumplen la validación HTML5.
`:user-valid`	Idem a `:valid`, pero cuando el usuario ha interactuado.
`:user-invalid`	Idem a `:invalid`, pero cuando el usuario ha interactuado.
`:in-range`	Cuando los campos numéricos están dentro del rango.
`:out-of-range`	Cuando los campos numéricos están fuera del rango.

https://lenguajecss.com/css/selectores/pseudoclases-formularios/#pseudoclases-de-validaci%C3%B3n

Pseudoelementos CSS

Los pseudoelementos permiten seleccionar y dar estilo a elementos que no existen en el HTML, o que no son un elemento en sí.

La sintaxis está precedida de dos puntos dobles (`::`) para diferenciarlos de las pseudoclases, que tienen dos puntos (`:`). Este cambio surgió posteriormente, por lo que aún es frecuente ver fragmentos de código con pseudoelementos con la sintaxis de pseudoclase con un solo par de puntos `:`.

Quizás los pseudoelementos más conocidos sean `::before` y `::after`, ambos necesitan de la propiedad `content` para poder mostrar contenido antes o después del elemento indicado.

Elemento	Descripción
`content`	Propiedad para generar contenido. Sólo usable en `::before` o `::after`.
`::before`	Aplica los estilos antes del elemento indicado.
`::after`	Aplica los estilos después del elemento indicado.
`attr()`	Función que hace referencia al atributo HTML indicado.

La propiedad `content`

La propiedad `content` admite parámetros de diverso tipo, incluso concatenando información mediante espacios. Podemos utilizar varios tipos de contenido:

Valor	Descripción	Ejemplo
`STRING “...”`	Añade el contenido de texto indicado.	`content: "Contenido:";`
`attr(atributo)`	Añade el valor del atributo HTML indicado.	`content: attr(href);`
`IMAGE url()`	Añade la imagen indicada en la URL.	`content: url(icon.png);`
`GRADIENT`	Añade un gradiente del tamaño indicado.	`content: linear-gradient(red, blue);`
`COUNTER`	Define un contador CSS para mostrar.	`content: counter(item);`

Ten en cuenta que la propiedad `content` sólo funciona dentro de los pseudoelementos `::before` y `::after`. No puede utilizarse en otro lugar.

Pseudoelemento `::before`

El navegador se encargará de añadir contenido antes del inicio de la etiqueta de apertura del elemento que has seleccionado con el resto del selector:

```css
q::before {
  content: "«";
  color: #888;
}
```

En este caso, estamos añadiendo el texto « justo antes de los elementos `<q>` que aparezcan en nuestro documento, además de pintarlos de color gris. De esta forma, podemos generar información (usualmente con fines decorativos) que no existe en el HTML, pero que por circunstancias de diseño sería más apropiado colocar en el código CSS.

Pseudoelemento `::after`

El pseudoelemento ::after, que permitirá añadir contenido después de la etiqueta de cierre en cuestión. El código correspondiente sería el siguiente:

```
q::after {
  content: "»";
  color: #888;
}
```

Los ejemplos anteriores unidos, permitirían insertar el carácter « antes de las citas indicadas con el elemento HTML <q> y el carácter » al finalizar la misma, ambas de color gris.

Función `attr()`

Esta función se puede utilizar en la propiedad `content` para recuperar el valor del atributo HTML especificado en A. Veamos un ejemplo para clarificarlo, concatenándolo con un texto aplicado en la propiedad `content`:

```
a::after {
  content: " ( " attr(href) " )";
}
```

A todo los enlaces `<a>`, muestra un contenido de texto posterior (`::after`) donde, envuelto entre paréntesis, escribe el contenido del atributo `href` del enlace tratado. Esto puede ser realmente útil en una página de estilos que se aplica en el momento de imprimir, en los cuales se pierde la información del enlace al no ser un medio interactivo.

También se puede utilizar la función `url()` para añadir una imagen al contenido generado, tal y como lo hacemos en la propiedad `background`, por ejemplo.

Otros pseudoelementos

También tenemos pseudolementos tipográficos, de resaltado y otros más específicos.

Pseudolemento	Descripción
`::first-letter`	Aplica los estilos en la primera letra del texto.
`::first-line`	Aplica los estilos en la primera línea del texto.
`::selection`	Aplica estilos al fragmento de texto seleccionado por el usuario.
`::target-text`	Aplica estilos al fragmento de texto enlazado tras el ancla de la URL.
`::spelling-error`	Aplica estilos al fragmento de texto resaltado por error ortográfico.
`::grammar-error`	Aplica estilos al fragmento de texto resaltado por error gramatical.
`::marker`	Aplica estilos a los elementos de cada ítem de una lista.
`::backdrop`	Aplica estilos al fondo exterior de un elemento en primer plano (sin que afecte a este).
`::placeholder`	Aplica estilos a los textos de sugerencia de los campos `<input>`.
`::file-selector-button`	Aplica estilos a los botones de campo `<input>` de subir archivos.

https://lenguajecss.com/css/selectores/pseudoelementos/#pseudoelementos-tipogr%C3%A1ficos

¿Qué son las reglas CSS?

Son un tipo de declaración especial que permite indicar comportamientos especiales en muchos contextos. Su sintaxis suele determinarse incluyendo una palabra clave que comienza por `@` (como por ejemplo `@media` o `@import`) y dependiendo de la regla en cuestión, puede tener una sintaxis u otra.

En general, se pueden colocar en cualquier parte de la hoja de estilos, tanto al principio, al medio como al final. Sin embargo, algunas reglas tienen ciertas restricciones. Por ejemplo, como comentaremos, en el caso de la regla `@import` debe colocarse en las primeras líneas del fichero.

https://lenguajecss.com/css/reglas-css/que-son-reglas-css/#reglas-css

La regla `@media`

Las reglas `media queries` son un tipo de reglas de CSS que permiten crear un bloque de código que sólo se procesa en los dispositivos que cumplan los criterios especificados como condición.

Regla	Descripción
`@media (<condición>)`	Si se cumple la condición, se aplican los estilos de su interior.
`@media not (<condición>)`	Si no se cumple la condición, se aplican los estilos de su interior.

