

INDICE DE PSEUDO-ELEMENTOS – solo de ayuda para estudio.
Documento con contenido extraído de Internet para estudiantes
IFCD0110.

TIPOS DE PSEUDO-ELEMENTOS

¿Qué es un Pseudoelemento?

Los pseudoelementos son una característica de CSS que permite hacer referencias a «comportamientos virtuales no tangibles».

¿Qué significa esto? Dicho de otra forma: Los pseudoelementos permiten seleccionar y dar estilo a elementos que no existen en el HTML, o que no son un simple elemento en sí. La sintaxis de los pseudoelementos está precedida de dos puntos dobles (::) para diferenciarlos de las pseudoclases, las cuales sólo tienen dos puntos (:).

```
selector #id .clase [atributo] :pseudoclase ::pseudoelemento {  
    propiedad : valor ;  
    propiedad : valor  
}
```

La sintaxis con :: no surgió desde los inicios de CSS, por lo que aún hoy en día es posible encontrar fragmentos de código desactualizados que utilizan pseudoelementos como ::before o ::after con la sintaxis de pseudoclase: :before y :after.

Tipos de pseudoelementos

Existen varios tipos de pseudoelementos, que se encuentran organizados en categorías. Puedes encontrarlos en la siguiente tabla:

Pseudoelementos	Significado
Contenido generado	Información generada desde CSS, sin existir en el HTML. <code>::before</code> , <code>::after</code>
Contenido tipográfico	Pseudoelementos relacionados con temas de tipografías. <code>::first-line</code> , <code>::first-letter</code>
Contenido destacado	Pseudoelementos para remarcar o destacar información. <code>::selection</code> , <code>::target-text</code> , <code>::spelling-error</code> , <code>::grammar-error</code>
WebComponents	Pseudoelementos relacionados con WebComponents <code>::part</code> , <code>::slotted</code>
View Transition API	Pseudoelementos de transición de cambio de página. <code>::view-transition</code> , <code>::view-transition-group</code> , <code>::view-transition-image-pair</code> <code>::view-transition-new</code> , <code>::view-transition-old</code>
Otros pseudoelementos	Pseudoelementos de otras categorías variadas <code>::marker</code> , <code>::placeholder</code> , <code>::file-selector-button</code>

Contenido generado en CSS

Dentro de la categoría de los pseudoelementos CSS, y quizás uno de los más conocidos, se encuentra la propiedad **content**.

Esta propiedad se utiliza en selectores que incluyen los pseudoelementos **::before** o **::after** (que explicaremos un poco más adelante), y **su objetivo es indicar que vamos a crear contenido antes o después del elemento indicado en cuestión.**

Veamos un pequeño resumen de los elementos que participan en esta categoría:

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

Vamos a explicar como funciona cada uno de ellos.

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 texto indicado.	<code>content: "Contenido:";</code>
<code>attr(atributo)</code>	Añade el valor del atributo HTML indicado.	<code>content: attr(href);</code>
IMAGE <code>url("...")</code>	Añade la imagen indicada en la URL.	<code>content: url("icon.png");</code>
GRADIENT	Añade un gradiente del tamaño indicado.	<code>content: linear-gradient(red, blue);</code>
COUNTER	Define un contador CSS para mostrar.	<code>content: counter(item);</code>

Ten en cuenta que la propiedad content sólo funciona dentro de pseudoelementos, como por ejemplo `::before` y `::after`, los cuales explicaremos a continuación. No puede utilizarse en otros selectores.

El pseudoelemento ::before

Los pseudoelementos `::before` y `::after` permiten hacer referencia a «justo antes del elemento» y «justo después del elemento», respectivamente.

Además de selectores básicos como clase o id, combinadores, atributos o pseudoclases, puedes añadir un pseudoelemento, precedido por `::`

En el caso del 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:

```
q::before {  
  content: "«";  
  color: red;  
}
```

`<p>Y dije <q>Hola, ¿qué tal?</q>, entre susurros.</p>`

Y dije «Hola, ¿qué tal?, entre susurros.

En este caso, estamos añadiendo el texto « justo antes de los elementos <q> que aparezcan en nuestro documento, además de pintarlo de color rojo. 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

De la misma forma, tenemos el pseudoelemento ::after, que permitirá añadir contenido después de la etiqueta de cierre en cuestión. Vamos a completar el código anterior, utilizando también un ::after y añadiéndole más estilos a la propia etiqueta:

```
q::before {
  content: "«";
  color: red;
}

q::after {
  content: "»";
  color: red;
}

q {
  color: darkred;
  font-style: italic;
}
```

<p>Y dije <q>Hola, ¿qué tal?</q>, entre susurros.</p>

Y dije «*Hola, ¿qué tal?*», entre susurros.

Ahora, como se puede ver en la demo, el ejemplo es mucho más sencillo de entender visualmente, a la vez que posee cierta semántica que lo hace más fácil de entender, no sólo para humanos, sino también para una máquina, sistemas automatizados o desarrolladores que quieran interactuar con la información y comprenderla.

La función attr()

Es interesante recalcar la utilidad de la función CSS attr(A). Esta función se puede utilizar en la propiedad content para recuperar el valor del atributo HTML especificado en A.

Vamos a ampliar el ejemplo anterior, añadiéndole al elemento q un atributo con el autor de ese mensaje:

```
q::before {
  content: "«";
  color: red;
}

q::after {
  content: "» (" attr(data-author) )";
  color: red;
}

q {
  color: darkred;
  font-style: italic;
}
```

```
<p>Y dije <q data-author="ManzDev">Hola, ¿qué tal?</q>, entre susurros.</p>
```

Y dije «*Hola, ¿qué tal?» (ManzDev)*, entre susurros.

En este ejemplo, añadimos el contenido del atributo data-author de las etiquetas <q> para mostrarlo en la página entre paréntesis.

Observa que la concatenación (unión) de los caracteres en CSS se hace simplemente con espacios, abriendo y cerrando las comillas para indicar los textos. Esto puede realizarse con cualquier etiqueta y atributo que desees.

El uso de la función attr() podría ser muy útil en ciertos casos, como por ejemplo, en una página que muestra enlaces y el usuario va a imprimir.

Podríamos utilizar medios de impresión que apliquen un CSS especial, donde el contenido de los atributos href en las etiquetas <a> sean mostrados, ya que en una impresión no podemos hacer doble clic en los textos subrayados (con enlace):

```
a[href]::after {
  content: "( " attr(href) " )";
  padding: 5px;
  color: navy;
}
```

```
<ul>
  <li>Enlace a mi página: <a href="https://manz.dev/">ManzDev</a></li>
  <li>Enlace a mi Twitter: <a href="https://twitter.com/Manz">@Manz</a></li>
</ul>
```

- Enlace a mi página: [ManzDev \(https://manz.dev/ \)](https://manz.dev/)
- Enlace a mi Twitter: [@Manz \(https://twitter.com/Manz \)](https://twitter.com/Manz)

Truco: 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.

Pseudoelementos tipográficos

Aunque `::before` y `::after` suelen ser los ejemplos de pseudoelementos más frecuentes, existen muchos otros pseudoelementos. Por ejemplo, dentro de una categoría de pseudoelementos tipográficos, podemos encontrar los pseudoelementos **`::first-letter` o `::first-line`**:

Pseudoelemento	Descripción
<code>::first-letter</code>	Aplica los estilos en la primera letra de un texto.
<code>::first-line</code>	Aplica los estilos en la primera línea de un texto.

El pseudoelemento `::first-letter`

El pseudoelemento `::first-letter`, como su propio nombre indica, **permite seleccionar y dar estilo a la primera letra del texto indicado**.

Así podremos recrear el efecto clásico de cuentos infantiles o algunas otras obras, donde la primera letra se establece mucho más grande que el resto del texto y con una tipografía decorativa mucho más llamativa.

```
@import url("https://fonts.googleapis.com/css2?family=Cinzel+Decorative&display=swa");

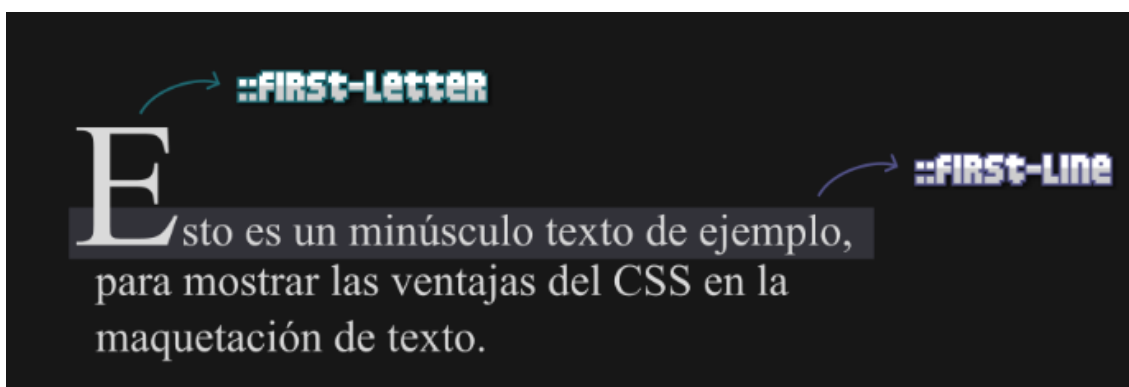
p {
  font-family: 'Open Sans', sans-serif;
  font-size: 1.5rem;
}

p::first-letter {
  font-family: 'Cinzel Decorative', serif;
  font-size: 5rem;
}
```

```
<p>  
  En un lugar de la Mancha, cuyo nombre no logro acordarme...  
</p>
```

En un lugar de la Mancha, cuyo nombre no logro acordarme...

En el caso del pseudoelemento `::first-letter`, le damos estilo a la primera letra del texto, a la cuál le colocamos una tipografía (que cargamos de Google Fonts) y le establecemos un tamaño de texto mayor.



El pseudoelemento `::first-line`

Por otro lado, el pseudoelemento `::first-line` es muy útil para aplicar un estilo solamente a la primera línea del texto indicado.

Puede ser interesante si queremos cambiar algún detalle, pero que afecte exclusivamente a la primera línea, independientemente del tamaño que tenga (lo calcula el navegador).

```
p {  
  font-family: 'Open Sans', sans-serif;  
  font-size: 1rem;  
  color: #333;  
}  
  
p::first-line {  
  color: red;  
}
```

```
<p>  
  En un lugar de la Mancha, <br>  
  cuyo nombre no logro acordarme...  
</p>
```

En un lugar de la Mancha,
cuyo nombre no logro acordarme...

Pseudoelementos de resaltado

Existe una serie de pseudoelementos **orientados a la selección o resaltado de texto en un documento HTML mostrado a través de un navegador**. Veamos que pseudoelementos tenemos a nuestra disposición:

Pseudoelemento	Descripción
::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 un error ortográfico.
::grammar-error	Aplica estilos al fragmento de texto resaltado por un error gramatical.

El pseudoelemento ::selection

Cuando seleccionamos un fragmento de texto, el navegador suele aplicar un color de fondo que depende del sistema operativo, del tema, o similar.

Al igual que ocurre con la propiedad `accent-color`, es posible que queramos aprovechar esto para definir un color que tengan sentido con los colores corporativos de la marca o web, por lo que podríamos cambiarlo haciendo uso del pseudoelemento `::selection`:

```
::selection {  
  background: indigo;  
  color: white;  
}
```

<p>Selecciona este texto para ver el color.</p>

Selecciona este texto para ver el color.

