Guía HTML / CSS

HTML / CSS

Introducción al CSS

¿De qué esta hecha la web?

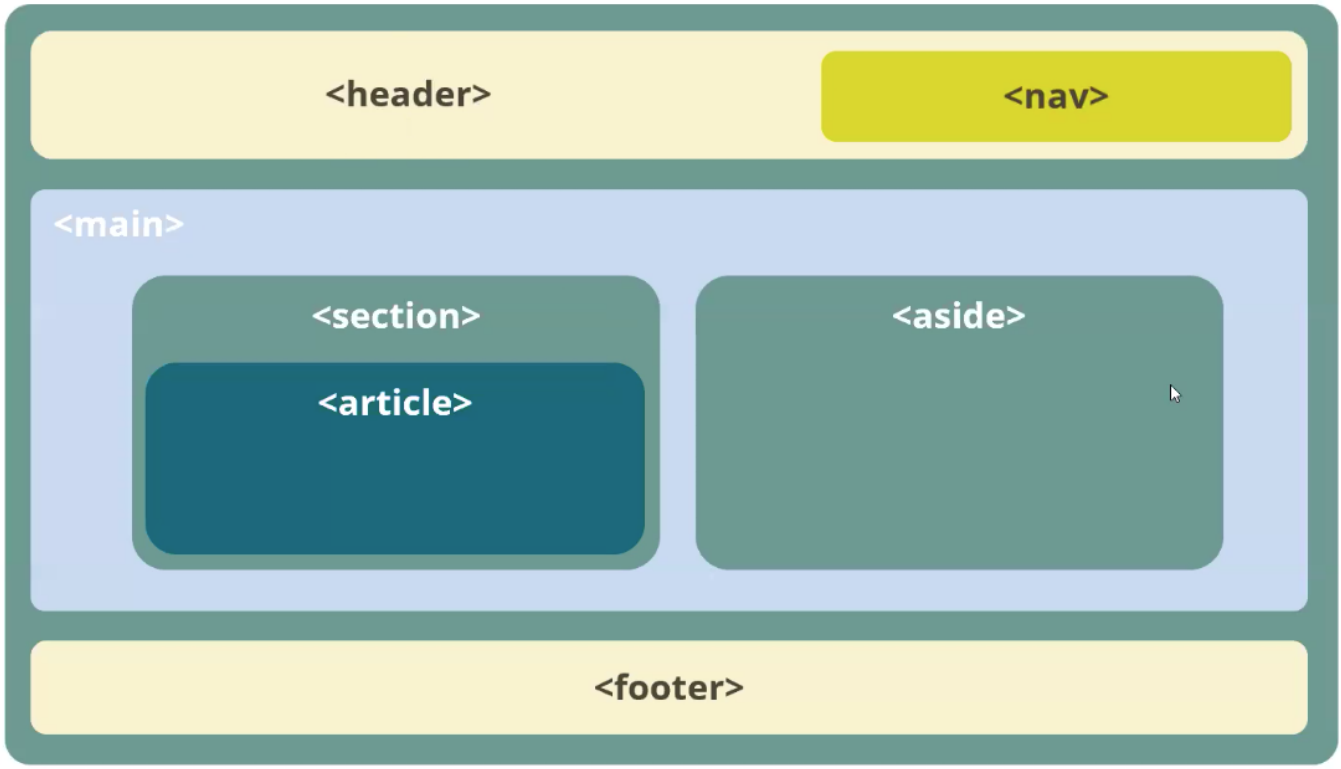
Las aplicaciones web están compuestas de HTML, CSS, y Javascript:

- Con el HTML definimos el contenido de una página (que controles vamos a poner, que textos...).
- Con el CSS definimos el estilo (colores, fuentes, tamaños, ...)
- Con el JavaScript definimos el comportamiento de la aplicación (por ejemplo como nos traemos datos de un servidor, como validamos que un campo esté informado y tenga un formato dado, etc...)

Sintaxis básica

Estructura HTML y semántica

Con la entrada de la versión 5 de HTML se introdujo una estructura semántica, en la MDN (Mozilla Developer Network) podemos encontrar unas definiciones muy útiles:



- **Header:** representa un grupo de ayudas introductorias o de navegación, puede contener algunos elementos de encabezado, así como también un logo, un formulario de búsqueda, un nombre de autor y otros componentes.
- **Nav:** el elemento *<nav>* representa una sección de una página cuyo propósito es proporcionar enlaces de navegación, ya sea dentro del documento actual o a otros documentos. Ejemplos comunes de secciones de navegación son: menús, tablas de contenido e índices.
- **Main:** representa el contenido principal del *<body>* de un documento o aplicación. No debe de haber más de un elemento *<main>* en un documento y este no debe ser descendiente de un elemento *<article>*, *<aside>*, *<footer>*, *<header>* o *<nav>*.
- **Section:** El elemento de HTML section (<section>) representa una sección genérica de un documento. Sirve para determinar qué contenido corresponde a qué parte de un esquema. Piensa en el esquema como en el índice de contenido de un libro; un tema común y subsecciones relacionadas. No se debe usar el elemento <section> como un mero contenedor genérico; para esto ya existe , especialmente si el objetivo solamente es aplicar un estilo (CSS) a la sección.
- **Article:** El Elemento article de HTML (<article>) representa una composición auto-contenida en un documento, página, una aplicación o en el sitio, que se destina a distribuir de forma independiente o reutilizable, por ejemplo, en la indicación. Podría ser un mensaje en un foro, un artículo de una revista o un periódico, una entrada de blog, un comentario de un usuario, un widget interactivo o gadget, o cualquier otro elemento independiente del contenido.
- **Aside:**: representa una sección de una página que consiste en contenido que está indirectamente relacionado con el contenido principal del documento. Estas secciones son a menudo representadas como barras laterales o como inserciones y contienen una explicación al margen como una definición de glosario, elementos relacionados indirectamente, como publicidad, la biografía del autor, o en aplicaciones web, la información de perfil o enlaces a blogs relacionados.

- **Footer:**: representa un pie de página para el contenido de sección más cercano o el elemento raíz de sección (p.e, su ancestro mas cercano *<article>*, *<aside>*, *<nav>*, *<section>*, *<blockquote>*, *<body>*, *<details>*, *<fieldset>*, *<figure>*, *<td>*). Un pie de página típicamente contiene información acerca de el autor de la sección, datos de derechos de autor o enlaces a documentos relacionados.