En algunas ocasiones, queremos indicar que las reglas `@media` sólo se pongan en funcionamiento en determinados tipos de dispositivos. Son los llamados tipos de medios. Existen los siguientes:

Tipo de medio	Significado
`all`	Todos los dispositivos o medios. El que se utiliza por defecto.
`screen`	Monitores o pantallas de ordenador. Es el más común.
`print`	Documentos de medios impresos o pantallas de previsualización de impresión.
`speech`	Lectores de texto para invidentes.

Condiciones de Media Queries

Tipo de característica	Valores	¿Cuándo se aplica?
`width`	`SIZE`	Si el dispositivo tiene el tamaño de ancho exactamente.
`min-width`	`SIZE`	Si el dispositivo tiene un tamaño de ancho mayor al indicado.
`max-width`	`SIZE`	Si el dispositivo tiene un tamaño de ancho menor al indicado.
`height`	`SIZE`	Si el dispositivo tiene el tamaño de alto exactamente.
`min-height`	`SIZE`	Si el dispositivo tiene un tamaño de alto mayor al indicado.
`max-height`	`SIZE`	Si el dispositivo tiene un tamaño de alto menor al indicado.
`aspect-ratio`	`aspect-ratio`	Si el dispositivo encaja con la proporción de aspecto indicada.
`orientation`	`landscape | portrait`	Si el dispositivo está en colocado en modo vertical o apaisado.

https://lenguajecss.com/css/responsive-web-design/media-queries/#condiciones-de-media-queries

La regla `@import`

Es una regla de CSS que permite cargar un fichero .css externo, leer sus líneas de código e incorporarlo al archivo actual. Estas reglas CSS se suelen indicar en las primeras líneas del fichero, ya que deben figurar antes de otras reglas CSS o contenido CSS similar.

Tipo de importación	Descripción
`@import URL`	Importamos una hoja de estilos CSS externa.
`@import URL media query`	Importamos una hoja de estilos CSS sólo si coincide con el `media query` indicado.
`@import URL supports(`condición`)`	Importamos una hoja de estilos CSS sólo si el navegador soporta la condición.
`@import URL layer(`nombre`)`	Importamos una hoja de estilos CSS y la colocamos en la capa nombre.
`@import URL layer()`	Importamos una hoja de estilos CSS y la colocamos en una nueva capa anónima.

Un detalle muy importante es que hay que tener en cuenta que la regla `@import` se evalúa en el navegador a la hora de cargar la página, por lo que cada regla `@import` equivale a una petición al servidor de un nuevo archivo .css.

Formato de `@import`

En principio, existen dos formas de cargar ficheros externos mediante la regla `@import`: utilizando la función `url()` o indicando simplemente un con el archivo o dirección URL:

Formato	Descripción
`@import url("fichero.css")`	Importa el fichero.css utilizando la función `url()` de CSS.
`@import "fichero.css"`	Importa el fichero.css utilizando un STRING

También podemos indicar tras la URL, una media query que nos permitirá que esa hoja de estilos externa se descargue sólo si estamos en un navegador que cumple las condiciones de la media query. Por otro lado, podemos combinar la regla `@supports` con la regla `@import` y establecer condiciones específicas.

```
@import url("mobile.css") screen and (max-width: 640px);
@import url("print.css") print;

@import url("flex-fallback.css") supports(not (display: grid));
@supports (display: grid) {
  /* ... */
}
```

La regla `@font-face`

Para solucionar el problema de utilizar una tipografía que el usuario no tiene instalada en su sistema, utilizaremos la regla `@font-face` de CSS. Dicha regla nos permite descargar en el navegador una tipografía desde una página y utilizarla de forma transparente al usuario.

Valor	Significado
`local("Nombre")`	¿Está la fuente instalada? Si es así, no hace falta descargarla, la usa.
`url("file.woff2")`	Formato Web Open Font Format 2. Mejora de WOFF con Brotli.
`url("file.woff")`	Formato Web Open Font Format. Es un TTF comprimido, ideal para web.
`url("file.ttf")`	Formato True Type / Open Type. .ttf o .otf. Para soportar navegadores antiguos.

https://lenguajecss.com/css/fuentes-y-tipografias/regla-font-face/#la-regla-font-face

Tipografías de Google Fonts

En la actualidad, es muy común utilizar Google Fonts como repositorio proveedor de tipografías para utilizar en nuestros sitios web por varias razones:

- Gratuitas: Disponen de un amplio catálogo de fuentes y tipografías libres y/o gratuitas.
- Cómodo: Resulta muy sencillo su uso: Google nos proporciona un código y el resto lo hace él.
- Rápido: El servicio está muy extendido y utiliza sus propios servidores.