El pseudoelemento ::target-text

En algunos casos, al crear un enlace a una página, **tras el ancla de la URL definida con el carácter #**, se puede añadir el fragmento de texto **:::text=** seguido del texto, palabra o frase a buscar en la propia página. **Al hacer esto, el navegador resaltará esa parte para que sea más sencillo encontrarla.**

Esta página suele estar destacada con color de fondo amarillo sobre letras negras, pero podemos personalizarlo a través del **pseudoelemento ::target-text**:

```
::target-text {  
  background: lime;  
  color: black;  
}
```

Ahora, en una URL terminada en **#:::text=ManzDev**, todos los textos **ManzDev** aparecerían resaltados en el color seleccionado por **::target-text**.

El pseudoelemento ::spelling-error

El pseudoelemento **::spelling-error** nos permite modificar los estilos que se aplican a como se muestra un **error ortográfico** en el navegador, que normalmente se visualiza con un subrayado ondulado rojo en la palabra o texto afectado.

```
textarea {  
  min-width: 400px;  
  min-height: 100px;  
  font-size: 1.25rem;  
}  
  
::spelling-error {  
  background: darkred;  
  color: white;  
}
```

```
<p>
  Pulsa en el interior del campo de texto
  para que revise la ortografía:
</p>

<textarea spellcheck="true">Vamos a cometer un error hortográfico para ver el resa
```

Pulsa en el interior del campo de texto para que revise la ortografía:

Vamos a cometer un error
hortográfico para ver el resaltado
de sintaxis.

El pseudoelemento ::grammar-error

Por su parte, el pseudoelemento ::grammar-error es exactamente igual al anterior, sólo que se encarga de señalar **los errores gramaticales del texto**, y no los ortográficos.

Nuevamente, hagamos un ejemplo, añadiendo esta pseudoclase sobre el ejemplo anterior:

```
textarea {
  min-width: 400px;
  min-height: 100px;
  font-size: 1.25rem;
}

::spelling-error {
  background: darkred;
  color: white;
}

::grammar-error {
  background: orangered;
  color: white;
}
```

```
<p>
  Pulsa en el interior del campo de texto
  para que revise la ortografía:
</p>

<textarea spellcheck="true">Vamos a cometer un error ortográfico para ver el resa
```

Pulsa en el interior del campo de texto para que revise la ortografía:

Vamos a cometer un error
ortográfico para ver el resultado
de sintaxis, y además, también vamos
a cometer 1 errores gramatical.

Slots en WebComponents (JavaScript)

Variables CSS y CSS Parts (JavaScript)

View Transition API. Animaciones de transición entre páginas y transacciones CSS (lo veremos el viernes o el lunes de la próxima semana)

Generalmente, cuando realizamos animaciones o transiciones, los elementos deben estar presentes en el DOM, es decir, tienen que existir en la página.

El **modelo de objeto de documento (DOM)** es una interfaz de programación para los documentos HTML y XML. Facilita una representación estructurada del documento y define de qué manera los programas pueden acceder, al fin de modificar, tanto su estructura, estilo y contenido.

Si pensamos en la animación o transición de una página a otra página, se trata de animar unos elementos del DOM de una página hasta unos elementos del DOM de otra página que aún no está cargada.

Si en un sitio web MPA (web orientada a contenido o SEO) pulsamos en un enlace, el navegador se mueve hacia otra página, volviendo a recargar toda la página y produciéndose una especie de parpadeo, que dependerá de la velocidad con la que cargue la nueva página, la conexión a Internet, etc. Ese parpadeo, generalmente es muy molesto y rompe con una navegación fluida.

Sin embargo, meter animaciones en este proceso es complejo, porque básicamente queremos animar con información que aún no tenemos disponible (y cuando la tengamos, hemos abandonado la anterior).

Aquí es donde entra un nuevo estándar llamado View Transitions, que lo soluciona de una forma muy original.

¿Qué son las View Transition?

El nombre de View Transition viene de «transiciones de vistas», es decir, transiciones entre páginas de navegación, ya que a estas páginas muchas veces se les suele denominar Vistas (views).

¿En qué se basan las View Transition API? En unos conceptos muy sencillos:

- 📷 Cuando se navega entre páginas, el navegador hace una «foto» de la página antes de abandonarla.
- 🧑 El navegador carga la nueva página de forma transparente al usuario (mantiene la foto anterior)
- 📷 Cuando la nueva página ha cargado, hace una nueva «foto» de la página de destino
- 🛠 Aplica la animación o transición CSS de una «foto» a otra.

En la actualidad, esta característica está en fase **EXPERIMENTAL**, por lo que si quieres utilizarla necesitarás revisar el soporte y activar la opción **View Transition on navigation** (en el caso de Chrome), por lo que tendrás que abrir una pestaña con la URL <chrome://flags/#view-transition-on-navigation> para activarla.

Ahora mismo, la compatibilidad de esta característica es muy reciente, por lo que sólo algunos navegadores disponen de esta feature.

Se está implementando no sólo para aplicaciones SPA (Vue, React, etc...), sino también para webs MPA (Astro, Next, Nuxt, u otras más tradicionales donde el enfoque está en el contenido y en el SEO). Poco a poco irá ampliándose su soporte:

Current aligned		Usage relative		Date relative	Filtered	All												
Chrome	Edge	Safari	Firefox	Opera	IE		Chrome for Android	Safari on iOS	Samsung Internet	Opera Mini	Opera Mobile	UC Browser for Android	Android Browser	Firefox for Android	QQ Browser			
106	105	16.0	104	91				16.0	19.0									
107	107	16.1	105	92				16.1	19.0									
108	108	16.2	106	94				16.2										
109	109	16.3	107	95				16.3	20									
110	110	16.4	108	96				16.4	20									
111	111	16.5	109	97				16.5	21									
112	112	16.6	110	98				16.6	22									
113	113	16.7	111	99				16.7										
114	114	16.8	112	100				16.8										
115	115	16.9	113	101				16.9										
116	116	17.0	114	102				17.0										
117	117	17.1	115	103				17.1	23									
118	118	17.2	116	104				17.2										
119	119	17.3	117	105				17.3										
120	120	17.4	118	106				17.4										
121	121	17.5	119	107				17.5										
122	122	17.6	120	108														
123	123	17.7	121	109														
124	124	17.8	122	110														
125	125	17.9	123	111														
126	126	18.0	124	112														
127	127	18.1	125	113														
128	128	18.2	126	114														
129	129	18.3	127	115														
130	130	18.4	128	116														
131	131	18.5	129	117														
132	132	18.6	130	118														
133	133	18.7	131	119														
134	134	18.8	132	120														
135	135	18.9	133	121														
136	136	19.0	134	122														
137	137	19.1	135	123														
138	138	19.2	136	124														
139	139	19.3	137	125														
140	140	19.4	138	126														
141	141	19.5	139	127														
142	142	19.6	140	128														
143	143	19.7	141	129														
144	144	19.8	142	130														
145	145	19.9	143	131														
146	146	20.0	144	132														
147	147	20.1	145	133														
148	148	20.2	146	134														
149	149	20.3	147	135														
150	150	20.4	148	136														
151	151	20.5	149	137														
152	152	20.6	150	138														
153	153	20.7	151	139														
154	154	20.8	152	140														
155	155	20.9	153	141														
156	156	21.0	154	142														
157	157	21.1	155	143														
158	158	21.2	156	144														
159	159	21.3	157	145														
160	160	21.4	158	146														
161	161	21.5	159	147														
162	162	21.6	160	148														
163	163	21.7	161	149														
164	164	21.8	162	150														
165	165	21.9	163	151														
166	166	22.0	164	152														
167	167	22.1	165	153														
168	168	22.2	166	154														
169	169	22.3	167	155														
170	170	22.4	168	156														
171	171	22.5	169	157														
172	172	22.6	170	158														
173	173	22.7	171	159														
174	174	22.8	172	160														
175	175	22.9	173	161														
176	176	23.0	174	162														
177	177	23.1	175	163														
178	178	23.2	176	164														
179	179	23.3	177	165														
180	180	23.4	178	166														
181	181	23.5	179	167														
182	182	23.6	180	168														
183	183	23.7	181	169														
184	184	23.8	182	170														
185	185	23.9	183	171														
186	186	24.0	184	172														
187	187	24.1	185	173														
188	188	24.2	186	174														
189	189	24.3	187	175														
190	190	24.4	188	176														
191	191	24.5	189	177														
192	192	24.6	190	178														
193	193	24.7	191	179														
194	194	24.8	192	180														
195	195	24.9	193	181														
196	196	25.0	194	182														
197	197	25.1	195	183														
198	198	25.2	196	184														
199	199	25.3	197	185														
200	200	25.4	198	186														
201	201	25.5	199	187														
202	202	25.6	200	188														
203	203	25.7	201	189														
204	204	25.8	202	190														
205	205	25.9	203	191														
206	206	26.0	204	192														
207	207	26.1	205	193														
208	208	26.2	206	194														
209	209	26.3	207	195														
210	210	26.4	208	196														
211	211	26.5	209	197														
212	212	26.6	210	198														
213	213	26.7	211	199														
214	214	26.8	212	200														
215	215	26.9	213	201														
216	216	27.0	214	202														
217	217	27.1	215	203														
218	218	27.2	216	204														
219	219	27.3	217	205														
220	220	27.4	218	206														
221	221	27.5	219	207														
222	222	27.6	220	208														
223	223	27.7	221	209														
224	224	27.8	222	210														
225	225	27.9	223	211														
226	226	28.0	224	212														
227	227	28.1	225	213														
228	228	28.2	226	214														
229	229	28.3	227	215														
230	230	28.4	228	216														

Cómo usar View Transitions

Vamos a analizar la forma más sencilla de crear View Transition.

En primer lugar, necesitaremos añadir la siguiente etiqueta HTML en la cabecera de la página, es decir, en el interior de la etiqueta <head> del HTML.

La etiqueta <meta> indicará al navegador que se van a activar las **View Transition** y con el valor **same-origin** se indica que se van a realizar en el mismo dominio:

```
<head>
  <meta name="view-transition" content="same-origin">
</head>
```

La idea de esta especificación es que en el futuro se puedan hacer animaciones **cross-origin**, es decir, entre dominios diferentes. Sin embargo, de momento sólo se soportan en el mismo dominio.

Crear la animación entre vistas

Una vez hecho esto, ya tendremos unas transiciones suaves por defecto, pero serán muy genéricas, por lo que no serán muy vistosas.

Lo que haremos será utilizar la **propiedad CSS view-transition-name** mediante la cuál le vamos a dar un nombre al elemento que queremos personalizar su animación:

Propiedad	Valor	Descripción
view-transition-name	none NAME	Le damos un nombre al elemento, para utilizarlo posteriormente en los pseudoelementos.

Así pues, por ejemplo, podríamos elegir **el elemento con clase .container** que es el que tiene todo el contenido de texto de la página, para animarlo posteriormente.

En este caso, le daremos el nombre de page (podría ser cualquier otro nombre):

```
.container {  
  view-transition-name: page;  
}
```

Animación inicial y final

Ahora, vamos a utilizar los pseudoelementos que hacen referencia a las «fotos» de las que hablabamos antes.

Por un lado, el pseudoelemento `::view-transition-old()` hace referencia a la «foto» del DOM de la página anterior a pulsar en el enlace y navegar, y el pseudoelemento `::view-transition-new()` hace referencia a la «foto» del DOM de la página nueva a la que hemos navegado:

Pseudoelementos	Descripción
<code>::view-transition-old(NAME)</code>	Inicio de la transición. Hace referencia al DOM de la página antigua.
<code>::view-transition-new(NAME)</code>	Final de la transición. Hace referencia al DOM de la página nueva.