Si usamos correctamente estas etiquetas y añadimos el contenido adecuado, esto nos va a servir de gran ayuda para que nuestra página se accesible (que por ejemplo una persona invidente lo tenga más fácil para poder entender el contenido de nuestro sitio), y se posicione en buscadores (SEO, un buscador tiene claro que contenido buscar en cada sección).

CSS

CSS es el acrónimo de **Cascading Style Sheets** hojas de estilo en cascada.

Nos permite estilar y maquetar páginas web, nos permite modificar aspectos tales como la fuente, el color o tamaño de un contenido, organizarlo en columnas, añadir animaciones...

Este CSS después se aplica a páginas HTML, esto nos permite separar estilado de contenido y además poder, por ejemplo, realizar cambios de imagen en un sitio web sin tener que ir tocando página por página.

Los navegadores web aplican los cambios que definimos en los CSS (también conocido como hojas de estilo) a través de *reglas css*. Un tema importante a tener en cuenta es que, el resultado de aplicar estas reglas no tiene porque ser homogéneo, hay navegadores que pueden tener diferentes formas de interpretarlo (sobre todo si hablamos de navegadores antiguos como es *Internet Explorer*).

Vamos a definir lo que es una reglas CSS:

Una regla CSS es un conjunto de propiedades con sus correspondiente valores, todas ellas agrupadas bajo un selector

Vamos a diseccionar esta definición:

- Una propiedad es por ejemplo el color de un texto, o la fuente, o si queremos ponerle un margen a un elemento, a estas propiedades podemos asignarle valores.
- Un selector identifica un elemento HTML de una página web (un body, un párrafo, una imagen, una tabla...) un selector está formado por propiedades que me permite cambiar el aspecto de un elemento, este selector más adelante podemos aplicarlo al HTML que tengamos.

Así pues las hojas de estilo son el conjunto de reglas CSS que afectan a una página web.

Si estilamos usando CSS en vez de poner a fuego los estilos en el HTML, tenemos una serie de ventajas:

- Separamos responsabilidades, por un lado en el HTML nos podemos centrar en mostrar el contenido, por otro en el CSS como queremos presentarlo (colores, fuentes, layout...). Esto hace que el código sea más mantenible, ya que entre otras ventajas reducimos la duplicidad y el "copia/pega", imagínate que tienes un tipo de lista definido y lo quieres usar para todas las páginas de tu sitio... si tuvieras que ir copiando el estilado en cada una de las listas donde lo usas sería un engorro, más si un día te piden cambiar el estilo de las mismas, aplicando reglas de estilo sólo tendrías que hacerlo en un solo sitio.

- Podemos crear versiones para distintos dispositivos, sea en el mismo fichero CSS o separados, podemos añadir media queries, que permiten maquetar de una forma para, por ejemplo, escritorio y o móvil, sin tener que tocar el contenido.

Historia del CSS

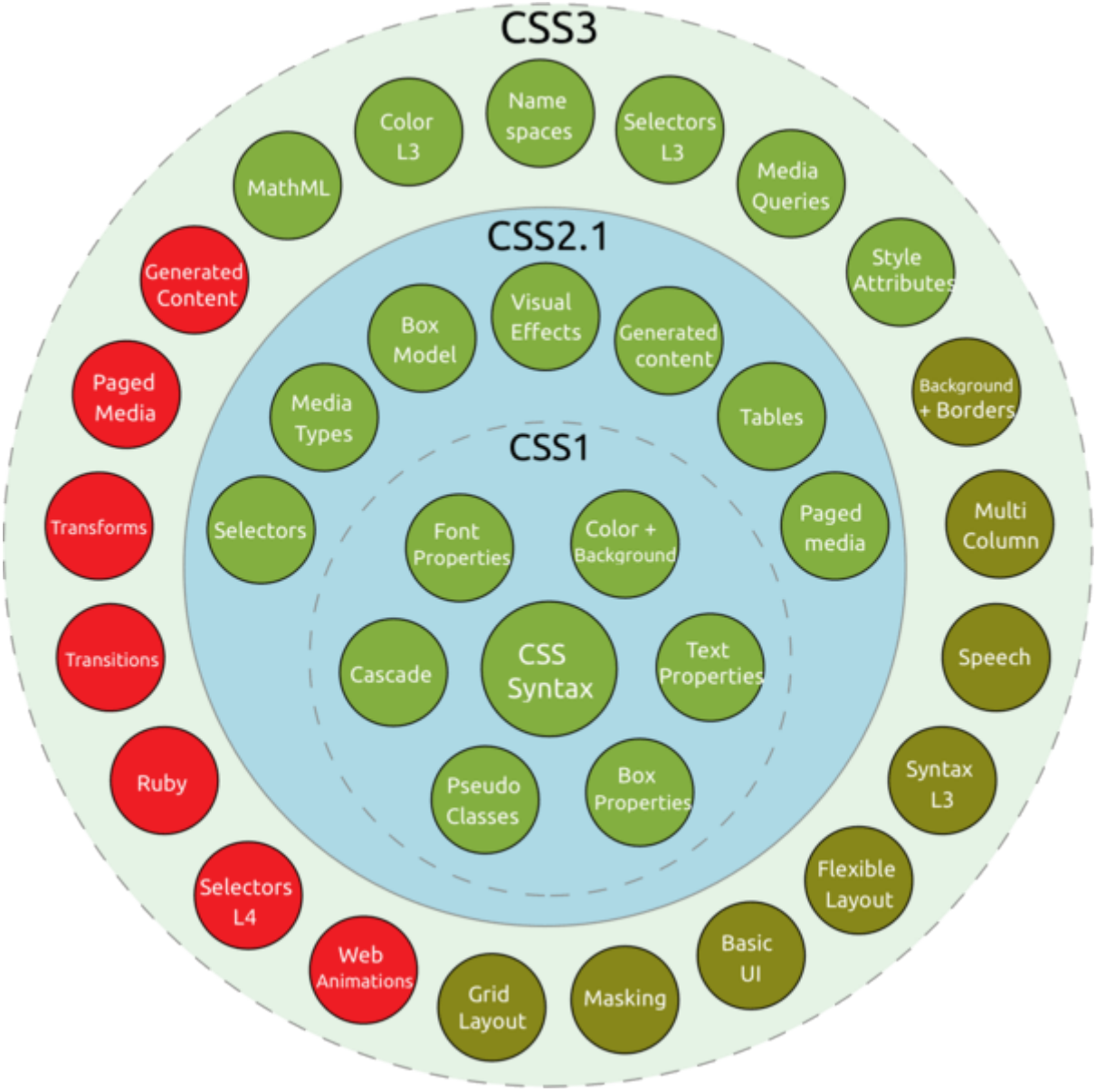
El estándar CSS está definido por el W3C (World Wide Web Consortium), en la redacción de este estándar participan sus entidades miembro, miembros del equipo de W3C, expertos invitados, e incluso cualquier usuario que desee mostrar su opinión.