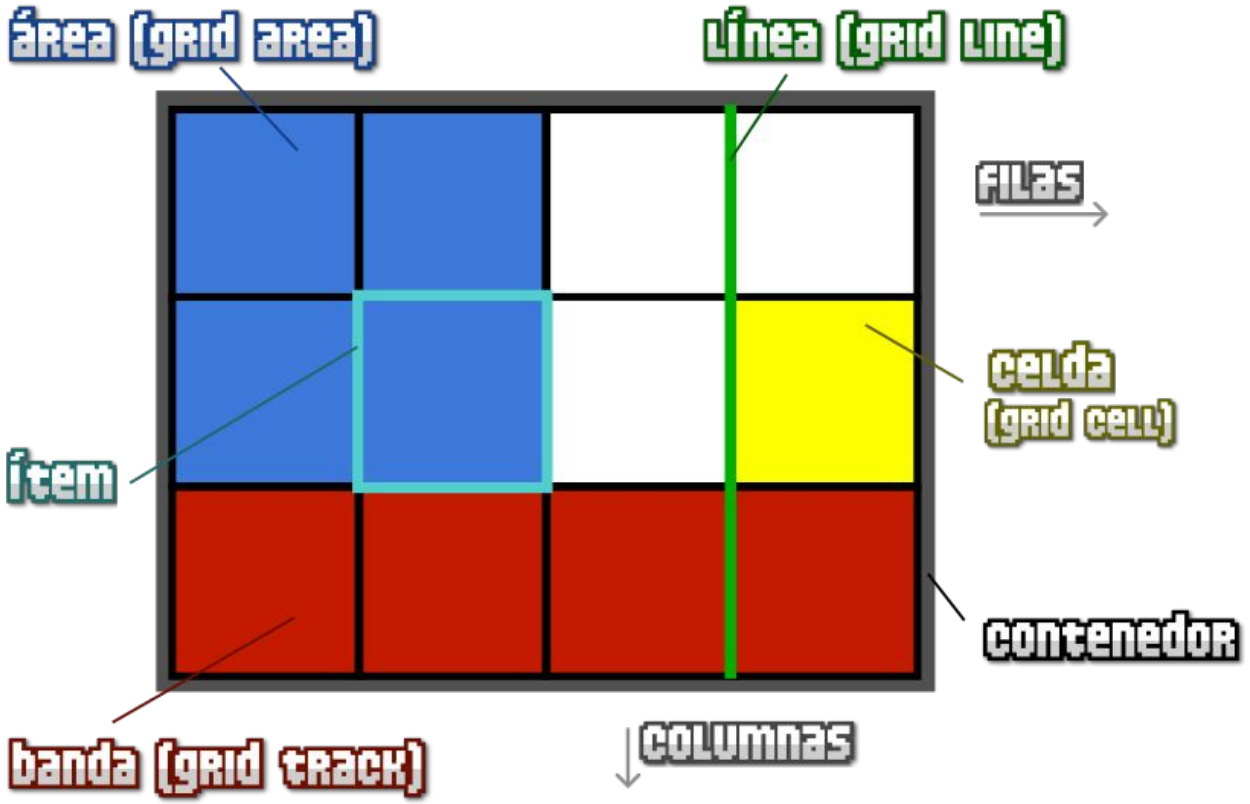
En la propia página de Google Fonts podemos seleccionar las fuentes con las características deseadas y generar un código HTML con la tipografía (o colección de tipografías) que vamos a utilizar.

https://lenguajecss.com/css/fuentes-y-tipografias/tipografias-google-fonts/#seleccionar-la-tipograf%C3%ADa

Grid CSS - https://lenguajecss.com/css/maquetacion-y-colocacion/grid-css/

Grid CSS nace de la necesidad de maquetar siguiendo una cuadrícula como hacían muchos frameworks, añadiéndole numerosas mejoras y características que permiten crear rápidamente cuadrículas flexibles y potentes de forma prácticamente instantánea con una nueva familia de propiedades CSS.

- **Contenedor**: El elemento padre contenedor que definirá la cuadrícula o rejilla.
- **Ítem**: Cada uno de los hijos que contiene la cuadrícula (elemento contenedor).
- **Celda (grid cell)**: Cada uno de los cuadritos (unidad mínima) de la cuadrícula.
- **Area (grid area)**: Región o conjunto de celdas de la cuadrícula.
- **Banda (grid track)**: Banda horizontal o vertical de celdas de la cuadrícula.
- **Línea (grid line)**: Separador horizontal o vertical de las celdas de la cuadrícula.



Definir filas y columnas fijas

En Grid CSS, la forma principal de definir una cuadrícula es indicar el tamaño de sus filas y sus columnas de forma explícita. Para ello, sólo tenemos que usar las propiedades CSS `grid-template-columns` y `grid-template-rows`:

Propiedad	Valor	Descripción
`grid-template-columns`	[col1][col2]...	Establece el SIZE de cada columna (col 1, col 2...).
`grid-template-rows`	[fila1][fila2]...	Establece el SIZE de cada fila (fila 1, fila 2...).

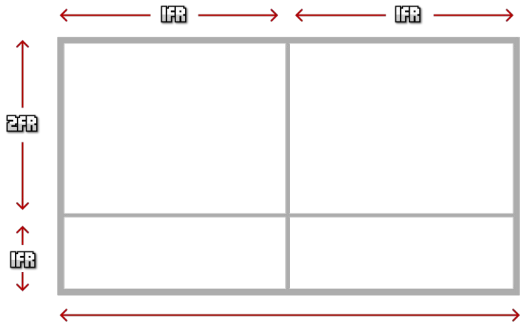
Tenemos que vigilar que el número de elementos hijos, ya que dependiendo del número de elementos hijos que tengamos definidos, tendremos una cuadrícula de 2x2 elementos (4 ítems), 2x3 elementos (6 ítems), 2x4 elementos (8 ítems) y así sucesivamente. Incluso, si el número de ítems es impar (por ejemplo, 5 ítems), la última celda de la cuadrícula se quedaría vacía.

A medida que vamos incluyendo más ítems, podemos aumentar el número de parámetros en `grid-template-columns` y/o `grid-template-rows`. En caso de tener más ítems estos se incluirían sin formato. De tener menos, simplemente se ocuparían los ítems implicados.

Definir filas y columnas. Unidad fracción restante (`fr`)

También podemos utilizar otras unidades (o incluso combinarlas): porcentajes, la palabra clave `auto` (que obtiene el tamaño restante) o la unidad especial de grid `fr` (fracción restante), que explicaremos a continuación.

```css
.grid {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: 2fr 1fr;
}
```



Este nuevo ejemplo, también crea una cuadrícula de 2x2, donde el tamaño de la cuadrícula se divide en:

- Dos columnas: Mismo tamaño de ancho para cada una.
- Dos filas: La primera fila ocupará el doble (`2fr`) que la segunda fila (`1fr`).

De esta forma, es muy fácil predecir el espacio que va a ocupar la cuadrícula, ya que sólo tenemos que sumar todas las unidades para saber el tamaño total, y comparar con cada columna o fila para saber cómo de grande o pequeña es respecto al total. Así tendremos un mejor control del espacio restante de la cuadrícula, y resultará más intuitivo calcularlo.

Definir filas y columnas. Filas y columnas repetitivas

En algunos casos, en las propiedades `grid-template-columns` y `grid-template-rows` podemos necesitar indicar las mismas cantidades un número alto de veces, resultando repetitivo y molesto escribirlas varias veces. Se puede utilizar la función `repeat()` para indicar repetición de valores, señalando el número de veces que se repiten y el tamaño en cuestión.