A continuación, colocamos una animación tanto en la transición de inicio como en la transición de final. El código que escribiríamos sería el siguiente (indicamos el nombre `page` entre paréntesis):

```
::view-transition-old(page) {  
  animation: fade 0.2s linear forwards;  
}  
  
::view-transition-new(page) {  
  animation: fade 0.3s linear reverse;  
}
```

Observa que en el primer caso, `::view-transition-old()` realizamos la animación `fade` durante `0.2s` a un ritmo constante `linear` y una vez termine se queda en el último frame.

Entonces la animación continuará con la de final, `::view-transition-new()` y realizará la misma animación `fade`, pero durante `0.3s` y al revés, de esta forma nos ahorramos tener que crear dos animaciones diferentes.

Por ejemplo, si la animación `fade` es la siguiente, la animación de inicio (`old`) ocultará el contenido del elemento `.container` (baja el `opacity` hasta cero), y mientras lo hace, hará que descienda `50px` hacia abajo.

Al terminar, continuará la animación de final (new), que realizará la animación al revés, es decir, aparecerá desde totalmente invisible y desplazada 50px hacia abajo, subirá 50px hasta volverse opaca del todo:

```
@keyframes fade {  
  0% {  
    opacity: 1;  
    transform: translateY(0);  
  }  
  
  100% {  
    opacity: 0;  
    transform: translateY(50px);  
  }  
}
```

```
.container {  
  view-transition-name: page;  
}  
  
::view-transition-old(page) {  
  animation: fade 0.2s linear forwards;  
}  
  
::view-transition-new(page) {  
  animation: fade 0.3s linear reverse;  
}  
  
@keyframes fade {  
  0% {  
    opacity: 1;  
    transform: translateY(0);  
  }  
  
  100% {  
    opacity: 0;  
    transform: translateY(50px);  
  }  
}
```

Otros pseudoelementos

Al margen de los pseudoelementos anteriores, explicados en su respectiva sección, nos quedan algunos pseudoelementos sin catalogar.

Vamos a repasarlos:

Pseudoelemento	Descripción
::marker	Aplica estilos a las marcas o símbolos 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.

Pseudoelemento ::marker

El pseudoelemento **::marker** sirve para hacer referencias a los signos o marcas de la listas (**** o ****), en el caso de que queramos que tengan un estilo diferente al del texto de la lista.

En este ejemplo se aplica el estilo a los elementos **** de los ítems de una lista ****:

```
ul li::marker {
  content: "➤ ";
  color: red;
}
```

```
<ul>
  <li>Opción número 1.</li>
  <li>Opción número 2.</li>
  <li>Opción número 3.</li>
  <li>Opción número 4.</li>
  <li>Opción número 5.</li>
</ul>
```

- Opción número 1.
- Opción número 2.
- Opción número 3.
- Opción número 4.
- Opción número 5.

Ten en cuenta que ciertas propiedades CSS puede que no tengan efecto en este pseudoelemento. Sin embargo, propiedades como font-size, color o content funcionarán correctamente.

Pseudoelemento ::backdrop (con javascript)

El pseudoelemento ::backdrop nos permite aplicar estilos como oscurecer o desenfocar el fondo detrás de un elemento para darle más protagonismo al elemento que está en primer plano.

Este pseudoelemento debe utilizarse en un diálogo o mensaje modal <dialog>, que comunica información importante y hay que prestarle atención y poner el foco en él.

Veamos un pequeño ejemplo:

```
dialog::backdrop {  
  background: linear-gradient(140deg, indigo, black);  
  opacity: 0.9;  
}
```

```
<p>Texto de la página</p>
```

```
<dialog>Hello!!!</dialog>
```

```
const dialog = document.querySelector("dialog");  
dialog.showModal();
```

Texto de la página

Hello!!!

En este caso, el pseudoelemento ::backdrop aplicará los estilos al contenido de fondo del diálogo, sin que afecte al mismo. También se podría utilizar con una imagen de fondo, junto a la propiedad CSS backdrop-filter con el valor blur(3px) (o similar). En ese caso, se desenfocaría la imagen y se le restará importancia al fondo y el usuario se centrará en el mensaje o la información de primer plano.

Pseudoelemento ::placeholder (con formularios)

Mediante el pseudoelemento ::placeholder podemos dar estilos particulares a los elementos <input> que tienen el atributo placeholder definido.

El atributo placeholder sirve para indicar una sugerencia o mensaje de ayuda o de información con la finalidad de ese campo de texto, normalmente una ayuda de lo que deben escribir o similar.

```
input::placeholder {  
  background: darkred;  
  color: white;  
  padding: 5px;  
}
```

```
<input type="text" placeholder="Sugerencia de texto">
```



Con ::placeholder podremos cambiar este color, por ejemplo, que suele ser un gris apagado por defecto. Ten en cuenta que afectará sólo al mensaje de placeholder, no a las dimensiones del propio elemento <input>.

Pseudoelemento ::file-selector-button (con formularios)

El pseudoelemento ::file-selector-button hace referencia al <button> que se incluye dentro de un elemento <input type="file">, o lo que es lo mismo, un botón para enviar ficheros a través de un formulario.

De esta forma, podemos personalizar la apariencia del botón del formulario.

```
input::file-selector-button {  
  background: indigo;  
  color: white;  
  padding: 0.5rem 1rem;  
  border: 2px solid black;  
}
```

```
<input type="file" value="Enviar archivo">
```

Seleccionar archivo

Ninguno archivo selec.

Formas de enlazar contenidos CSS

```
<p>
  Lorem ipsum dolor sit ame
  tempor incididunt ut labo
  quis nostrud exercitation
  consequat. Duis aute irur
  cillum dolore eu fugiat n
  proident, sunt in culpa q
</p>
<style type="text/css">
</style>
```

Con la etiqueta style, rellenando el contenido css como si fuera una etiqueta más. El tipo a aplicar es CSS.

En línea con la etiqueta:

```
<p style="color:red">
  Lorem ipsum dolor sit amet, consectetur adipisicing
  tempor incididunt ut labore et dolore magna aliqua.
  quis nostrud exercitation ullamco laboris nisi ut a
  consequat. Duis aute irure dolor in reprehenderit i
  cillum dolore eu fugiat nulla pariatur. Excepteur s
  proident, sunt in culpa qui officia deserunt mollit
</p>
```

```
<link rel="stylesheet" type="text/css" href="estilo.css">
</head>
<body>
```

Con link con el atributo rel y el valor stylesheet (hoja de estilo). Tipo texto CSS y el enlace al fichero CSS dónde están los estilos.

Formas de cargar fuentes en CSS

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.

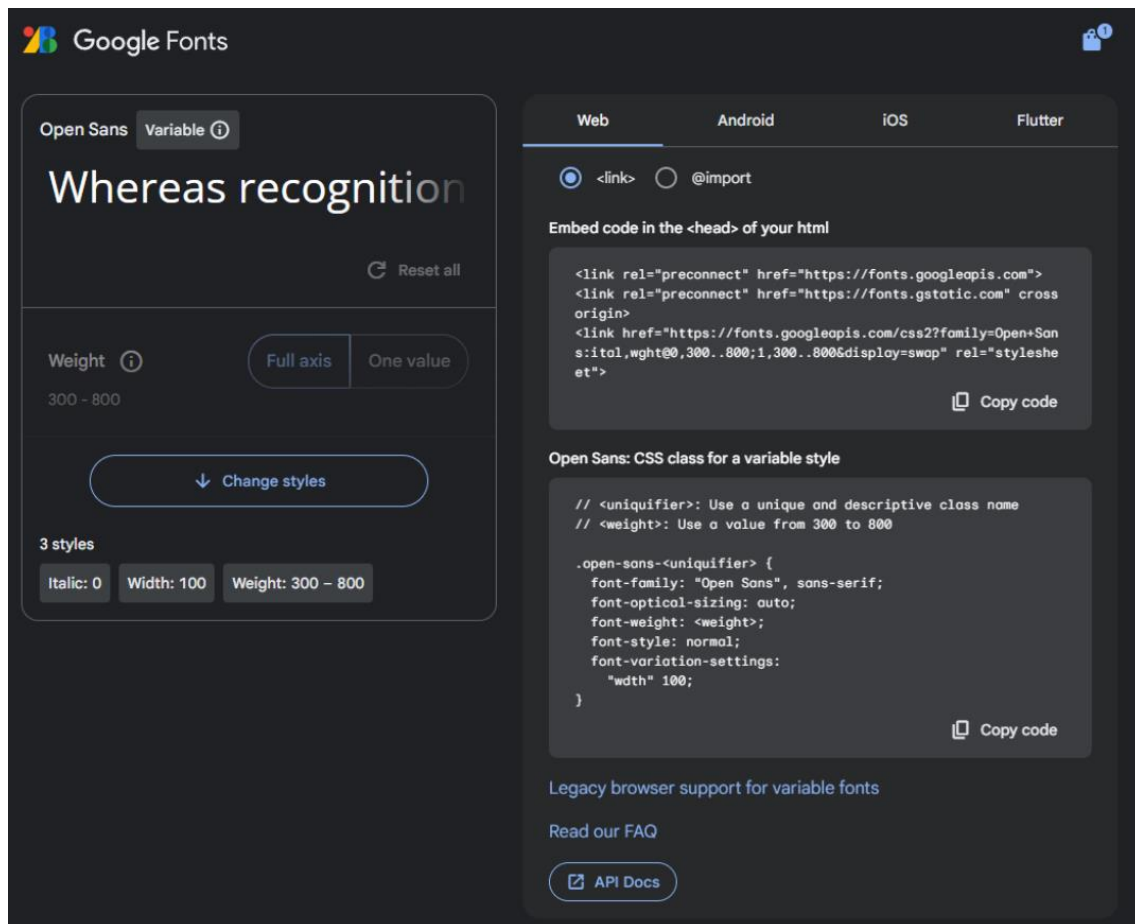
Antiguamente, una de las ventajas de usar Google Fonts era el CDN compartido entre dominios, es decir, si la tipografía se usaba en una web, se compartía en otras. A partir de 2020, los navegadores han cambiado su política de caché, por lo que ya no aplica.

Seleccionar la tipografía

Así pues, lo primero es acceder a **Google Fonts** y **revisar la tipografía deseada o realizar una búsqueda para encontrarla**. En nuestro caso, lo haremos con Open Sans. Una vez que encontremos la tipografía deseada, **pulsamos en ella, y luego en el botón superior derecho «Get font»**:



Esto nos llevará a una nueva página, donde se irán acumulando todas las tipografías que hemos ido seleccionando. En nuestro caso, solo tendremos una. Vamos a la zona derecha, y pulsamos en el botón «**Get embed code**». Nos mostrará varios fragmentos de código, que debemos comprender para utilizar la tipografía:



En la parte izquierda tenemos algunas opciones de personalización. En la parte derecha, se nos muestra una serie de pestañas dependiendo si queremos utilizar la tipografía en web o en apps (Android, iOS o Flutter). Nos centraremos en la de web, y vamos a analizar el fragmento de código que nos proporciona.

Preconexión a Google Fonts

La etiqueta `<link rel="preconnect">` es un fragmento de código para que el navegador haga una preconexión a los subdominios de descarga de las tipografías. Esto hace que el tiempo de descarga sea un poco más corta:

```
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
```

Cargar la tipografía en HTML

Existen dos formas de incluir la tipografía en nuestra página.