El avance de estos estándares es lento si lo comparamos con un estándar Ad Hoc (un estándar Ad Hoc es una tecnología / librería / etc... que define una empresa o un grupo de desarrolladores), por otro lado, nos garantiza que en un tiempo razonable todos los navegadores van a ofrecer unas características similares.

Versiones de CSS:

- CSS 1 (1996): podemos encontrar propiedades para definir fuentes, colores, alineación de elementos...
- CSS 2 (1998): podemos encontrar propiedades de posicionamiento, tipos de medios, entre otras características.
- CSS 3 (2011): Inicio de módulos separado con funcionalidades nuevas (niveles de versiones).

¿Y CSS 4? No existe, ni existirá, se ha cambiado la forma de versionar, ahora cada módulo ira evolucionando de forma independiente, así su implementación es más sencilla y pueden evolucionar y ser actualizados por separado



Selectores

Como hemos comentado antes, un selector identifica un elemento HTML de una página web (un body, un párrafo, una imagen, una tabla...) un selector está formado por propiedades que me permiten cambiar el aspecto de un elemento.

Vamos arrancar con la sintaxis básica:

```
selector {
  propiedad: valor;
}
```

- **Selector:** a que elemento HTML vamos a aplicar un estilo concreto.
- **Propiedades:** nos permiten modificar una característica de un elemento HTML, se divide en propiedad y valor que queremos aplicar, por ejemplo, queremos modificar el margen derecho de un div y que sea de 8 pixeles:
    - Propiedad: *margin-right*.

    - Valor: *8px*

Por ejemplo, imaginemos que queremos poner todos los divs de nuestra aplicación con un color de fondo rojo, ¿Que podríamos hacer?

- Elegir el selector *div*
- Definir dentro la propiedad *background-color: red*

```
div {
  background-color: red;
}
```

Así todos los divs que estén en nuestro HTML, tienen un color de fondo rojo.

Si quieres ver todas las palabras reservadas que se usan en CSS los chicos del MDN tienen publicada una referencia: https://developer.mozilla.org/es/docs/Web/CSS/Reference

Veamos un ejemplo un poco más elaborado, para todos los elementos *h1* quiero:

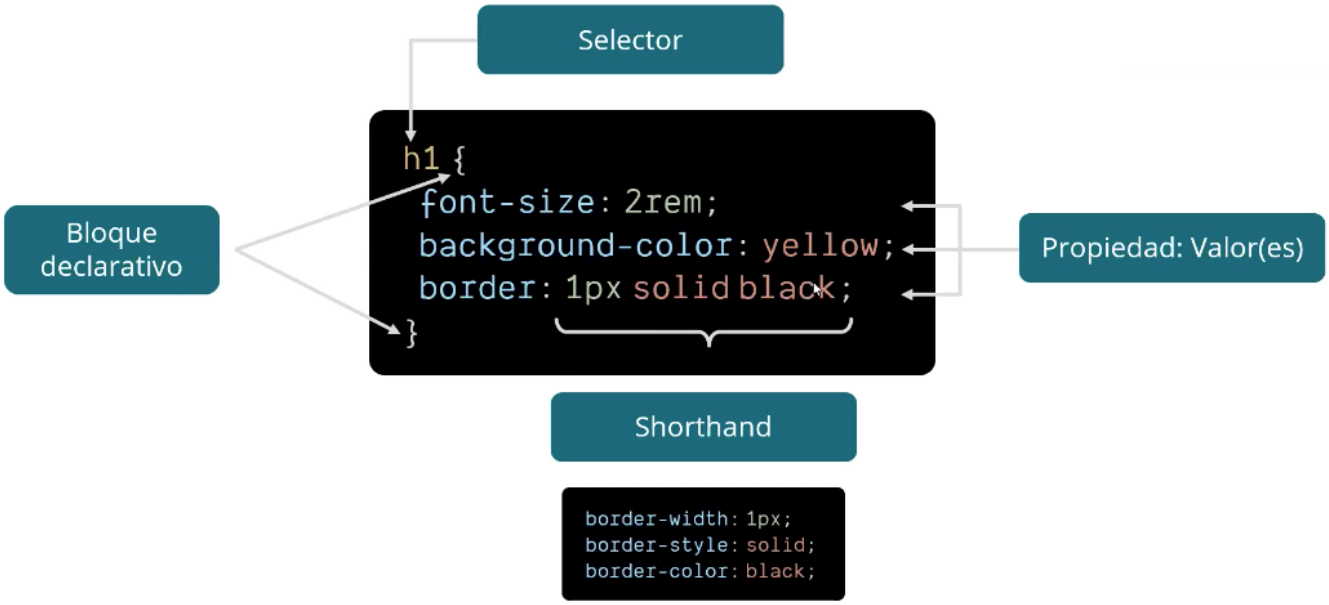
- Un tamaño de fuentes de *2rem*
- Un color de fondo amarillo.
- Un borde que rodee al H1 que tenga color negro y un pixel de ancho.