La expresión a utilizar sería la siguiente: `repeat(número de veces, tamaño)`:

```css
.grid {
  display: grid;

  grid-template-columns: 100px repeat(4, 50px) 200px;
  grid-template-rows: repeat(2, 1fr 2fr);

  /* Equivalente a... */
  grid-template-columns: 100px 50px 50px 50px 50px 200px;
  grid-template-rows: 1fr 2fr 1fr 2fr;
}
```

Definir filas y columnas. Función `minmax()`

La función `minmax()` se puede utilizar como valor para definir rangos flexibles de celda. Funciona de la siguiente forma: `minmax(min, max)` definiendo un rango entre un mínimo y un máximo.

```
<div class="container">
  <div class="item item-1">Item 1</div>
  <div class="item item-2">Item 2</div>
  <div class="item item-3">Item 3</div>
  <div class="item item-4">Item 4</div>
</div>

<style>
.container {
  display: grid;
  grid-template-columns: repeat(2, minmax(400px, 600px));
  grid-template-rows: repeat(2, 1fr);
  gap: 5px;
}

.item {
  background: black;
  color: white;
  padding: 1em;
}
</style>
```

Definir filas y columnas. `Auto-fill` y `Auto-fit`

En la función `repeat()` es posible utilizar las palabras claves `auto-fill` o `auto-fit` para indicar al navegador que queremos que rellene o ajuste el contenedor `grid` con múltiples elementos hijos dependiendo del tamaño del `viewport` (región visible del navegador).

Es decir, si utilizamos `repeat(auto-fill, minmax(300px, 1fr))`, el navegador se va a encargar de que los elementos hijos con el tamaño mínimo quepan en la primera fila, y los que no quepan, se desplacen a las siguientes filas del `grid`, de modo que se aproveche lo mejor posible el contenedor, y consigamos un efecto similar al que se consigue con `media queries`, pero de una forma más directa y con menos código.

```css
.grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
  background: grey;
  gap: 10px;
}

.item {
  background: blue;
  color: #fff;
  font-size: 2rem;
}
```

Huecos en grid (`gap`)

Por defecto, la cuadrícula tiene todas sus celdas pegadas a sus celdas contiguas. Aunque sería posible darle un `margin` a las celdas dentro del contenedor, existe una forma más apropiada: los huecos (`gutters`).

Para especificar los huecos (espacio entre celdas) podemos utilizar las propiedades `column-gap` y/o `row-gap`. En ellas indicaremos el tamaño de dichos huecos:

Propiedad	Descripción
`column-gap`	Establece el SIZE de los huecos entre columnas (líneas verticales).
`row-gap`	Establece el SIZE de los huecos entre filas (líneas horizontales).
`gap`	Establece el SIZE de los huecos de `row-gap` y `column-gap`.

Alinear elementos

En `grid` tenemos unas propiedades que nos permiten colocar y ajustar nuestra cuadrícula `grid` o los ítems a lo largo de ella. Estas propiedades están basadas en `CSS flex`, sin embargo, en `grid` pueden tener un comportamiento diferente.

Propiedad	Descripción	Eje
`justify-items`	Alinea los elementos (hijos) en horizontal (eje principal) dentro de cada celda.	Elementos eje principal
`align-items`	Alinea los elementos (hijos) en vertical (eje principal) dentro de cada celda.	Elementos eje secundario
`justify-content`	Alinea el contenido (la cuadrícula) en horizontal (eje secundario) en el contenedor padre.	Contenido eje principal
`align-content`	Alinea el contenido (la cuadrícula) en vertical (eje secundario) en el contenedor padre.	Contenido eje secundario

Alinear elementos . La propiedad `justify-items`

Sirve para colocar los ítems de un contenedor grid a lo largo de sus celdas correspondientes, siempre en el eje principal (por defecto, en horizontal). Los valores que puede tomar esta propiedad son los siguientes:

Propiedad	Descripción
`start`	Coloca cada ítem al **inicio** de su celda en el eje principal.
`end`	Coloca cada ítem al **final** de su celda en el eje principal.
`center`	Coloca cada ítem en el **centro** de su celda en el eje principal.
`stretch (default)`	Hace que cada ítem **se estire** y ocupe todo el espacio disponible de su celda en el eje principal.

Alinear elementos. La propiedad `align-items`

Sirve para colocar los ítems de un contenedor grid a lo largo de sus celdas correspondientes, pero en lugar de el eje principal, las coloca en el eje secundario (por defecto, en vertical). Los valores que puede tomar son los mismos que la propiedad anterior.

Propiedad	Descripción
`start`	Coloca cada ítem al **inicio** de su celda en el eje secundario.
`end`	Coloca cada ítem al **final** de su celda en el eje secundario.
`center`	Coloca cada ítem en el **centro** de su celda en el eje secundario.
`stretch (default)`	Hace que cada ítem **se estire** y ocupe todo el espacio disponible de su celda en el eje secundario.

Alinear elementos. La propiedad `justify-content`

Permite modificar la distribución del contenido de la cuadrícula en su contenedor padre, a lo largo de su eje principal (por defecto, el horizontal). Los valores que puede tomar son los siguientes:

Propiedad	Descripción
`start`	Coloca la cuadrícula en su conjunto al **inicio** del contenedor padre en su eje principal (horizontal).
`end`	Coloca la cuadrícula en su conjunto al **final** del contenedor padre en su eje principal (horizontal).
`center`	Coloca la cuadrícula en su conjunto al **centro** del contenedor padre en su eje principal (horizontal).
`stretch (default)`	**Estira** la cuadrícula ocupando todo el espacio disponible del contenedor padre en su eje principal (horizontal).
`space-between`	Establece **espacios sólo entre las celdas**, en su eje principal (horizontal).
`space-around`	Establece **espacios alrededor de las celdas**, en su eje principal (horizontal).
`space-evenly`	Idem al anterior, pero **solapando los espacios**, de modo que sean todos de tamaño equivalente.

Alinear elementos. La propiedad `align-content`

Sirve para colocar el contenido de la cuadrícula en su contenedor padre, pero a lo largo de su contenedor secundario (por defecto, el vertical). Los valores que puede tomar son exactamente los mismos que la propiedad anterior.