- Mediante un fragmento de código HTML, [marcando la opción <link>](#),
- o mediante un fragmento de código CSS, [marcando la opción @import](#).

Veamos el fragmento de código que nos aparece si marcamos la primera, <link> para añadirlo en nuestro código HTML, más concretamente, antes de cerrar la etiqueta </head>:

```
<link href="https://fonts.googleapis.com/css2?family=Open+Sans&display=swap"
      rel="stylesheet">
```

En el código anterior podemos ver una etiqueta <link rel="stylesheet">, que es un archivo CSS prefabricado de Google con reglas @font-face para cargar la tipografía Open Sans desde sus servidores.

Observa bien la URL de su atributo href, porque cada una de las partes de la URL hace referencia a algún detalle especial de esta tipografía, que vamos a analizar a continuación.

Tipografía con múltiples pesos

Al añadir el código anterior, estaremos incluyendo código CSS para preparar nuestra página para utilizar la tipografía Open Sans con el peso 400 (el peso por defecto).

Sin embargo, es posible que en la URL tengamos varios pesos diferentes de la tipografía. En ese caso, se vería así:

```
<link href="https://fonts.googleapis.com/css2?family=Open+Sans:ital,wght@0,300..800;1,300..800&display=swap"
      rel="stylesheet">
```

El texto ital nos indica que queremos la tipografía con estilo itálica.

El texto wght nos indica que queremos pesos variables.

El texto 300..800 nos indica que queremos pesos desde 300 hasta 800

Pero hablaremos más sobre esto posteriormente en el apartado de Fuentes variables. Volvamos a las tipografías individuales.

Múltiples tipografías

Además, **también podemos hacer lo mismo añadiendo diferentes tipografías, con diferentes pesos**. Por ejemplo, en el siguiente fragmento seleccionamos la tipografía **Roboto** (por defecto, grosor 400), la tipografía Lato (con grosor 300). El código correspondiente sería el siguiente:

```
<link href="https://fonts.googleapis.com/css2?family=Lato:wght@300&family=Roboto&disp
      rel="stylesheet">
```

De esta forma conseguimos cargar varias tipografías desde el repositorio de Google de una sola vez, sin la necesidad de varias líneas de código diferentes, que realizarían varias peticiones diferentes a Google Fonts, especialmente importante cuantas más tipografías tengamos.

Nota que en este último ejemplo, en caso de no tener instaladas ningunas de las tipografías anteriores, Estaríamos realizando 4 descargas:

Descargamos el CSS de Google Fonts.

Luego, el navegador lee las reglas @font-face del CSS de Google, y ahí encontrará 2 descargas: **la tipografía Lato de peso 300 y la tipografía Roboto de peso 400.**

Cargar la tipografía en CSS

Recuerda que también podemos incluir las tipografías en nuestro CSS (en lugar del HTML), seleccionando **@import en lugar de <link> en la página de Google Fonts**. Normalmente se suele hacer en el HTML porque suele ser ligeramente más rápido, aunque hablamos de tiempos muy pequeños.

```
@import url('https://fonts.googleapis.com/css2?family=Lato&display=swap');
```

Ten en cuenta que Google Fonts te da este código entre etiquetas <style> y </style>. En ese caso, el fragmento de código iría en HTML, pero si lo quieres añadir en CSS, debes eliminar dichas etiquetas y la regla @import siempre debe ir al principio del fichero .css.

Definir la tipografía a usar

Por último, no hay que olvidar que necesitaremos añadir la propiedad **font-family** al selector CSS que deseemos, como hemos hecho hasta ahora, indicando la tipografía que queremos utilizar. **Sin esto, tendremos cargada la tipografía, pero no la estaremos utilizando en ningún sitio:**

El parámetro display con valor swap que aparece en la URL de Google Fonts, corresponde a la propiedad font-display.

No hay que olvidar que **cuantas más tipografías (y pesos)** incluyamos, **más lenta será la experiencia del usuario**, ya que más contenido tendrá que descargar. Salvo excepciones particulares, lo habitual suele ser elegir entre **1-4 tipografías**, cada una con una finalidad concreta: encabezados o titulares, tipografía de lectura normal, tipografía monoespaciada y tipografía secundaria, por ejemplo.

Fuentes variables CSS – Tipografía variada

Hasta ahora, es posible que sólo hayas utilizado fuentes estáticas (tipografías que tienen un peso y un estilo concreto), ya que son las que tradicionalmente se han utilizado en CSS. Sin embargo, existe también una categoría alternativa llamada fuentes variables, que posee ciertas diferencias y son mucho más prácticas e interesantes en ciertos casos. Veamos las características de cada uno de estos grupos para aclarar conceptos.

Fuentes estáticas

Las fuentes estáticas se dividen en diferentes ficheros según sus pesos y/o estilos. Por ejemplo, la tipografía Open Sans dispone de 10 ficheros con diferentes pesos: 300, 400, 600, 700 y 800 (para el estilo normal) y otros 5 ficheros con los mismos pesos mencionados, pero para los estilos en cursiva (italic). En las fuentes estáticas los pesos suelen variar de 100 en 100.

Si en una web utilizamos **fuentes estáticas**, lo habitual es escoger los pesos concretos que vamos a utilizar. Por ejemplo, cargar en la web los ficheros `open-sans-300.woff2`, `open-sans-300-italic.woff2` y `open-sans-400.woff2`. Con esto podríamos utilizar Open Sans con los pesos 300, 300 en cursiva y 400, concretamente. El tamaño aproximado de las 3 tipografías podría ser unos **150KB** (*50KB cada fichero*).

Fuentes variables

Por contraposición, las fuentes variables se aglutinan generalmente en un solo fichero, conteniendo estilos y diferentes pesos, pudiendo variarlos dinámicamente y con gran precisión a través de propiedades de CSS.

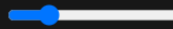
El fichero será más pesado que uno de los anteriores, pero muy probablemente más ligero que 3 o 4 ficheros (combinación más que habitual cuando trabajamos con fuentes estáticas).

Si utilizamos **fuentes variables** en una web, podríamos cargar la tipografía `open-sans-vf.woff2`. Con esta tipografía podríamos usar Open Sans con pesos entre el rango **100** y **900** (*incluyendo valores específicos como 105 o 112*). El fichero de la tipografía podría ocupar unos **130KB**.

Esto hace que, si queremos utilizar una tipografía con diferentes pesos específicos (titular, párrafos, botones, etc...), sea mucho más práctico utilizar una tipografía variable que una tipografía estática, además de tener mucha más flexibilidad y variedad de pesos.


A continuación, puedes observar un ejemplo de cada una, utilizando la tipografía Jost, tanto en su versión estática como en su versión variable:

Fuente estática

 400

- Tipografía: Jost
- Pesos cargados: 300, 400, 500, 600, 700 y 800.
- Tamaño total: 77KB x 6 = **462KB**

Fuente variable

 400

- Tipografía: Jost VF
- Rango de pesos: 100-900.
- Tamaño total: **120KB**

Propiedades de fuentes variables

Al utilizar tipografías o fuentes variables podemos seguir trabajando con la mayoría de las propiedades CSS de fuentes que usamos habitualmente, sin embargo, algunas cosas cambian ligeramente.

Veamos que propiedades pueden ser necesarias para trabajar fuentes variables:

Propiedad CSS	Descripción	Equivalencia
font-weight	Indica el peso que debe tener el texto.	'wght'
font-stretch	Indica el ancho de cada letra del texto.	'wdth'
font-style	Indica la inclinación (slant) o la activación de la cursiva (italic).	'slnt' ó 'ital'
font-optical-sizing	Indica si el texto debe estar optimizado para diferentes tamaños.	'opsz'
font-variation-settings	Propiedad a bajo nivel de las características de fuentes variables.	

Las propiedades CSS anteriores (salvo la última) son las propiedades CSS que se suelen utilizar para ajustar temas relacionados con los estilos y características de las tipografías.

Sin embargo, cuando trabajamos con fuentes variables, muchos navegadores puede que no soporten adecuadamente estas propiedades o características relacionadas con fuentes variables. En ese caso, podemos recurrir a una propiedad a bajo nivel llamada **font-variation-settings**, la cual funcionará si las anteriores no son soportadas.

Cargar fuentes variables

Para cargar tipografías variables personalizadas, podemos utilizar la regla **@font-face** como aprendimos en temas anteriores. Sin embargo, existen algunas diferencias al utilizar tipografías variables:

```

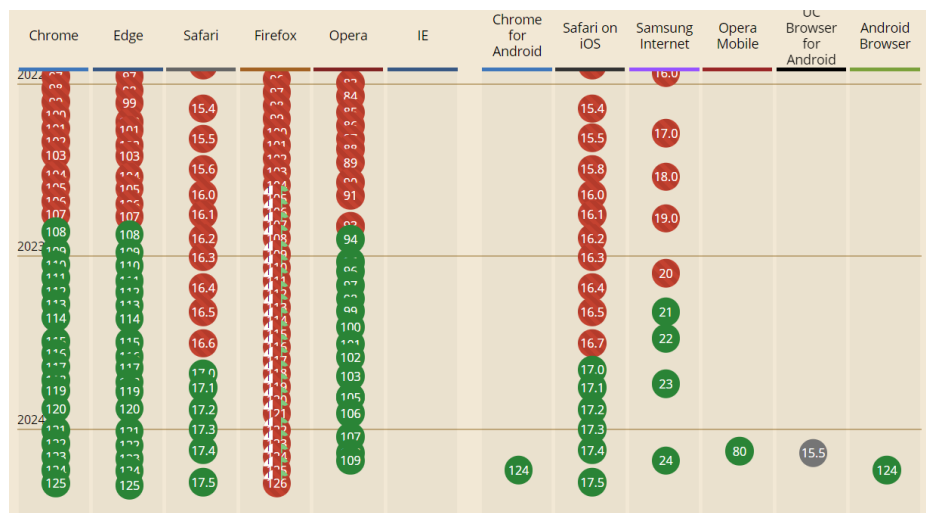
CSS

@font-face {
  font-family: "Montserrat";
  src:
    url("montserrat-vf.woff2") format(woff2) tech(variations), /* Sintaxis moderna */
    url("montserrat-vf.woff2") format("woff2-variations");
  font-weight: 100 900;
  font-stretch: 50% 200%;
}

```

En primer lugar, observa que indicamos la tipografía variable (montserrat-vf.woff2), indicando el formato woff2 y una función tech() con el valor variations.

Esta es una sintaxis moderna que puede que los navegadores aún no soporten. La segunda, por su parte, es una sintaxis anterior que probablemente soporten todos los navegadores de momento:



Un poco más abajo, observamos que utilizamos las propiedades `font-weight` y `font-stretch`, pero con dos valores en lugar de uno. En las tipografías variables, en lugar de utilizar un valor establecemos un rango. De esta forma, estamos permitiendo pesos entre 100 y 900 y ancho de letra entre 50% y 200%. Los valores fuera de estos rangos, se establecerán en el valor más cercano permitido.

Cargar fuentes variables con fallbacks

Podría interesar hacer una carga mixta, en la que se cargue una tipografía variable sólo si la soporta el navegador (si conoce la propiedad `font-variation-settings`), y en caso contrario, una tipografía estática normal.