¿Cómo sería esto en CSS?

```
h1 {
  font-size: 2rem;
  background-color: yellow;
  border: 1px solid black;
}
```

Si miramos cada pieza podemos ver:

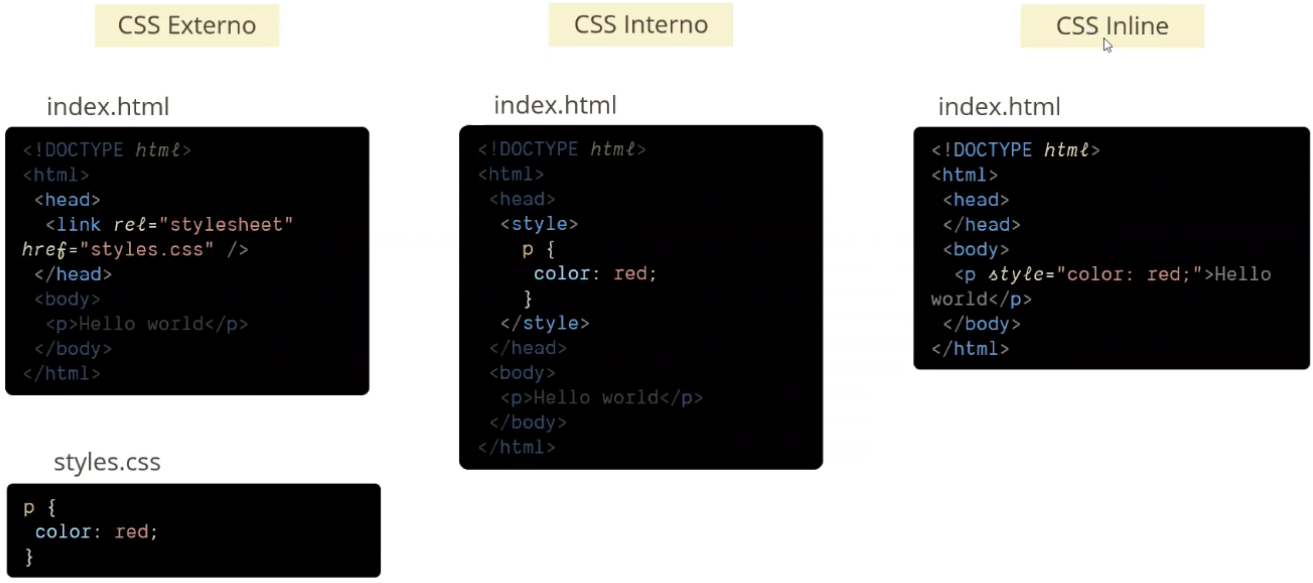
- El selector (en este caso el *h1*)
- El Bloque Declarativo (las llaves que siguen al h1).
- Las propiedades y sus valores (*font-size*, *background-color*, *border*)
- Y también podemos ver un short hand, resulta que para las propiedades de border tendríamos que definir *border-width*, *border-style* y *border-color*, esto lo puedo agrupar todo en un string



Donde escribir el CSS

El CSS podemos escribirlo en varios sitios:

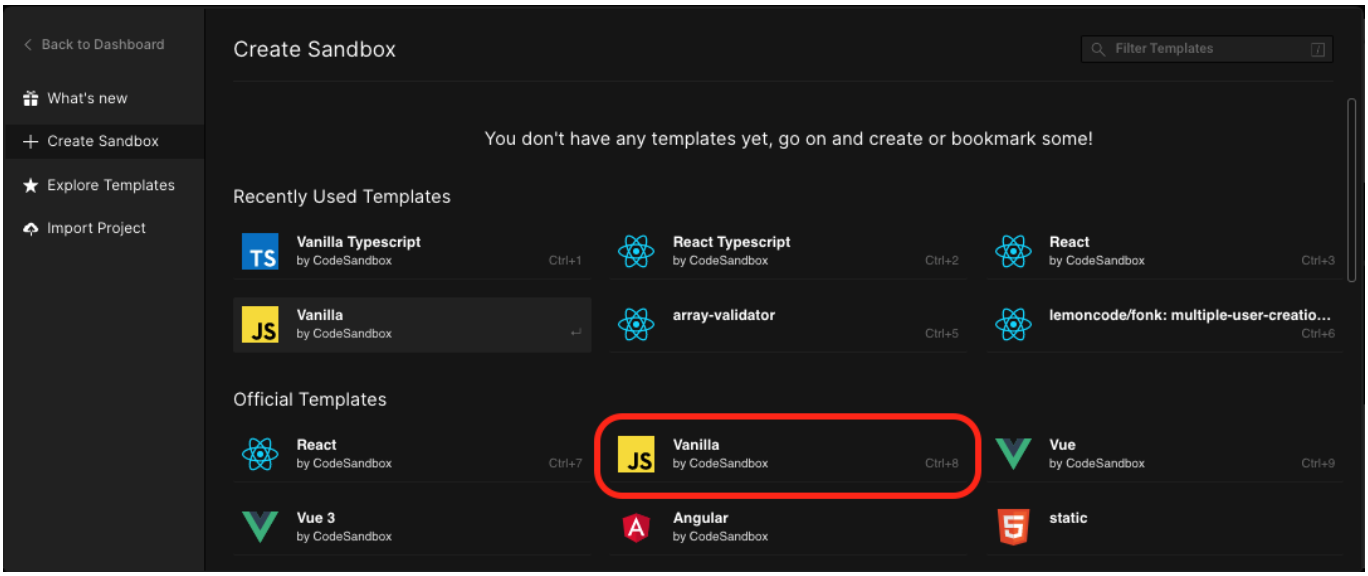
- Inline: directamente como estilo en un elemento de HTML (por ejemplo un párrafo). Esto lo debemos usar de forma limitada, aquí no necesitamos selectores ya que directamente lo aplicamos al elemento, el problema de esta aproximación es que no nos permite reusar estilos, no tenemos separación de responsabilidades, y además estamos metiendo más peso en la página.
- Interno: En un mismo fichero HTML puedo declarar una sección de estilos, en esta sección si que usamos selectores, esto mejora la opción anterior ya que lo tenemos todo encapsulado, pero no podemos compartir estos CSS entre distintos HTML.
- Externo: El CSS lo puedo extraer a un fichero con extensión CSS e importarlo usando el tag *link* en el propio HTML.



Vamos a abrir un codesandbox y jugar con una plantilla estándar html/css.

https://codesandbox.io/

Y elegimos *Vanilla* JS



codesandbox.png

Borramos el contenido de *index.js*

Y vamos a jugar con HTML y CSS

Abrimos el *index.html* y añadimos un *h1* y un *h2*

```
<body>
    <div id="app">
+   <h1>Bienvenido al módulo de layout</h1>
+   <h2>by Lemoncode</h2>
  </div>
    <script src="src/index.js">
    </script>
</body>
```

Podemos ver que sale por pantalla.

Empecemos a estilar, primero vamos a por los estilos inline, le decimos que el body de la página es de color marrón.