Propiedad	Descripción
`start`	Coloca la cuadrícula en su conjunto al **inicio** del contenedor padre en su eje secundario (vertical).
`end`	Coloca la cuadrícula en su conjunto al **final** del contenedor padre en su eje secundario (vertical).
`center`	Coloca la cuadrícula en su conjunto al **centro** del contenedor padre en su eje secundario (vertical).
`stretch (default)`	**Estira** la cuadrícula ocupando todo el espacio disponible del contenedor padre en su eje secundario (vertical).
`space-between`	Establece **espacios sólo entre las celdas**, en su eje secundario (vertical).
`space-around`	Establece **espacios alrededor de las celdas**, en su eje secundario (vertical).
`space-evenly`	Idem al anterior, pero **solapando los espacios**, de modo que sean todos de tamaño equivalente.

Alineaciones específicas

En el caso de que queramos que uno de los ítems hijos tenga una distribución diferente al resto, podemos aplicar en el elemento hijo la propiedad `justify-self` (eje principal) o `align-self` (eje secundario) sobreescribiendo su distribución su general, y aplicando una específica.

Propiedad	Descripción
`justify-self`	Altera la alineación del ítem hijo en el eje horizontal y la sobreescribe con la indicada.
`align-self`	Altera la alineación del ítem hijo en el eje vertical y la sobreescribe con la indicada.

Recuerda que estas propiedades funcionan exactamente igual que sus análogas `justify-items` o `align-items` y tienen los mismos valores, sólo que en lugar de indicarse en el elemento padre contenedor, se hace sobre un elemento hijo y repercute en dicho elemento hijo específicamente.

Atajo alineaciones `Grid`

Si vamos a crear estructuras grid donde utilicemos los pares de propiedades `justify-items` y `align-items` por un lado, `justify-content` y `align-content` por otro, e incluso `justify-self` y `align-self` por otro, podemos utilizar las siguientes propiedades de atajo:

Propiedad	Valores	Descripción
`place-items`	`[align-items] [justify-items]`	Propiedad de atajo para `*-items`
`place-content`	`[align-content] [justify-content]`	Propiedad de atajo para `*-content`
`place-self`	`[align-self] [justify-self]`	Propiedad de atajo para `*-self`

Alinear elementos . La propiedad `Order`

Es una propiedad mediante la cual podemos modificar y establecer un orden de los elementos mediante números que actuarán como «peso» del elemento:

Propiedad	Descripción
`order`	Número (peso) que indica el orden de aparición de los ítems.

Por defecto, todos los elementos hijos de un contenedor `flex` tienen establecido un `order` por defecto al valor 0. Si indicamos una propiedad `order` con un valor numérico diferente, recolocará los ítems según dicho número, colocando antes los elementos con un número `order` más pequeño (incluso números negativos) y los elementos con números más altos después.

Grid por áreas

Mediante los Grids por área es posible indicar el nombre y posición concreta de cada área de una cuadrícula. Para ello utilizaremos la propiedad `grid-template-areas` en nuestro contenedor padre, donde debemos especificar el orden de las áreas en la cuadrícula. Posteriormente, en cada ítem hijo, utilizamos la propiedad `grid-area` para indicar el nombre del área del que se trata y que el navegador pueda identificarlas:

Propiedad	Descripción
`grid-template-areas`	Indica la disposición de las áreas en el grid. Cada texto entre comillas simboliza una fila.
`grid-area`	Indica el nombre del área. Se usa sobre ítems hijos del grid.

La propiedad `grid-template-areas`

La propiedad `grid-template-areas` es la propiedad principal de este sistema, y debe utilizarse en el contenedor padre `grid`.

Cada una de estas filas se definen como un donde indicaremos el nombre de un área que posteriormente definiremos en nuestro código CSS. Cada fila puede tener ninguna o varias áreas que habría que separar por espacio. A continuación veremos algunos ejemplos de los valores que podemos indicar en esta propiedad y su significado:

Propiedad	Descripción
`none`	Indica que no se creará ninguna plantilla de áreas.
`"head"`	Indica que se creará una fila de una columna con el área head.
`"head menu"`	Indica que se creará una fila de 2 columnas con el área head en una y el área menu en otra.
`"head head"`	Indica que se creará una fila de 2 columnas con el área head ocupando ambas.
`"."`	Indica que se colocará una celda sin nombre (nula) en esta posición.

La propiedad `grid-area`

Por otro lado, al utilizar la propiedad `grid-template-areas` y nombrar varias áreas en su valores, es necesario que dichas áreas estén definidas mediante la propiedad `grid-area` en sus elementos hijos. Recuerda no confundir nombre de área, con nombre de clase, puesto que no es lo mismo.

Esta propiedad permite nombrar un elemento del HTML con un nombre de área. Mucho cuidado, ya que este nombre no es un `string`, y por lo tanto no debe definirse entre comillas ". Estos nombres se utilizarán en la propiedad `grid-template-areas` para definir donde irán ubicados.

Propiedad	Descripción
`auto`	Coloca la celda en la próxima área vacía que se encuentre disponible.
`nombre`	Le da un nombre de área al elemento en cuestión.

Celdas irregulares

Hasta ahora los Grid que hemos podido definir tienen ciertos límites, sobre todo cuando queremos que una celda de la cuadrícula tengan una distribución peculiar, y hagan que la cuadrícula sea irregular. Para estos casos, tenemos las siguientes propiedades que se utilizan en los elementos hijos de la cuadrícula.

Propiedad	Descripción
`grid-column-start`	Indica en qué columna empezará el ítem de la cuadrícula.
`grid-column-end`	Indica en qué columna terminará el ítem de la cuadrícula.
`grid-row-start`	Indica en qué fila empezará el ítem de la cuadrícula.
`grid-row-end`	Indica en qué fila terminará el ítem de la cuadrícula.