Si eso es lo que queremos, podemos utilizar el siguiente fragmento de código, donde se utiliza la regla `@supports` para ello:

```

@font-face {
  font-family: "Montserrat";
  src:
    url("montserrat-400.woff2") format("woff2"),
    url("montserrat-400.woff") format("woff"),
    url("montserrat-400.ttf") format("truetype"),
  font-weight: 400;
}

@supports (font-variation-settings: normal) {
  @font-face {
    font-family: "Montserrat";
    src:
      url("montserrat-vf.woff2") format(woff2) tech(variations), /* Sintaxis moderna
      url("montserrat-vf.woff2") format("woff2-variations");
    font-weight: 100 900;
    font-stretch: 50% 200%;
  }
}

```

Observa que lo que hemos hecho es establecer una tipografía estática en el CSS común superior, que inmediatamente será reemplazada por el que está contenido en la regla `@supports` si el navegador soporta tipografías variables. De esta forma no habrá conflicto con las propiedades `font-weight` o similares, que toman dos parámetros en el caso de fuentes variables.

Propiedades específicas

Podemos utilizar las siguientes propiedades para modificar el peso de la tipografía, **el ancho de la letra, la modalidad de cursiva o slant o el ajuste óptico**, que es un modo de ciertas tipografías en el que, si estableces tamaños muy pequeños, los trazos y serifas de la tipografía se muestran de forma más gruesa para que se puedan ver correctamente.

Propiedades CSS	Valores	Descripción
<code>font-weight</code>	400 WEIGHT	Indica el peso a utilizar: 400, 100, 800...
<code>font-stretch</code>	100% PERCENT	Indica el ancho de letra. Rango entre 50% ~ 200%.
<code>font-style</code>	normal italic oblique	Indica si usar cursiva (italic) o cursiva artificial (oblique).
<code>font-style</code>	oblique ANGLE	Inclinación concreta deseada. Por defecto, 14deg .
<code>font-optical-sizing</code>	auto none	Indica si se optimizarán trazos para mejorar visibilidad.

Pero como comentábamos antes, es posible que algunas de estas propiedades CSS no funcionen correctamente aún en navegadores, por lo que podemos utilizar la propiedad de bajo nivel **font-variation-settings**, la cuál nos permite configurar a nuestro gusto una o todas las opciones anteriores de una sola vez.

Usando font-variation-settings

Esta propiedad de bajo nivel permite indicar una o varias características especiales de una tipografía variable.

La sintaxis que se utiliza se basa en indicar la característica a configurar (una cadena de texto de 4 letras), seguida de un valor asociado a dicha característica:

Propiedad CSS	Valor	Descripción
font-variation-settings	"feat" <input type="text" value="VALUE"/>	Activa la característica feat con el valor indicado.
font-variation-settings	"feat1" <input type="text" value="VALUE"/> , "feat2" <input type="text" value="VALUE"/> , ...	Activa varias características a la vez.

Este ejemplo se traduciría a código como veremos a continuación, utilizando la característica wght (weight) con peso 500:

```
.element {  
  font-family: "Montserrat", serif;  
  font-variation-settings: 'wght' 500;  
}
```

Puedes utilizar la página [Wakamai Fondue](#) para arrastrar una tipografía y examinar de que características dispone. Por ejemplo, si [arrastramos la tipografía Montserrat VF](#) vemos que nos da datos como los siguientes:

- Carácteres: 967
- Glifos: 1944
- Layout features: 24 Más info sobre layout font features
- Tamaño: 120KB
- Autora: Julieta Ulanovsky
- Version: 8.000

Si nos fijamos un poco más abajo, [hay un apartado llamado Variable](#) donde aparecen las características que posee la tipografía.

Depende de cada tipografía que características tenga.

A continuación, podemos ver una tabla con algunas de las características que podemos encontrar y cuál es su nombre de 4 letras para indicar en la propiedad font-variation-settings:

Nombre	Propiedad específica	Propiedad a bajo nivel
wght	font-weight	font-variation-settings: 'wght' 400
wdth	font-stretch	font-variation-settings: 'wdth' 100%
slnt	font-style: oblique 14deg	font-variation-settings: 'slnt' 14deg
ital	font-style: italic	font-variation-settings: 'ital' 1
opsz	font-optical-sizing: auto	font-variation-settings: 'opsz' 1

Observa que las características a bajo nivel se identifican siempre con una palabra clave de 4 letras (`wght`, `wdth`, etc...) que generalmente es una abreviatura de la característica en sí.

Luego, se le asocia un valor que depende de la característica. Pueden ser valores como tamaños, porcentajes, unidades o valores 1 o 0 para activar o desactivar respectivamente.

Ten en cuenta que la propiedad **font-variation-settings** establece todas las características de una sola vez.

Si volvemos a escribir la propiedad con otra característica, sobrescribirá la anterior. Para evitar esto, podemos utilizar **CSS Custom Properties** separando por comas múltiples características:

```
.element {
  --font-weight: 400;
  --font-stretch: 100%;

  font-family: "Montserrat", sans-serif;
  font-variation-settings:
    'wght' var(--font-weight),
    'wdth' var(--font-stretch);
}
```

En este ejemplo establecemos las «variables» `--font-weight` y `--font-stretch` en la propiedad `font-variation-settings`.

Si en algún momento queremos modificarlas, no hace falta modificar la propiedad, sino directamente la variable CSS.

En las páginas [V-Fonts](#), [Axis Praxis](#) y en [GitHub](#) puedes encontrar multitud de **fuentes variables** para utilizar en tus diseños. Recuerda investigar bien la tipografía en la página del autor, ya que muchas tipografías son gratuitas, pero muchas otras son de pago y debes tener una licencia.

La regla `@font-face`

- Utilizar tipografías en CSS es maravilloso. Permite personalizar nuestra página web con fuentes elegantes y atractivas.
- Sin embargo, este sistema tenía originalmente un problema: dependen de que la tipografía esté instalada en el sistema.
- Si utilizamos la propiedad **font-family** y **especificamos una tipografía que no está instalada en el sistema, pasará a utilizar otra tipografía alternativa**.
- Sin embargo, que el usuario tenga que instalar la tipografía antes de visitar una web no es algo práctico.
- Por lo tanto, se ideó una forma para que los navegadores puedan precargar tipografías sin que el usuario tenga que instalar dichas tipografías.

Tipografías no instaladas

Imaginemos que tenemos el siguiente fragmento de código CSS:

```
.container {  
  font-family: Vegur, Georgia, sans-serif;  
}
```

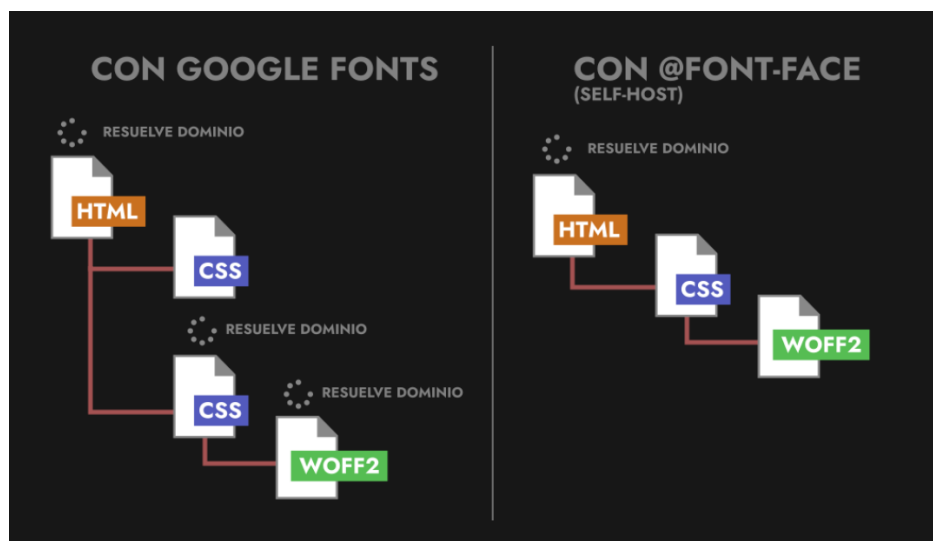
- Un usuario con la tipografía **Vegur** instalada, vería sin problemas el diseño ideal.
- Si no tiene la tipografía Vegur instalada en el sistema, la vería con la segunda tipografía: Georgia.
- Si tampoco tiene la tipografía Georgia instalada, buscará una tipografía de sistema «sin serifa».

Muchas tipografías tienen **derechos de autor** y no se deberían utilizar en nuestra página web si no hemos pagado su licencia. Antes de utilizar una tipografía, investiga que tipo de licencia tiene o si puedes utilizarla. Si tienes dudas, busca tipografías de **dominio público** o **CCO**.

¿Por qué no usar Google Fonts?

Aunque utilizar tipografías de Google Fonts con el código que nos proporciona Google es una opción viable, **en temas de rendimiento web suele ser mucho peor**.

Cuando utilizamos el código que nos proporciona Google Fonts, realmente, también usamos la regla **@font-face**, lo que ocurre es que el camino es «más largo» por lo que nuestro navegador tiene que hacer varias cosas:



- **1** Buscar (resolver) el dominio fonts.googleapis.com
- **2** Descargar y procesar el CSS desde fonts.googleapis.com
- **3** Buscar (resolver) el dominio fonts.gstatic.com (donde están las fuentes alojadas)
- **4** Descargar y procesar las fuentes de dicho dominio

Sin embargo, si descargamos la tipografía a nuestro sitio web, no tendrá que gastar tiempo y recursos en resolver esos dominios (el nuestro ya lo conoce), el CSS no se descarga desde otra URL porque ya está en nuestro código CSS principal, y descargará las tipografías desde nuestro sitio, algo que puede llegar a optimizarse si utilizas protocolos modernos como HTTP2 o superior.

Google fonts incorpora unas [etiquetas de precarga](#) para anticipar lo máximo posible la resolución del dominio, pero aún en ese caso, estamos realizando tareas extra que no se realizarían si alojamos las tipografías en nuestra propia web.

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 aunque no esté instalada en el sistema. Todo ello de forma transparente al usuario.

Observa el siguiente ejemplo de código donde precargamos **la tipografía Open Sans, utilizando la regla CSS @font-face**.

Esta regla suele colocarse al principio del fichero CSS para avisar al navegador que vamos a utilizar una tipografía que es muy posible que no se encuentre instalada.

En el ejemplo siguiente, lo hemos hecho con la tipografía Open Sans, **una fuente libre creada por Steve Matteson para Google**:

```
@font-face {
  font-family: 'Open Sans';
  font-style: normal;
  font-weight: 400;
  src:
    local("Open Sans"),
    url("/fonts/opensans.woff2") format("woff2"),
    url("/fonts/opensans.woff") format("woff"),
    url("/fonts/opensans.ttf") format("truetype");
}
```

Basicamente, abrimos un bloque `@font-face`, establecemos su nombre mediante `font-family` y, opcionalmente, definimos sus características mediante propiedades como `font-style` o `font-weight`.