./index.html

```
-   <body>
+   <body style="background-color: brown">
    <div id="app">
            <h1>Bienvenido al módulo de layout</h1>
            <h2>by Lemoncode</h2>
    </div>
```

```
    <script src="src/index.js"></script>
  </body>
```

También podemos cambiar el color de los *h1* y *h2*

./index.html

```
    <div id="app">
-           <h1>Bienvenido al módulo de layout</h1>
+     <h1 style="color: white">Bienvenido al módulo de layout</h1>
-           <h2>by Lemoncode</h2>
+     <h2 style="color: white">by Lemoncode</h2>
    </div>
```

Podemos ver el resultado que obtenemos

Si analizamos el código que hemos generado podemos ver:

- Que es complicado de leer.
- Que tenemos que ir copiando y pegando para por ejemplo definir el color de fondo blanco en el h1 y h2.
- Que si creara otro h1 en la página, tendría que copiar el estilo del otro elemento h1.

Vamos a aislar estos estilos en una sección del documento y crear selectores para definirlos.

Para ello abrimos sección en el head de estilos

./index.html

```
  <head>
    <title>Parcel Sandbox</title>
```

```
    <meta charset="UTF-8" />
+   <style>
+     body {
+       background-color: brown
+     }
+     h1 {
+       color: white
+     }
+     h2 {
+       color: white
+     }
+   </style>
  </head>
```

Y podemos borrar los estilos en línea:

```
-  <body style="background-color: brown;">
+  <body>
    <div id="app">
-      <h1 style="color: white;">Bienvenido al módulo de layout</h1>
+      <h1>Bienvenido al módulo de layout</h1>
-      <h2 style="color: white;">by Lemoncode</h2>
+      <h2>by Lemoncode</h2>
    </div>
    <script src="src/index.js"></script>
  </body>
</html>
```

El HTML que obtenemos es más limpio

```
<body>
  <div id="app">
    <h1>Bienvenido al módulo de layout</h1>
    <h2>by Lemoncode</h2>
  </div>
  <script src="src/index.js"></script>
</body>
```

Pero podemos hacer mejora en el CSS, por ejemplo si en el *h1* y *h2* quiero aplicar el mismo color de fondo no tengo que duplicarlo, puedo especificar varios selectores y poner una sola entrada de propiedades:

```
-     h1 {
-       color: white
-     }
-     h2 {
-       color: white
```

```
-     }
+     h1, h2 {
+       color: white
+    }
```

Si quisiéramos aplicar esto a otra página HTML, me haría falta sacarlo a un fichero externo para poder aprovechar la misma hoja de estilo, para ello puedo crear un fichero externo:

./src/miestilo.css

```
body {
  background-color: brown;
}

h1,
h2 {
  color: white;
}
```

Y puedo referenciarlo en la página HTML (ya podemos eliminar la sección de CSS que habíamos creado en el mismo fichero HTML).

./index.html

```
  <head>
    <title>Parcel Sandbox</title>
    <meta charset="UTF-8" />
-    <style>
-      body {
-        background-color: brown;
-      }
-      h1,
-      h2 {
-        color: white;
-      }
-    </style>
+   <link rel="stylesheet" type="text/css" href="./src/miestilo.css"/>
  </head>
```

Con este tag le indicamos al navegador que vamos a cargar un fichero, con *rel* le indicamos que es una hoja de estilo, y con type que es de tipo *css*.

Selectores

Características de los selectores:

- Es un mecanismo para hacer target sobre elementos HTML.
- Determinan sobre que elemento se va a aplicar la regla CSS de la que forman parte.

- También nos permiten definir bajo que estado se va a aplicar dicha regla css (por ejemplo cuando el ratón este sobre un elemento aplicar un estilo)

Existe una gran variedad de selectores, hasta ahora hemos visto los de elemento, pero podemos aplicar más e incluso podemos combinarlos.

Selectores

En este caso tenemos el elemento y:

- Selectores simples:
  - Tag: por ejemplo un parrrafo (*p*) o un heading (*h1*).
  - Identificadores; un identificador de un elemento (el *id* de un elemento de *HTML*)
  - Clases: los elementos de *HTML* tiene un atributo clase que nos permite añadir una o más clases de CSS (en este caso en vez de usar un *id* para definir el selector de la clase* le definimos un nombre de _clase*)
- Los selectores complejos los iremos viendo paso a paso más adelante:
  - Pseudo clases.
  - Pseudo elementos.
  - Atributos.

¿Cómo se escribe esto en CSS?

Un selector de elemento (todos los h1)

```
h1 {
}
```

Un selector de clase, fíjate en el punto que ponemos como prefijo (afecta tags HTML que tengan definido en el atributo *class* ese selector)

```
.miSuperClase {
}
```

Un selected de identificador, fíjate en el carácter almohadilla que ponemos como prefijo (afecta a los tags HTML que tengan el *id* con el nombre indicado, en el caso del ejemplo *idElemento*):

```
#idElemento {
}
```

Y por último, ver que podemos aplicar de una sola tacada las mismas propiedades a varios selectores (en este caso le decimos que todos los *h1*, *h2* y *.title* van a aplicar las propiedades que se indiquen):

```
```

```
h1,
h2,
.title {
}
```

Selectores de clase y de identificador

Volvemos sobre el ejemplo anterior (Codesandbox), vamos a añadir dos *div*, una que tendrá la clase *title* y otro *div* con el *id marca*

```
    <div id="app">
      <h1>Bienvenido al módulo de layout</h1>
            <h2>by Lemoncode</h2>
+           <div class="title">Esto es el título</div>
+           <div id="marca">Hola desde el id marca</div>
    </div>
```

Sobre referenciar por *clase* versus por *id* hay que tener en cuenta que, un identificador sólo debería de aplicarse una sola vez en un HTML, no es buena idea que más de un elemento HTML tenga dicho *id* en la misma página, estilar usando *id* normalmente se usa para escenarios concretos de implementación.

Vamos al CSS y hagamos target sobre el *div* que tenga el *class* target, si ahora usáramos un selector del tipo

./src/styles.css

```
h1,
h2 {
  color: white;
}

+ div {
+  /* tu estilo aqui */
+ }
```

No nos valdría ya que afectaría a todos los div de la página, en este caso vamos a identificar a esos div por la classe *title*:

```
h1,
h2 {
  color: white;
}

+ .title {
+   color: blue;
```

```
+   font-size: 500%;
+ }
```

Y el que tiene *id* vamos a seleccionarlo por su identificar (*marca*), vamos a decirle, por ejemplo que tengo un margen de 25 pixeles.

```
h1,
h2 {
  color: white;
}

.title {
   color: blue;
   font-size: 500%;
}

+ #marca {
+  margin: 25px;
+ }
```

Ahora podemos ver que el contenido del *div* que tiene aplicada la clase *title* muestra el texto en color azul, la fuente cinco veces más grande, y el que tiene el identificador *marca* muestra el texto desplazado 25 pixeles.