Tenemos dos pares de propiedades, una con el prefijo `grid-column-` y otro par con el prefijo `grid-row-`. Luego, dentro de ambas, tenemos un sufijo `-start` para indicar dónde empieza y otra con un sufijo `-end` para indicar donde termina. Con dichas propiedades, le asignaremos al hijo la parte de la cuadrícula donde debe empezar y dónde terminar, creando un caso particular sobre el resto.

Celdas irregulares

En las propiedades anteriores podemos establecer diferentes valores para indicar dónde comienza o termina una celda irregular en nuestro grid de CSS:

Propiedad	Descripción
`auto`	No se indica ningún comportamiento particular. Se queda igual. Valor por defecto.
`NUMBER`	Indica la línea específica, es decir, la separación de columnas o filas.
`span NUMBER`	Indica hasta cuántas líneas debe llegar.
`LINENAME`	Idem al anterior, pero con líneas nombradas.
`span LINENAME`	Idem al anterior, indicando nombre de línea hasta donde llegar.
`LINENAME NUMBER`	Idem al anterior, pero busca el nombre que aparece por número vez.

Atajo: `grid-column` y `grid-row`

El módulo grid de CSS proporciona las propiedades de atajo grid-column y grid-row donde se nos permite escribir en un formato abreviado las cuatro propiedades anteriores:

Propiedad	Descripción
`grid-column`	Propiedad de atajo para utilizar `grid-column-start` y `grid-column-end`.
`grid-row`	Propiedad de atajo para utilizar `grid-row-start` y `grid-row-end`.

Atajo: La propiedad `grid-area`

La propiedad `grid-area` nos permite resumir las cuatro propiedades `grid-column-start`, `grid-column-end`, `grid-row-start` y `grid-row-end` en una sola.

Propiedad	Valor	Descripción
`grid-area`	`grid-row-start / grid-column-start / grid-row-end / grid-column-end`	Atajo para utilizar `grid-row` y `grid-column`

Líneas con nombre

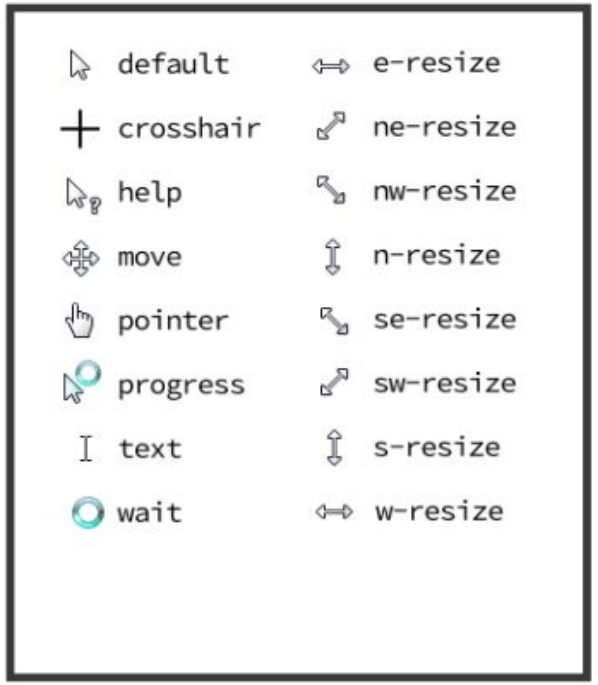
Hasta ahora hemos trabajado con Grid CSS hablando de celdas o casillas de una cuadrícula. Sin embargo, también podemos hablar de líneas, e incluso ponerles nombres y luego hacer referencia a ellas en ciertas propiedades.

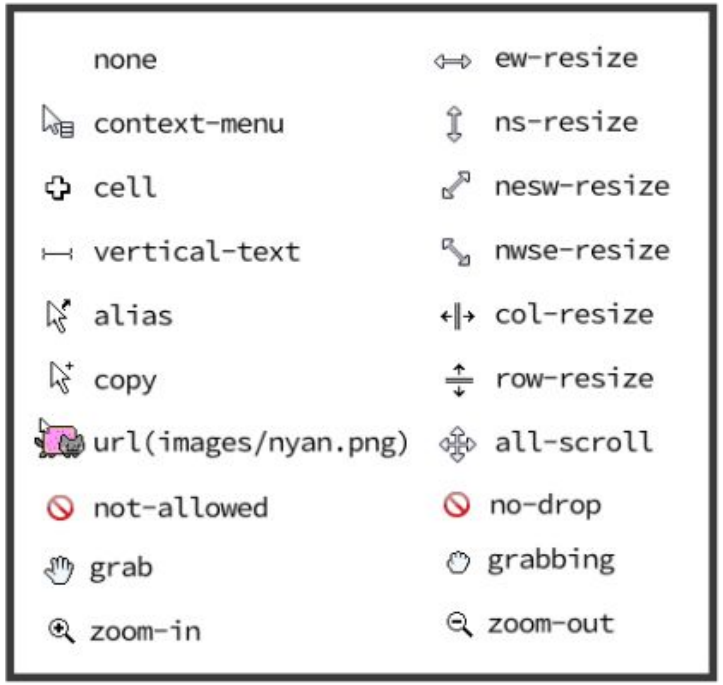
Propiedad	Valores	Descripción
`grid-template-columns`	[nombreLinea1] SIZE col1 [nombreLinea2] SIZE col2 ... [últimalinea]	Las líneas se definen entre corchetes.
`grid-template-rows`	[nombreLinea1] SIZE fila1 [nombreLinea] SIZE fila2 ... [últimalinea]	Las líneas se definen entre corchetes.

Cursor del ratón

Para modificar el cursor del ratón solo tenemos que especificar la propiedad `cursor` dentro del elemento que busquemos, junto al valor del cursor deseado:

Propiedad	Valores	Descripción
`cursor`	palabra clave de cursor	Muestra un cursor de ratón específico.





Propiedad `scroll-behavior`

La propiedad `scroll-behavior` nos permite activar un desplazamiento suave (`smooth scroll`) al pulsar en un enlace de anclas `<a>`. Como hemos dicho, por defecto, al pulsar sobre estas anclas, nos desplazamos directa e instantáneamente a nuestro objetivo.

Propiedad	Valores	Descripción
`scroll-behavior`	`auto | smooth`	Aplicado sobre `<html>`, suaviza ciertos desplazamientos de scroll.