El factor clave viene a la hora de indicar la tipografía a utilizar, que se hace mediante la propiedad `src`:

Valor	Significado	Soporte
<code>local("Nombre")</code>	¿Está la fuente instalada? Si es así, no hace falta descargarla, la usa.	✓ Todos
<code>url("file.woff2")</code>	Formato Web Open Font Format 2 . Mejora de WOFF con Brotli .	✓ Bueno
<code>url("file.woff")</code>	Formato Web Open Font Format . Es un TTF comprimido, ideal para web.	✓ Bueno
<code>url("file.ttf")</code>	Formato True Type / Open Type . .ttf o .otf. Para soportar navegadores antiguos.	✓ Bueno
<code>url("file.eot")</code>	Formato Embedded OpenType . Mejora de OTF, propietaria de Microsoft.	✗ Sólo IE
<code>url("file.svg")</code>	Tipografías creadas como formas SVG. No usar, considerada obsoleta.	✗ No usar

Es buena práctica usar la expresión `local()` seguida de los formatos WOFF2 y WOFF (*en dicho orden*), dando así soporte a la mayoría de navegadores, ya que WOFF tiene muy buen soporte. Para dar soporte a navegadores muy antiguos quizás podría ser necesario incluir también el formato TTF.

Observa que tras indicar la `url()` con el archivo con la tipografía, una buena práctica es indicar el formato de la tipografía, para que el navegador lo conozca antes de descargar la tipografía y anticiparse al soporte que podría tener ese navegador.

La sintaxis a utilizar sería la siguiente:

Sintaxis clásica	Sintaxis moderna	Significado
<code>format("woff2")</code>	<code>format(woff2)</code>	Formato .woff2
<code>format("woff")</code>	<code>format(woff)</code>	Formato .woff
<code>format("truetype")</code>	<code>format(truetype)</code>	Formato .ttf o .otf
<code>format("collection")</code>	<code>format(collection)</code>	Formato .otc o .ttc
<code>format("*-variations")</code>	<code>format(*) tech(variations)</code>	Formato .woff2, .woff, .ttf o .otf

La nueva sintaxis `tech()` aún no está soportada ampliamente en navegadores:

Chrome	Edge *	Safari	Firefox	Opera	IE ⚠ *	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mobile *	Browser for Android	Android Browser *
109							15.8				
122							16.7				
123	123		124				17.3				
124	124	17.4	125	107			17.4				
125	125	17.5	126	109	11	124	17.5	24	80	15.5	124
126		17.6	127				17.6				
127		TP	128								
128			129								

La regla `@font-face` permite cargar cualquier tipografía que queramos.

Modo de carga de las tipografías

Las tipografías que veremos en esta sección tratan de formas específicas de controlar la forma en la que se cargan y se ven o aparecen las tipografías en la página. Y para ello, hay que entender una serie de conceptos previos.

Modo de aparición

La **propiedad `font-display`** nos brinda un mecanismo muy interesante para determinar como queremos que se comporte la tipografía al cargar la página.

Cuando cargamos una página con una tipografía que no se encuentra instalada, el navegador debe descargar la tipografía y activarla, tarea que no ocurre instantáneamente, ya que depende de nuestra conexión, del tamaño de la fuente y del sistema utilizado.

En el instante anterior a descargar la tipografía pueden ocurrir varias cosas:

- **1 FOIT** (Flash of Invisible Text): El texto permanece **oculto** (invisible) al usuario hasta que la tipografía deseada se ha descargado y puede visualizarse.
- **2 FOUT** (Flash of Unstyled Text): El texto permanece **sin estilo** (con otra tipografía diferente) hasta que la tipografía deseada se ha descargado y puede visualizarse.

La propiedad `font-display`

Dependiendo del valor de la propiedad `font-display` podemos modificar dicho comportamiento:

Propiedad	Valor	Significado
<code>font-display</code>	<code>auto</code> <code>block</code> <code>swap</code> <code>fallback</code> <code>optional</code>	Modo en el que se muestra el texto cuando se cargan tipografías.

Como podemos ver, esta propiedad tiene varios valores que modifican su comportamiento.

Cada uno de ellos, va a indicarle al navegador como tiene que comportarse al cargar una tipografía externa, mediante la regla `@font-face`, y que tiene que hacer durante el tiempo que la tipografía no está disponible.

Veamos que ocurre con cada uno de los valores posibles a indicar a la propiedad `font-display` (en el momento que la tipografía no está descargada):

Valor	Descripción
auto	El navegador decide que método usar. Generalmente, usan block .
block	Fuente invisible ► Fuente deseada desde que cargue
swap	Fuente fallback ► Fuente deseada desde que cargue
fallback	Fuente fallback ► Fuente deseada si carga rápido, sino se queda fuente fallback.
optional	Si la fuente deseada carga rápido, usa fuente deseada, sino fuente fallback.

Estos valores son difíciles de resumir en una tabla, así que vamos a definirlos uno por uno, profundizando en sus características:

auto: Es el valor por defecto de la propiedad. El navegador decide que comportamiento aplicar, que suele ser FOIT (el texto aparece invisible).

En algunos navegadores o situaciones, el tiempo de FOIT puede llegar a ser considerablemente alto, y bastante molesto.

block: El navegador mantiene el texto invisible (FOIT) durante un corto periodo de tiempo (~3seg), mostrando la tipografía deseada desde que esté cargada. Este valor es ideal para pequeños fragmentos de texto como titulares, que no interesa que se vean con otra tipografía diferente.

swap: El navegador muestra inmediatamente el texto con la siguiente tipografía de la lista de alternativas durante un cortísimo periodo de tiempo (~100ms), mostrando la tipografía deseada desde que está cargada. Ideal para pequeños fragmentos de texto como titulares.

fallback: El navegador muestra el texto con la siguiente tipografía de la lista de alternativas de font-family durante un cortísimo periodo de tiempo (~100ms), mostrando la tipografía deseada desde que está cargada. Como se puede ver, es idéntico al valor swap. La diferencia es que si pasa demasiado tiempo sin que la tipografía se cargue, se utiliza de forma definitiva la alternativa que se estaba usando y no se utiliza la tipografía deseada. Puede ser apropiada para usar en fragmentos largos de texto.

optional: El navegador considera opcional el uso de la tipografía deseada si tarda demasiado o cree que hay algún problema en la descarga, utilizando la siguiente alternativa en la lista de tipografías de font-family. Ideal para textos largos que no produzcan saltos.

La propiedad unicode-range

Cuando cargamos tipografías con la regla **@font-face**, es una buena práctica utilizar la propiedad **unicode-range** para indicar cuales son los caracteres que utilizaremos de la tipografía.

Una tipografía cuenta con multitud de caracteres o glifos: caracteres alfanuméricos, caracteres básicos ASCII, caracteres hebreos, caracteres japoneses, caracteres especiales, etc...

Sin embargo, hay ciertos caracteres que al menos en algunas situaciones no vamos a utilizar, por lo que sería un desperdicio descargarse la tipografía para luego no utilizarla.

La propiedad **unicode-range** nos permite indicar el rango que vamos a utilizar con una tipografía, y en el caso de que la página o documento no necesite ningún carácter de ese rango, la tipografía no se descargará.

Propiedad	Valor	Descripción
unicode-range	U+0-10FFFF RANGE	Indica el rango unicode a utilizar.

Los rangos unicode se definen utilizando el **prefijo U+ o u+**, seguido del **número del rango en hexadecimal**. Los rangos con esta propiedad se pueden indicar de la siguiente forma:

- 1 **Un carácter individual**: Indicamos sólo un carácter concreto. Por ejemplo **U+0042** o **U+01FF**.
- 2 **Un rango**: Indicamos un rango de caracteres. Por ejemplo **U+0042-U+01FF**, que son los caracteres desde **42** hasta **1FF**.
- 3 **Un rango con comodines**: Indicamos un rango con comodines. Por ejemplo **U+00??**, que abarcaría todos los caracteres desde **00** hasta **FF**.

Veamos un fragmento de código donde tenemos un ejemplo de uso de esta propiedad:

```
@font-face {
  font-family: "Virgil 3 YOFF";
  src:
    local("Virgil 3 YOFF"),
    url("/fonts/virgil-3.woff2") format("woff2"),
    url("/fonts/virgil-3.woff") format("woff"),
    url("/fonts/virgil-3.ttf") format("truetype");
  unicode-range: U+000-U+0FF;
}
```

La herramienta [FontForge](#) es capaz de cargar tipografías y mostrarnos los diferentes rangos unicode de cada tipografía y que elementos se usan en cada uno de ellos, por si queremos examinar detalladamente. En [SYMBL](#) o la aplicación **Mapa de caracteres** de Windows, también tienes los bloques que se suelen utilizar.

COMBINADORES

Al margen de los selectores básicos de CSS, elementos, clases, id e incluso atributos, existe una amplia gama de formas de crear selectores más complejos que permitan seleccionar elementos HTML de una forma más potente y flexible.

¿Qué es un Combinador CSS?

Un combinador CSS es un símbolo que permite combinar dos o más selectores CSS, formando uno más complejo y potente. Existen varios combinadores en CSS, que mostraremos en la siguiente tabla:

Nombre	Símbolo	Ejemplo	Significado
Combinador descendiente	(espacio)	<code>#page div { }</code>	Selecciona elementos dentro de otros (cualquier nivel).
Combinador hijo	<code>></code>	<code>#page > div { }</code>	Selecciona elementos hijos directos (primer nivel).
Combinador hermano adyacente	<code>+</code>	<code>div + div { }</code>	Selecciona elementos contiguos a otros (mismo nivel).
Combinador hermano general	<code>~</code>	<code>div ~ div { }</code>	Selecciona elementos que siguen a otros (mismo nivel).
Combinador universal	<code>*</code>	<code>#page * { }</code>	Selecciona todos los elementos (cualquier nivel).

En los siguientes apartados, veremos varios ejemplos gráficos sobre un supuesto ejemplo de documento HTML, dibujado en forma de árbol esquemático. Así sabremos que elementos están dentro de otros y nos será más fácil entender cada uno de los combinadores CSS.

Combinador descendiente

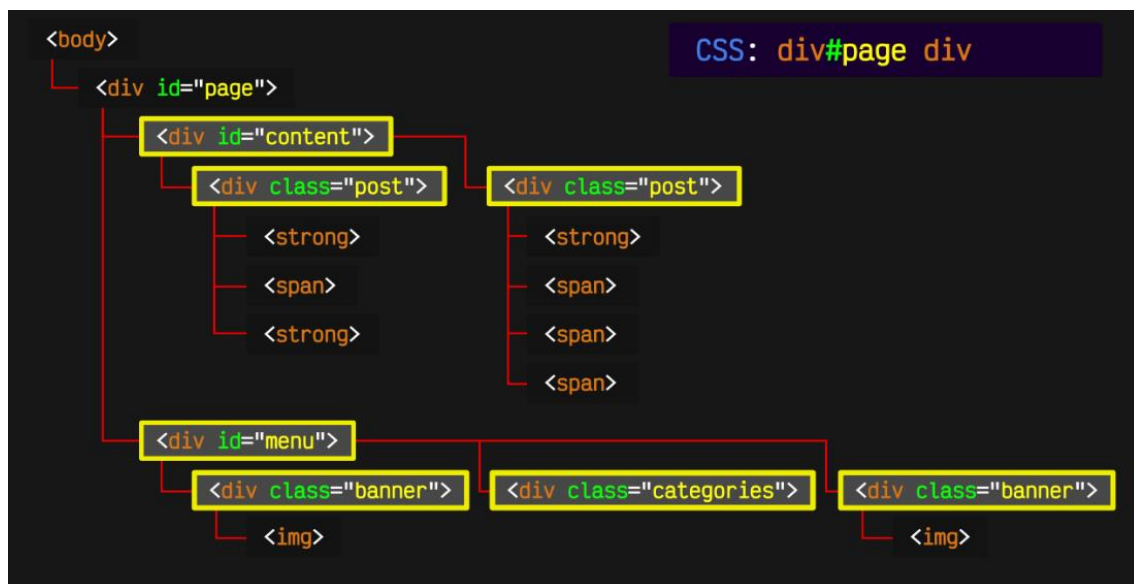
En CSS podemos utilizar lo que se llama el selector descendiente, que no es más que una forma de seleccionar ciertos elementos que están dentro de otros elementos. Esto puede parecer sencillo, pero cuidado, ya que puede ser una fuente de problemas si no se comprende bien.