Selector por clase título en blanco y fuente más grande, selector por Id texto desplazado 25 pixeles

¿Que pasaría si yo le aplico esa clase al *h1*?

./index.html

```
  <body>
    <div id="app">
-      <h1>Bienvenido al módulo de layout</h1>
+      <h1 class="title">Bienvenido al módulo de layout</h1>
           <h2>by Lemoncode</h2>
           <div class="title">Esto es el título</div>
```

Pues que el *h1* se muestra con el texto en azul y un tamaño de 500 pixeles, esto nos puede sonar raro, ya que por un lado tenemos un selector de css de elemento para el *h1* y por otro lado tengo otro que aplica a la clase *title* tenemos un conflicto aquí, como se resuelve en detalle lo veremos más adelante (cascading style sheet), como aperitivo:

- Se aplican los estilos del h1
- Después se aplican los de la clase (machacando los que define la clase y estén en el selector previo)

Una cosa que podemos indicarle es que una clase sólo aplique a los elementos que sean *div* ¿Cómo hacemos esto? Indicando que vamos a usar el selector de elemento *div* y a continuación concatenando el selector de clase *title*

```
- .title {
+ div.title {
    color: blue;
    font-size: 500%;
}
```

Con esto creamos un selector más complejo, le decimos que su target son todos los *div* en concreto los que tengan la clase *title* si ahora grabamos, podemos ver que el estilo de *title* solo se aplica al *div* y no al *h1* ya que el *h1* no es un elemento de tipo *div*.

Selectores de atributo

Los selectores de atributo me permiten seleccionar tags de HTML por algunas de sus propiedades (atributos), por ejemplo por el campo *title* de un tag html, por el campo *href*..., en el css lo marcamos entre corchetes.

Ejemplos de atributos:

Patrón	Descripción
[title]	Existe el atributo title
[title="EMail"]	Existe el atributo title y es igual a "EMail"
[title ~="logo"]	Existe el atributo title y posee una lista de valores que contiene "logo"
[lang|="en"]	Existe el atributo lang y contiene una lista de valores que alguno contiene "en" o comienza con "en-"
[href^="https"]	Existe el atributo href y comienza con "https"
[href$=".pdf"]	Existe el atributo href y termina por "pdf"
[href*="lemoncode"]	Existe el atributo href y contiene "lemoncode" en cualquier posición

Vamos a ver como aplicarlos a un elemento:

Volvemos al ejemplo anterior, vamos a añadir unos estilos basados en atributos:

Vamos a reemplazar el contenido del *body* por una lista de enlaces:

./index.html

```
<body>
  <body>
-    <div id="app">
-      <h1 class="title">Bienvenido al módulo de layout</h1>
-      <h2>by Lemoncode</h2>
-      <div class="title">Esto es el título</div>
-      <div id="marca">Hola desde #marca</div>
-    </div>
+    <a title="Website" href="http://www.lemoncode.net">lemoncode</a>
+    <a title="Email" href="mailto:info@lemoncode.net">EMail</a>
+    <a href="file1.pdf">File PDF</a>
+    <a href="file2.txt">File TXT</a>
     <script src="src/index.js"></script>
  </body>
</body>
```

Antes de seguir, si estás con el ejemplo del *codesandbox*, por salud visual vamos a quitar los estilos que teníamos (fuera fondo rojo 😃).

./src/miestilo.css

```
- body {
-  background-color: red;
- }
-
- h1,
- h2 {
```

```
-  color: white;
- }
-
- div.title {
-  color: blue;
-  font-size: 500%;
- }
-
- #marca {
-  margin: 25px;
- }
```

Vamos ahora a darle estilos a los enlaces.

En principio todos tienen un *href* podríamos decirles que tuvieran un tamaño de fuente y un padding, así ese selector me va a aplicar estos estilos a todo lo que tenga *href*

./src/miestilo.css

```
[href] {
  font-size: 125%;
  padding: 8px;
}
```

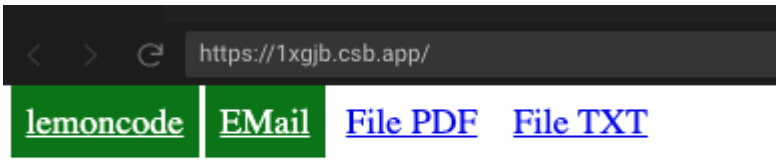
Si te fijas, todos los enlaces tienen ahora una fuente más grande y además hay un espacio entre los enlaces.

¿Y si quisiéramos hacer target sobre los href que contenga como substring en esta propiedad la palabra lemoncode para ponerles el color de fondo verde sobre texto blanco?

./src/miestilo.css

```
[href] {
  font-size: 125%;
  padding: 8px;
}

+ [href*="lemoncode"] {
+  background-color: green,
+  color: white,
+ }
```

Si te fijas, ahora todo lo que contiene el substring *lemoncode* en el *href* sale con color de fondo verde:



Además podríamos decir, oye y los *href* que terminan con *pdf* lo marcamos con otro color de fondo:

./src/miestilo.css

```
[href] {
  font-size: 125%;
  padding: 8px;
}

[href*="lemoncode"] {
  background-color: green,
  color: white,
}

+ [href$=".pdf"] {
+  background-color: indianred;
+  color: white;
+ }
```

Ahora podemos ver los ficheros con extensión PDF con color de fondo *rojo*

Y vamos a darle otro estilo a los *href* que terminen en *txt*

./src/miestilo.css

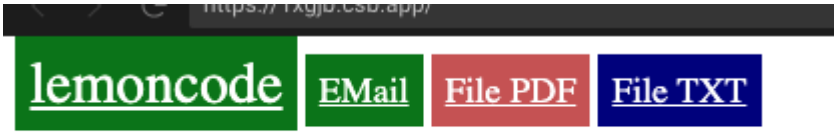
```
[href] {
  font-size: 125%;
  padding: 8px;
}

[href*="lemoncode"] {
  background-color: green,
  color: white,
}

 [href$=".pdf"] {
  background-color: indianred;
  color: white;
```

```
 }

+ [href$=".txt"] {
+  background-color: darkblue;
+  color: white;
+ }
```

Podemos también encadenar selectores, por ejemplo:

```
[title="Website"][href*="lemoncode"] {
  font-size: 200%;
}
```

El resultado de esto:



Selectores pseudo clases

Nos permiten seleccionar elementos según su comportamiento o su estado, por ejemplo si he puesto el ratón por encima del elemento (*hover*), o si está deshabilitado (disabled)...

```
a:visited { }

elemento:hover { }
elemento:active { }
elemento:focus { }
```

En esta imagen podemos ver:

- Un enlace visitado, en ese estado le puedo aplicar un estilo especial.
- Un elemento al que le indicas que cuando esté el ratón encima (hover), se le apliquen unos estilos (por ejemplo en un botón, que se resalte).