La propiedad `overflow`

Si tenemos un elemento al que hemos forzado el tamaño de ancho y/o alto para que tenga un tamaño específico (que es más pequeño que su contenido), se va a producir el caso en el que su contenido no tiene suficiente espacio, por lo que se producirá un desbordamiento CSS. Para controlar estos casos, tenemos a nuestra disposición la propiedad `overflow` en CSS, donde podremos definir el comportamiento adecuado.

Propiedad	Valores	Descripción
`overflow-x`	`visible | hidden | scroll | auto`	Establece el desbordamiento del eje X (en horizontal).
`overflow-y`	`visible | hidden | scroll | auto`	Establece el desbordamiento del eje Y (en vertical).
`overflow`	`[overflow-x] [overflow-y]`	Propiedad de atajo que establece desbordamiento de ambos ejes.

Sombras de texto

Crear sombras en textos mediante CSS es muy útil, puesto que es una forma interesante de suavizar y hacer más legibles los textos, o simplemente hacerlos más atractivos para el usuario que está viendo la página. Para ello, utilizaremos la propiedad `text-shadow`. Esta propiedad permite los siguientes parámetros:

Propiedad	Valores	Descripción
`text-shadow`	`none`	No aplica ninguna sombra en el texto (o la quita si existía previamente).
`text-shadow`	`POSX POSY`	Aplica una sombra de color negro, desplazándola en horizontal y en vertical.
`text-shadow`	`POSX POSY SIZE`	Idem a la anterior, pero establece un desenfoque a la sombra (0 sin desenfoque).
`text-shadow`	`POSX POSY SIZE COLOR`	Idem al anterior, pero indicando un COLOR personalizado para la sombra.

https://lenguajecss.com/css/sombras/text-shadow/#sombras-de-texto

Sombras en elementos

Se denominan sombras sobre cajas a las sombras en CSS que se pueden crear en una etiqueta o elemento HTML. Para ello, se utiliza la propiedad `box-shadow`, que funciona de forma muy similar a la que vimos en las sombras de texto, sólo que con algunos añadidos interesantes.

Propiedad	Valores	Descripción
`box-shadow`	`none`	Elimina (o simplemente no establece) sombra sobre un elemento.
`box-shadow`	`POSX POSY`	Crea una sombra color negro desplazándola ligeramente en horizontal y/o vertical.
`box-shadow`	`POSX POSY SIZE`	Idem a la anterior, pero desenfocando o difuminando la sombra.
`box-shadow`	`POSX POSY SIZE SIZE`	Idem a la anterior, pero además aplicando un factor de crecimiento a la sombra.
`box-shadow`	`POSX POSY SIZE COLOR`	Idem a la anterior, cambiando el color de la sombra.
`box-shadow`	`POSX POSY SIZE COLOR inset`	Idem a la anterior, pero estableciendo una sombra interna en lugar de externa.

https://lenguajecss.com/css/sombras/box-shadow/

Sombras idénticas

Para crear sombras idénticas se debe utilizar la función `drop-shadow()` que tiene la misma sintaxis exacta de `text-shadow`. La diferencia entre `drop-shadow()` y `box-shadow()` es que, mientras la última realiza una sombra de la caja, la primera realizará una sombra idéntica «como una gota de agua» (drop).

Propiedad	Valores	Descripción
`drop-shadow`	`none`	No aplica ninguna sombra (o la quita si existía previamente).
`drop-shadow`	`POSX POSY`	Crea una sombra idéntica utilizando la propiedad filter.
`drop-shadow`	`POSX POSY SIZE`	Idem a la anterior, con un desplazamiento y además un desenfoque.
`drop-shadow`	`POSX POSY SIZE COLOR`	Idem a la anterior, pero indicando un color específico.

https://lenguajecss.com/css/sombras/drop-shadow/#la-funci%C3%B3n-drop-shadow

Filtros CSS

Los filtros CSS son una característica muy atractiva de CSS que permite aplicar ciertos efectos de imagen, propios de aplicaciones de retoque fotográfico, como sepia, variaciones de brillo o contraste (u otros) al vuelo en el propio navegador, sin hacer cambios permanentes sobre una imagen.

Los filtros de CSS nos proporcionan un amplio abanico de funciones, listas para utilizar mediante la propiedad filter y aplicarlas a los elementos que queramos de nuestra página. Estos filtros permiten alterar los colores, tonalidades o diferentes aspectos visuales, tal como se harían desde un programa de diseño gráfico:

https://lenguajecss.com/css/efectos/filtros-css/

Transiciones

Las transiciones nos permiten hacer efectos de transición para conseguir un efecto suavizado entre un estado inicial y un estado final al realizar una acción.

Propiedades	Descripción	Valor
`transition-property`	Propiedades CSS afectadas por la transición.	`all` | `none` | propiedad css
`transition-duration`	Tiempo de duración.	`0` | TIME
`transition-timing-function`	Ritmo de la transición.	Valor del ritmo
`transition-delay`	Tiempo de retardo inicial.	`0` | TIME

https://lenguajecss.com/css/animaciones/transiciones/#propiedades-de-transici%C3%B3n

Animaciones CSS

Las animaciones dan un paso más respecto a las transiciones, convirtiéndolo en algo mucho más flexible y potente, en el que no es necesario que el usuario interactúe. Las animaciones nos permiten añadir más estados aún, podemos partir desde un estado inicial, a un estado posterior, a otro estado posterior, y así sucesivamente.

Para crear animaciones CSS es necesario realizar 2 pasos:

- Utilizar la propiedad `animation` (o derivadas) para indicar que elemento HTML vamos a animar.
- Definir mediante la regla `@keyframes` la animación en cuestión y sus estados (fotogramas clave).

En primer lugar, vamos a examinar las diferentes propiedades relacionadas con las animaciones, y más adelante veremos cómo crear fotogramas con la regla `@keyframes`.