Su sintaxis se basa en colocar los elementos uno a continuación de otro, separado por un espacio. Ese espacio hace de combinador con los demás selectores:

```
div#page div {  
  background-color: gold;  
}
```

En el ejemplo anterior, aplicamos los estilos CSS (color gold de fondo) a todos los elementos `<div>` que estén dentro del elemento `<div id="page">`. De esta forma, si existe un elemento `<div>` fuera del elemento con `id="page"`, no se aplicarán los estilos indicados.

Observa la siguiente estructura HTML y observa las líneas rojas que marcan que elementos están dentro de otros, para así ser conscientes de en qué nivel se encuentran:



Repasemos varios detalles importantes respecto a este combinador CSS:

- Se están seleccionando todos los elementos `<div>` que están dentro del elemento con `id="page"`.
- Observa que se seleccionan independientemente del nivel al que estén (hijos, nietos, ...)
- En este caso, el `div` de `div#page` es innecesario, ya que habíamos dicho que los `id` no se pueden repetir. Si ya existe un elemento con `id page`, no hace falta diferenciarlo también por etiqueta. Si se tratase de una clase, si podría usarse.

Se pueden construir selectores muy complejos con tantos elementos como se quiera, pero una buena práctica es intentar mantenerlos siempre lo más simples posibles. Cuántos más elementos descendientes existan en un selector, más complejo será el procesamiento de dicha regla por los navegadores y más específico será el navegador, lo que puede introducir algunas complicaciones.

Lo recomendable es utilizar buenos nombres, que sean simples, y que sean lo bastante claros para entenderlos rápidamente:

```
<div class="menu">
  <div class="options">
    <ul>
      <li><a href="/one">Option 1</a></li>
      <li><a href="/two">Option 2</a></li>
      <li><a href="/three">Option 3</a></li>
    </ul>
  </div>
</div>
```

Observando el fragmento de código HTML anterior, veamos dos formas de aplicar estilos CSS a los enlaces `<a>` del fragmento de código:

```
/* Forma 1: Muy específica */
.menu .options ul li a {
  color: orange;
}

/* Forma 2: Más general, menos específica */
.menu a {
  color: orange;
}
```

Mientras que la primera es mucho más específica, es una muy buena práctica en CSS mantener los selectores lo menos específicos posibles para evitar problemas de Especificidad

Combinador hijo >

Aunque el selector descendiente es bastante interesante, nos puede interesar hacer la misma operación, pero en lugar de seleccionar todos los elementos descendientes, seleccionar sólo los descendientes directos del elemento con el símbolo `>`, descartando así nietos y sucesivos.

```
#page > div {
  background-color: gold;
}
```

Veamos los elementos seleccionados en el documento de ejemplo para afianzar conceptos:



Al contrario que en el caso anterior, no se seleccionan todos los elementos `<div>` descendientes, sino solo aquellos que son hijos directos del primer elemento especificado.

Combinador hermano contiguo +

Es posible también hacer referencia a los elementos hermanos, es decir, aquellos elementos que están directamente a continuación del elemento especificado.

Mediante el símbolo **+** del selector hermano adyacente, se pueden seleccionar aquellos elementos hermanos que están seguidos el uno de otro (en el mismo nivel):

```
div.post span + span {  
  background-color: gold;  
}
```

Cómo se podrá ver en este nuevo ejemplo, este combinador CSS hará que se seleccionen los elementos **span** que estén a continuación de un **span** dentro de un **div** con **class .post**:



Obsérvese que el primer elemento **** no es seleccionado, puesto que es el que estamos tomando de referencia de base. Una buena forma para entenderlo es leerlo de la siguiente forma: «todo elemento **** que esté inmediatamente precedido de un ****».

Combinador hermano general ~

- Si pensamos otras opciones en el ejemplo anterior, es posible que necesitemos ser menos específicos y en lugar de querer seleccionar los elementos hermanos que sean adyacentes, queramos seleccionar todos los hermanos en general, sin necesidad de que sean adyacentes.
- Esto se puede conseguir con el selector hermano general, simbolizado con el carácter **~**:



Como se ve en el ejemplo, no es necesario que el elemento `strong` se encuentre adyacente al primero, sino que basta con que esté a continuación y sean hermanos en el mismo nivel.

Combinador universal *

Por último, el selector universal se simboliza con un asterisco `*` y es la forma de aplicar ciertos estilos en TODOS Y CADA UNO de los elementos HTML correspondientes.



Este ejemplo selecciona todos los elementos dentro de `div#menu`. Es importante recalcar la diferencia de seleccionar `#menu`, a seleccionar todos los elementos dentro de `#menu`, que es lo que estamos haciendo en este caso.

El selector universal puede ser muy útil en algunos casos para resetear ciertas propiedades de todo un documento, como en el siguiente ejemplo, donde se eliminan los márgenes de todos

los elementos del documento HTML, puesto que algunos navegadores ponen márgenes diferentes y esto puede producir ciertas inconsistencias en los diseños:

```
/* Elimina márgenes y rellenos de todos los elementos de un documento HTML */
* {
  margin: 0;
  padding: 0;
}
```

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 crear agrupaciones con diferentes selectores separando por comas.

Agrupación de selectores

- Imagina una situación en la que varios bloques de código CSS contienen las mismas propiedades con los mismos valores.
- Generalmente, escribir cada bloque de forma individual no es apropiado, ya que duplica un código que es exactamente igual:

```
.container-logo {
  border-color: red;
  background: white;
}

.container-alert {
  border-color: red;
  background: white;
}

.container-warning {
  border-color: red;
  background: white;
}
```

Si esto ocurre a menudo, el tamaño del documento CSS será más grande y tardará más en descargarse.

Una buena práctica para evitarlo es ahorrar texto y simplificar nuestro documento CSS lo máximo posible, por lo que podemos hacer uso de la agrupación CSS **utilizando el símbolo de la coma**.

De esta forma, podemos pasar de tener el ejemplo anterior, a tener el siguiente ejemplo (que es totalmente equivalente), donde hemos utilizado la agrupación para decirle al navegador que aplique dichos estilos a las diferentes clases:

```
.container-logo, .container-alert, .container-warning {  
  border-color: white;  
  background: red;  
}
```

Al margen de esto, dos buenas prácticas que podríamos aplicar en esta situación serían las siguientes:

- **Simplifica por responsabilidades:** .container-alert y .container-warning parecen tener un concepto muy similar: alertas o mensajes de advertencia.

Es posible que estos selectores tengan la misma funcionalidad y sean sinónimos. Si es así, lo ideal sería refactorizar y simplificarlos a uno: .container-warning, haciendo desaparecer el otro.

- **Legibilidad por delante:** El código CSS por si sólo puede ser difícil de leer y mantener.

Aunque a priori puede parecer que es mejor escribir la lista de selectores uno detrás de otro, la experiencia nos dicta que **deberíamos separarlos en una línea diferente cada selector. Esto lo hace mucho más legible a la hora de leer**.

```
.container-logo,  
.container-warning {  
  border-color: white;  
  background: red;  
}
```

Estos consejos pueden parecer poco importantes, pero a medida que avanzamos con nuestro diseño y escribimos más código CSS, este se hace muy grande y difícil de mantener, por lo que cuanto más sencillo lo mantengamos, mejor.

¿Qué es un combinador lógico?

Sin embargo, las comas sólo son la forma más sencilla y simple de reutilizar selectores.

En CSS, tenemos a nuestra disposición **una serie de mecanismos para agrupar o combinar selectores de una forma más potente y flexible, dentro de una categoría denominada combinadores lógicos.**

Estos **combinadores lógicos** nos permiten seleccionar elementos con ciertas restricciones y **funcionan como una pseudoclase**, sólo que se le pueden pasar parámetros, ya que son de **tipo pseudoclase funcional**.

Observa el siguiente fragmento de código CSS donde utilizamos **el combinador lógico :is()**:

```
:is(.container-logo, .container-warning) {  
  border-color: white;  
  background: red;  
}
```

Este fragmento es sólo un ejemplo simplificado para comprender fácilmente la sintaxis de un combinador lógico.

Tipos de combinadores lógicos

A continuación, una tabla donde podemos ver qué mecanismos de combinadores lógicos tenemos a nuestra disposición:

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

Como hemos visto en apartados anteriores, los combinadores lógicos son un tipo de selector que permite combinar varios selectores más simples. Utiliza una estructura funcional, de modo que se le puede colocar selectores por parámetro.

Esencialmente, tenemos dos combinadores lógicos principales: `:is()` y `:where()`. Vamos a empezar por el primero, ya que el segundo funciona exactamente igual con algunas diferencias leves.

El combinador `:is()`

La pseudoclase funcional `:is()` es 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 otros selectores con los pasados por parámetro a `:is()`.

Vamos con el primer ejemplo. El siguiente selector agrupa 3 partes diferentes:

```
.container .list,  
.container .menu,  
.container ul {  
    /* ... */  
}
```

Si nos fijamos bien, **la clase `.container` siempre aparece en cada uno de los tres casos**, sin embargo, no hay forma de abreviarla, aunque sólo cambie la última parte.

Por ejemplo, quizás podríamos intentarlo con el siguiente fragmento, sin embargo, **NO ES EQUIVALENTE** al fragmento de código anterior:

```
.container .list, .menu, ul {  
    /* ... */  
}
```

En este caso, estamos indicado los **elementos `.list` que se encuentren dentro de `.container`**, y además los **elementos `.menu` y los elementos `ul`**, estos dos últimos aunque no estén dentro del `.container`. Esto se acentúa cada vez más si el selector es más largo.

Con la pseudoclase `:is()` si que podemos abreviar la información repetida del ejemplo anterior, reescribiéndolo de la siguiente forma:

```
.container :is(.list, .menu, ul) {  
    /* ... */  
}
```

Observa que hemos indicado los 3 casos iniciales en un sólo selector (que añade 3 posibilidades diferentes por parámetro). Esto nos permite crear código mucho más compacto y sencillo de leer y escribir.

Antiguamente, esta pseudoclase era conocida como `:matches()`, pero finalmente fue renombrada a `:is()`, por lo que es posible que nos la encontremos de esta forma si accedemos a documentación antigua.

Especificidad de :is()

Mucho cuidado con la especificidad CSS, ya que no tiene por qué ser equivalente a la versión inicial que separábamos con comas.

Observa el siguiente ejemplo inicial:

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

Con la pseudoclase :is(), se calcula la especificidad sumando la especificidad más alta de sus parámetros.

Para el selector .container .list la especificidad sería (0,2,0) tanto usando :is() como no usándolo, sin embargo, en el caso del selector .container ul, en el primer caso sin :is(), la especificidad sería (0,1,1), mientras que con :is() seguiría siendo (0,2,0):

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

¿Cómo se calcula la Especificidad?

El navegador tiene un sistema llamado Especificidad CSS, donde en situaciones de conflicto como la anterior, calculará que selector es más específico, siguiendo unas ciertas normas, y obteniendo como resultado un valor numérico.

Dicho valor numérico se suele representar con 3 cifras, separadas por comas: A,B,C:

Valor Descripción

Valor A Número de veces que aparece un #id en el selector.

Valor B Número de veces que aparece una .clase, [atributo] o :pseudoclase en el selector.

Valor C Número de veces que aparece un elemento o un ::pseudoelemento en el selector.

Teniendo en cuenta esto, veamos algunos ejemplos con diferentes selectores CSS y calculemos su cifra de especificidad CSS. Recuerda que cuanto más alta sea, más específico es el selector y mayor prioridad de que gane sobre otros selectores en conflicto:

```
div { ... } /* 0,0,1 (1 elemento HTML) */
div div { ... } /* 0,0,2 (2 elementos HTML) */
#pagina div { ... } /* 1,0,1 (1 id y 1 elemento HTML) */
#pagina div:hover { ... } /* 1,1,1 (1 id, 1 pseudoclase y 1 elemento HTML) */
#pagina div:hover a { ... } /* 1,1,2 (1 id, 1 pseudoclase y 2 elementos HTML) */
#pagina .sel:hover>a { ... } /* 1,2,1 (1 id, 1 clase, 1 pseudoclase y 1 elemento HTML) */
```

Excepciones de Especificidad CSS

Recuerda también que hay ciertas excepciones a la hora de calcular la especificidad:

Excepción	Cómo se modifica su especificidad
1 Combinadores <code>:is()</code> , <code>:not()</code> o <code>:has()</code>	La especificidad del selector más específico en el interior de los paréntesis.
2 Pseudoclases <code>:nth-child()</code> o <code>:nth-last-child()</code>	Se suma la especificidad de la pseudoclase más el selector más específico.
3 La pseudoclase <code>:where()</code>	Se reemplaza su especificidad por 0.

El combinador `:where()`

Por otro lado, existe otro combinador lógico denominado `:where()`, que funciona exactamente igual que el combinador `:is()`.

La única diferencia que tiene es en cuanto a la especificidad CSS.

Mientras que con el combinador `:is()`, la especificidad es el valor más alto de la lista de parámetros de `:is()`, **en el caso de `:where()` la especificidad CSS es siempre cero.**

Veamos un ejemplo para clarificarlo:

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

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

Observa que en el caso de `:where()`, sólo se suma la especificidad de los selectores que están fuera del `:where()`.

El combinador `:where()` puede ser útil para casos en los que se quiere anular la especificidad de un elemento fácilmente si se sobrescribe con otro selector. Esto lo hace especialmente interesante para crear unos estilos con especificidad muy baja que posteriormente van a ser sobrescritos y no queremos que la especificidad lo evite o lo vuelva complejo de sobrescribir.

El combinador o selector `:not()`

El combinador, selector o 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. Esto puede simplificar mucho algunos fragmentos de CSS.

Veamos un sencillo ejemplo:

```
p:not(.main) {  
  border: 2px solid black;  
  padding: 8px;  
  color: white;  
  background: indigo;  
}
```

```
<p class="first">Hello, my first paragraph.</p>  
<p class="main">Again, my main paragraph.</p>  
<p class="last">Bye, my last paragraph.</p>
```

Hello, my first paragraph.

Again, my main paragraph.

Bye, my last paragraph.



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

Selectores complejos en `:not()`

Antiguamente, sólo se permitían selectores simples en el combinador `:not()`. Sin embargo, actualmente los navegadores permiten indicar listas de selectores o selectores complejos.

Observa el siguiente ejemplo. Aunque no tiene demasiado sentido, se puede ver como los selectores complejos son aceptados por parámetro del combinador `:not()`:

```
p:not(:is(.first, .last)) {  
  border: 2px solid black;  
  padding: 8px;  
  color: white;  
  background: indigo;  
}
```

```
<p class="first">Hello, my first paragraph.</p>
<p class="main">Again, my main paragraph.</p>
<p class="last">Bye, my last paragraph.</p>
```

Hello, my first paragraph.

Again, my main paragraph.

Bye, my last paragraph.



En este caso, tenemos tres elementos y le hemos dicho que le de estilo a los elementos que:

- Sean párrafos <p>
- No tengan la clase .first
- No tengan la clase .last

Como resultado, se le da estilo sólo al elemento <p> con clase .main, que es la única que cumple la restricción.

Selectores :not() encadenados

- Otra forma de escribir selectores complejos utilizando combinadores :not() es encadenando uno con otro.
- De esta forma, podemos conseguir cosas similares a los casos anteriores.

Veamos el ejemplo anterior, reescribiéndolo con :not() encadenados:

```
p:not(.first):not(.last) {
  border: 2px solid black;
  padding: 8px;
  color: white;
  background: indigo;
}
```

```
<p class="first">Hello, my first paragraph.</p>
<p class="main">Again, my main paragraph.</p>
<p class="last">Bye, my last paragraph.</p>
```

Hello, my first paragraph.

Again, my main paragraph.

Bye, my last paragraph.



En este caso, estamos dando estilo a los elementos que:

- Sean párrafos <p>
- No tengan la clase .first
- No tengan la clase .last

Como puedes ver, equivalente al caso anterior.

Detalles importantes sobre :not()

Algunos detalles adicionales (e importantes) sobre la pseudoclase funcional :not():

El combinador :not() se puede anidar dentro de otro :not().

El combinador :not() **no acepta pseudoelementos, como ::before o ::after, por parámetro.**

Al igual que con :is(), **la especificidad de :not() es el valor más alto de sus parámetros.**

Combinador o selector :has()

- Probablemente, **uno de los combinadores o selectores más potentes es :has()**.
- En CSS, cuando damos estilo, el elemento objetivo al que se le aplica el estilo es siempre el último que se escribe en el selector.

Por ejemplo, en este caso el elemento a quién se le da estilo es a .element, siempre y cuando cumpla el resto de restricciones, que en este caso es que se encuentre dentro de un elemento .container:

```
.container .element {  
  background: red;  
}
```

Sin embargo, con el combinador :has() esto se puede cambiar.

El combinador :has()

El combinador o pseudoclase :has() **permite seleccionar un elemento contenedor, siempre y cuando sus elementos hijos (descendientes) cumplan los criterios indicados por los parámetros de :has(), lo que comúnmente siempre se ha denominado el selector padre.**

```

a {
  /* ... */
}

a:has(> img) {
  /* ... */
}

```

- 1 En el primer caso, aplicamos estilos a TODOS los enlaces <a>.
- 2 En el segundo caso, aplicamos estilos a todos los enlaces <a> que contengan una imagen .

Sin embargo, recuerda que podemos añadir más selectores antes, después o el interior del :has(), consiguiendo criterios más complejos y potentes.

Selector padre con :has()

El fragmento de código anterior puede resultar un poco confuso, así que veámoslo aplicado a un ejemplo real en el que tenemos 3 enlaces: los dos primeros contienen una imagen y el tercero y último, contiene un texto:

```

<div class="container">
  <a href="https://manz.dev/"></a>
  <a href="https://manz.dev/"></a>
  <a href="https://manz.dev/">https://manz.dev/</a>
</div>

```

```

img {
  width: 64px;
  height: 64px;
}

a {
  border: 3px solid black;
  padding: 5px;
}

a:hover {
  border-color: blue;
  color: blue;
}

a:hover:has(> img) {
  border-color: red;
}

```



Si nos fijamos en el CSS, **en `a:hover` indicamos que cuando se mueva el ratón sobre el enlace:**

- Se aplique un color azul en el borde.
- Se aplique color de texto azul.
- Sin embargo, en el selector `a:hover:has(> img)` indicamos que cuando se mueva el ratón sobre un enlace que contenga una imagen:
 - Se aplique un color rojo en el borde.

Selector cuantificador `:has()`

Sin embargo, `:has()` es más potente de lo que a simple vista parece, ya que se puede combinar con otros selectores de forma que se consigan funcionalidades aún más potentes.

Por ejemplo, utilizando `:has()` junto a funciones como `:nth-child()` se puede contabilizar el número de elementos hijos que tiene al menos una cantidad de elementos:

```
.container {
  display: flex;
  gap: 10px;
  background: grey;
  margin: 10px;
}

.item {
  width: 50px;
  height: 50px;
  background: #222;
  border: 2px solid black;
}

.container:has(:nth-child(3 of .item)) {
  background: indigo;
}
```

```

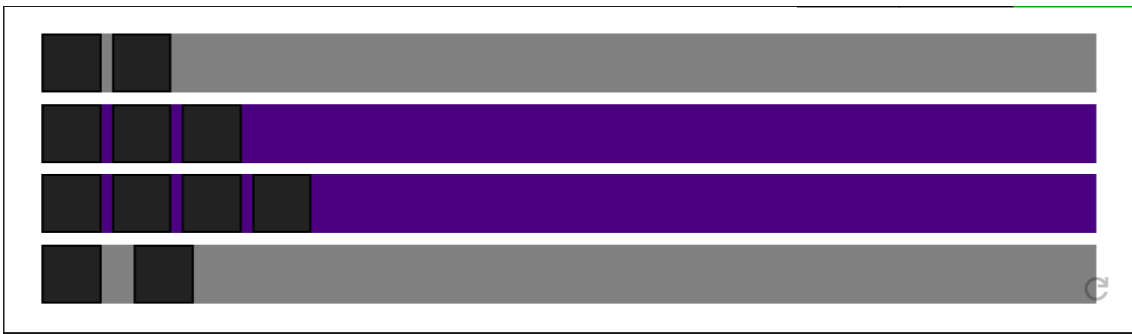
<div class="container">
  <div class="item"></div>
  <div class="item"></div>
</div>

<div class="container">
  <div class="item"></div>
  <div class="item"></div>
  <div class="item"></div>
</div>

<div class="container">
  <div class="item"></div>
  <div class="item"></div>
  <div class="item"></div>
  <div class="item"></div>
</div>

<div class="container">
  <div class="item"></div>
  <div class="noitem"></div>
  <div class="noitem"></div>
  <div class="item"></div>
</div>

```



En este ejemplo, nos centraremos en los 3 primeros contenedores de elementos: Uno con 2 items, otro con 3 items y otro con 4 items.

Utilizando el selector `.container:has(:nth-child(3 of .item))` estamos seleccionando:

- Un elemento padre con clase `.container`
- Que tenga al menos 3 elementos hijos con clase `.item`

Observa que si al cuarto y último grupo, le cambiáramos los elementos hijos, de modo que queden dos `.item` y dos `.noitem`, este grupo contenedor no tendría estilo, ya que no cumple los criterios de tener al menos 3 elementos con clase `.item`.

Selector de estados `:has()`

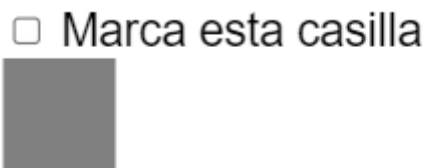
Quizás el detalle más interesante del combinador o selector `:has()` es que lo podemos utilizar para controlar estados de ciertos elementos de la página.

Para ello, podemos utilizar pseudoclasas como `:checked` o `:hover`, por ejemplo.

El siguiente ejemplo permite cambiar el diseño de un elemento, dependiendo de una casilla `<input>` que no tiene relación directa:

```
.item {  
  width: 50px;  
  height: 50px;  
  background: grey;  
}  
  
html:has(input:checked) .item {  
  background: indigo;  
}
```

```
<label>  
  <input type="checkbox"> Marca esta casilla  
</label>  
  
<div class="container">  
  <div class="item"></div>  
</div>
```



Detalles importantes sobre :has()

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.