Propiedades de animación CSS

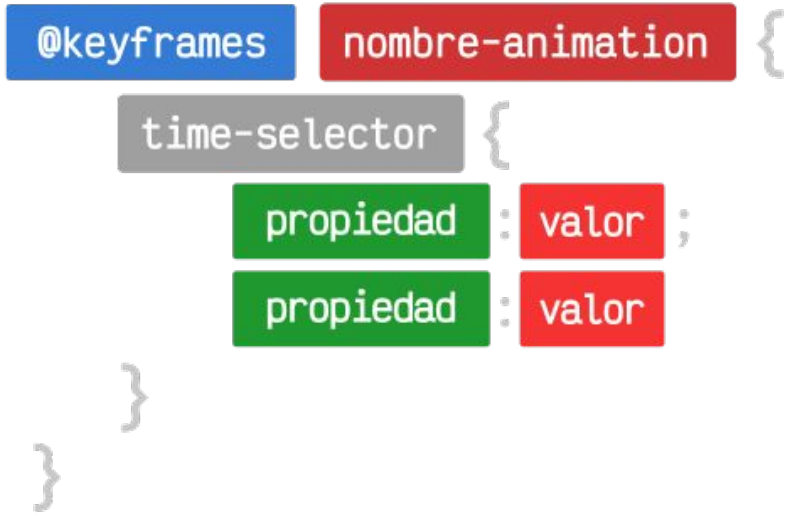
Para el comportamiento de una animación, necesitamos conocer las siguientes propiedades, que son una «ampliación» de las propiedades de las transiciones CSS:

Propiedades	Descripción	Valor
`animation-name`	Nombre de la animación a aplicar.	`none` | nombre
`animation-duration`	Duración de la animación.	`0` | TIME
`animation-timing-function`	Ritmo de la animación.	Valor del ritmo
`animation-delay`	Retardo en iniciar la animación.	`0` | TIME
`animation-iteration-count`	Número de veces que se repetirá.	`1` | `infinite` | NUMBER
`animation-direction`	Dirección de la animación.	`normal` | `reverse` | `alternate` | `alternate-reverse`
`animation-fill-mode`	Como se «completa» la animación.	`none` | `forwards` | `backwards` | `both`
`animation-play-state`	Estado de la animación.	`running` | `paused`

https://lenguajecss.com/css/animaciones/animaciones/#propiedades-de-animaci%C3%B3n-css

La regla `@keyframes`

Una animación está formada por varios fotogramas que mostradas una detrás de otra generan el efecto de movimiento. En CSS, los fotogramas se crean a partir de propiedades CSS, y sólo crearemos fotogramas clave y el resto los generará el navegador. Para definir esos fotogramas clave, utilizaremos la regla `@keyframes`, la cual es muy sencilla de utilizar. Se basa en el siguiente esquema:



Cada uno de estos `time-selector` será un momento clave de cada uno de los fotogramas clave de nuestra animación, y ya veremos que pueden definirse muchos en una misma animación.

https://lenguajecss.com/css/animaciones/keyframes/#selectores-from-y-to

La propiedad `transform`

Permite recibir una función de transformación determinada, la cual será aplicada en el elemento HTML que se verá transformado visualmente.

Propiedad	Descripción	Valor
`transform`	Aplica una (o varias) transformaciones CSS al elemento.	función 1, función 2, función 3

Tipo de transformación	Descripción
Traslación 2D	Desplaza un elemento en el eje X (izquierda, derecha) y/o en el eje Y (arriba, abajo)
Escalado 2D	Escala el elemento una determinada cantidad más grande o más pequeña. También se puede voltear.
Rotación 2D	Gira el elemento sobre su eje X o sobre su eje Y. También se puede girar sobre sí mismo.
Deformación 2D	Inclina el elemento sobre su eje X o sobre su eje Y.

https://lenguajecss.com/css/transformaciones/transform/#funciones-2d

Minificar CSS

La minificación (en inglés, uglify o minification) es la acción de eliminar caracteres o comentarios en un archivo de código para el navegador (.html, .css o .js, por ejemplo).

El objetivo es reducir su tamaño total para descargarlos más rápido. En archivos CSS muy grandes es una buena práctica utilizar herramientas de minificación para reducir el tamaño del archivo CSS, condensando toda su información, eliminando espacios, nuevas líneas, etc...

Haciendo esto, conseguiremos que el archivo ocupe menos, pero a cambio perderemos legibilidad. Por esta razón, es habitual conservar los archivos CSS originales (sin minificar) que serán los que editemos y modifiquemos, mientras que los archivos minificados se generarán a partir de estos mediante herramientas automatizadas:

- index.css (Archivo editable, legible, orientado a humanos)
- index.min.css (Archivo minificado, generado de forma automática, no editable, el que lee el navegador)

CSS Nesting nativo

La idea detrás del concepto de CSS Nesting (código CSS anidado) es tener fragmentos o bloques de código uno dentro de otros, haciendo que estos selectores sean mucho más intuitivos para el programador, y por lo tanto, mucho más fáciles de mantener.

Sintaxis simple

La sintaxis más sencilla que podemos utilizar es dentro de un elemento, podemos volver a mencionar otro.

Sintaxis avanzada

Dentro del CSS Nesting tenemos el operador &, que nos permite hacer referencia al selector inmediatamente padre dentro del anidamiento. Este selector lo tenemos que usar cuando anidemos etiquetas HTML o queramos concatenar elementos, clases, pseudoclases, etc.

https://lenguajecss.com/css/calidad-de-codigo/css-nesting/#sintaxis-de-anidamiento

Metodologías CSS

Se considera una metodología a una serie de métodos, consejos o forma de trabajar que harán que tu código sea más fácil mantener y organizar. Existen metodologías más generalistas, otras que simplemente son convenciones para la sintaxis CSS y la forma de escribir clases y otras que se centran en cómo organizar los ficheros o la arquitectura de tu página.

No estás obligado a seguir una metodología concreta, simplemente son recomendaciones con las que puede ser más sencillo organizarte al escribir y mantener código, y depende de tu forma de trabajar que te guste más una que otra.

Las metodologías más populares hoy en día son: **CSS Semántico** (en general), **BEM**, **CSS-in-JS** y **Utility-first CSS**.

https://lenguajecss.com/css/calidad-de-codigo/metodologias-css/#qu%C3%A9-es-una-metodolog%C3%ADa-de-css