- Un elemento que tiene el foco en ese momento, lo mismo podemos aplicarle un estilado para que se vea que el control contra el que el usuario puede interactuar en ese momento.

En la documentación oficial de mozilla podéis encontrar un listado completo de pseudoclases.

Veamos un ejemplo:

Vamos a limpiar el *body* del *html* y nuestra hoja de estilos.

./index.html

```
  <body>
-    <a title="Website" href="http://www.lemoncode.net">lemoncode</a>
-    <a title="Email" href="mailto:info@lemoncode.net">EMail</a>
-    <a href="file1.pdf">File PDF</a>
-    <a href="file2.txt">File TXT</a>
-    <script src="src/index.js"></script>
  </body>
```

./src/miestilo.css

```
- [href] {
-  font-size: 125%;
-  padding: 8px;
- }
-
- [href*="lemoncode"] {
-  background-color: green;
-  color: white;
- }
-
- [href$=".pdf"] {
-  background-color: indianred;
-  color: white;
- }
-
- [href$=".txt"] {
-  background-color: darkblue;
-  color: white;
- }
-
- [title="Website"][href*="lemoncode"] {
-  font-size: 200%;
- }
```

Vamos a añadir un *h1*, un *input* y un *div* con la clase *container* que va a contener una serie de *divs* con un texto cada uno:

./index.html

```
<body>
+  <h1>Heading</h1>
+  <input type="text">
+  <div class="container">
+    <div>Primer elemento</div>
+    <div>Segundo elemento</div>
+    <div>Tercer elemento</div>
+    <div>Cuarto elemento</div>
+  </div>
</body>
```

Vamos a trabajar sobre ello:

Quiero seleccionar un input, podría hacer algo así

./src/miestilo.css

```
input {
  padding: 15px;
}
```

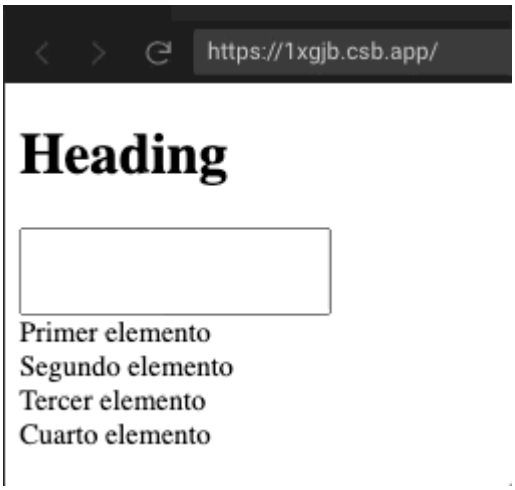
Ahora podemos ver la caja del *input* más grande aplicando un *padding*

¿Y si quisiéramos que cuando el input tuviera el foco cambiara el color de fondo a *verde*?

./src/miestilo.css

```
input {
  padding: 15px;
}

+ input:focus {
+   background-color: darkolivegreen;
+   color: white;
+ }
```

Si ahora pinchamos en la caja de input y empezamos a teclear podemos ver que cambia el color de fondo a *verde*



Vamos ahora a introducir un estilo para que cuando pase el ratón por encima se ponga el color de fondo a rojo, en este caso si no tengo el foco, y pongo el ratón por encima se pondría en rojo.

./src/miestilo.css

```
input {
  padding: 15px;
}

input:focus {
  background-color: darkolivegreen;
  color: white;
}

+ input:hover {
+  background-color: darkred;
+ }
```

Vamos a trabajar ahora con la pseudo clase nth-child.

En nuestro caso vamos a hacer target sobre el contenedor, y además todos los divs hijos (atento al espacio entre *container* y *div*, y vamos a indicarle que aplique el estilo a los elementos pares):

./src/miestilo.css

```
input:hover {
  background-color: darkred;
}

+ .container div:nth-child(2n) {
+   color: darkorange;
+ }
```

El resultado:

Si quisiéramos los impares:

./src/miestilo.css

```
- .container div:nth-child(2n) {
-   color: darkorange;
- }

+ .container div:nth-child(2n + 1) {
+  color: darkgreen;
+ }
```

Se le pueden aplicar muchas combinaciones a *nth-child*.

También podemos indicar que al último elemento de esos divs los ponga de color rojo:

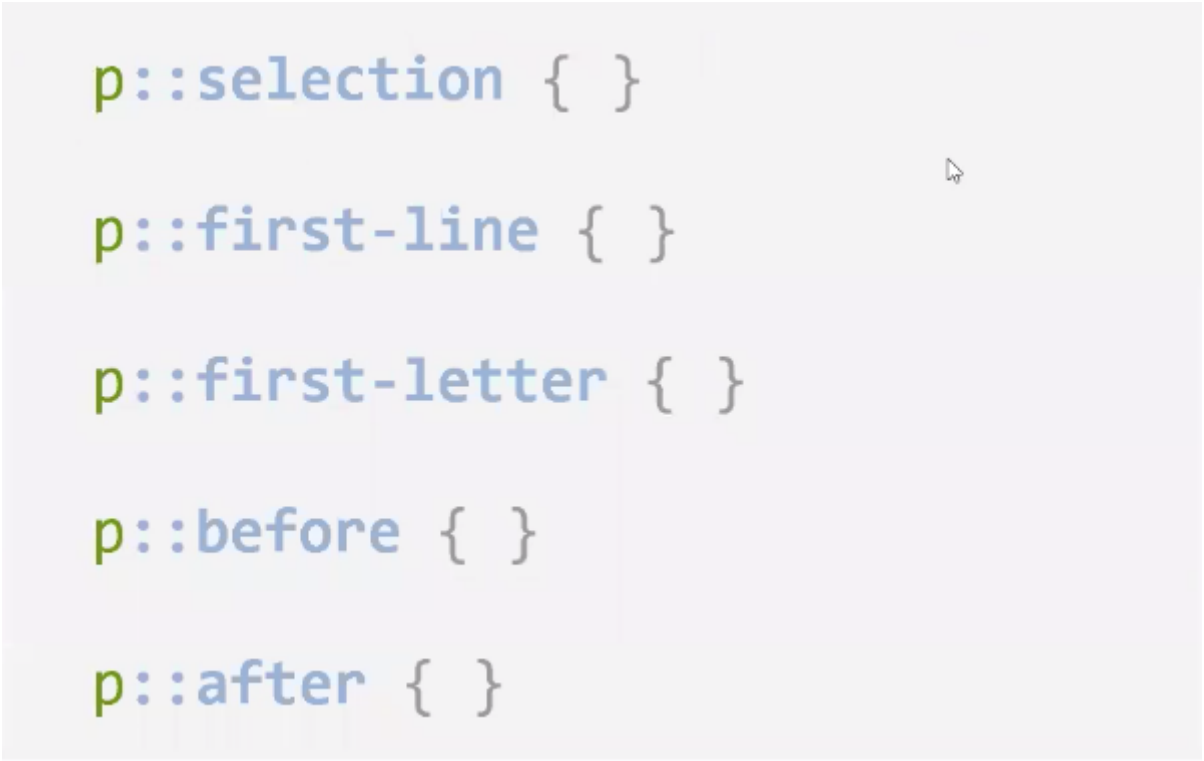
./src/miestilo.css

```
.container div:nth-child(2n + 1) {
  color: darkgreen;
}

+ .container div:last-child {
+   color: red;
+ }
```

Si quieres ver otras con las que poder jugar, puedes ver la lista completa en la web de developer mozilla (MDN) en la sección de referencia si buscáis por *pseudo-clases*

Selectores pseudo elementos

Nos permite seleccionar elementos según su situación en relación a otro elemento (primera palabra de un párrafo, contenido que sigue a un elemento, ...).



En estos ejemplos estamos viendo de un párrafo:

- Seleccionar la primera palabra o línea.
- El contenido que sigue a un elemento, o el que lo precede.
- Dentro de un párrafo el texto seleccionado.
- ...

Vamos a eliminar el contenido del body y añadir una cabecera (*h1*), un párrafo con un texto genérico (*lorem ipsum*) y añadir un *div* que ponga *lemoncode*.

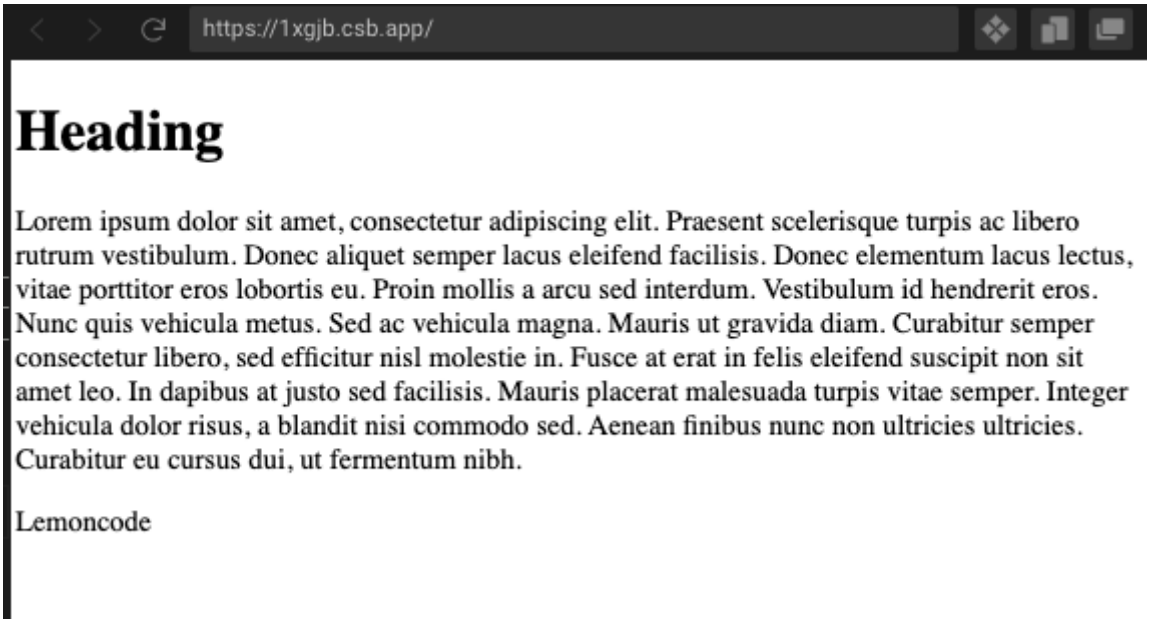
./index.html

```
  <body>
-    <h1>Heading</h1>
-    <input type="text" />
-    <div class="container">
-      <div>Primer elemento</div>
-      <div>Segundo elemento</div>
-      <div>Tercer elemento</div>
-      <div>Cuarto elemento</div>
-    </div>
+ <h1>Heading</h1>
+    <p>
+Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent
scelerisque turpis ac libero rutrum vestibulum. Donec aliquet semper lacus
eleifend facilisis. Donec elementum lacus lectus, vitae porttitor eros
lobortis eu. Proin mollis a arcu sed interdum. Vestibulum id hendrerit
eros. Nunc quis vehicula metus. Sed ac vehicula magna. Mauris ut gravida
diam. Curabitur semper consectetur libero, sed efficitur nisl molestie in.
Fusce at erat in felis eleifend suscipit non sit amet leo. In dapibus at
justo sed facilisis. Mauris placerat malesuada turpis vitae semper.
```

```
Integer vehicula dolor risus, a blandit nisi commodo sed. Aenean finibus
nunc non ultricies ultricies. Curabitur eu cursus dui, ut fermentum nibh.
+    </p>
+  <div class="marca">Lemoncode</div>
  </body>
```

Borramos el contenido del *css* que teníamos antes...

El aspecto que tenemos es:

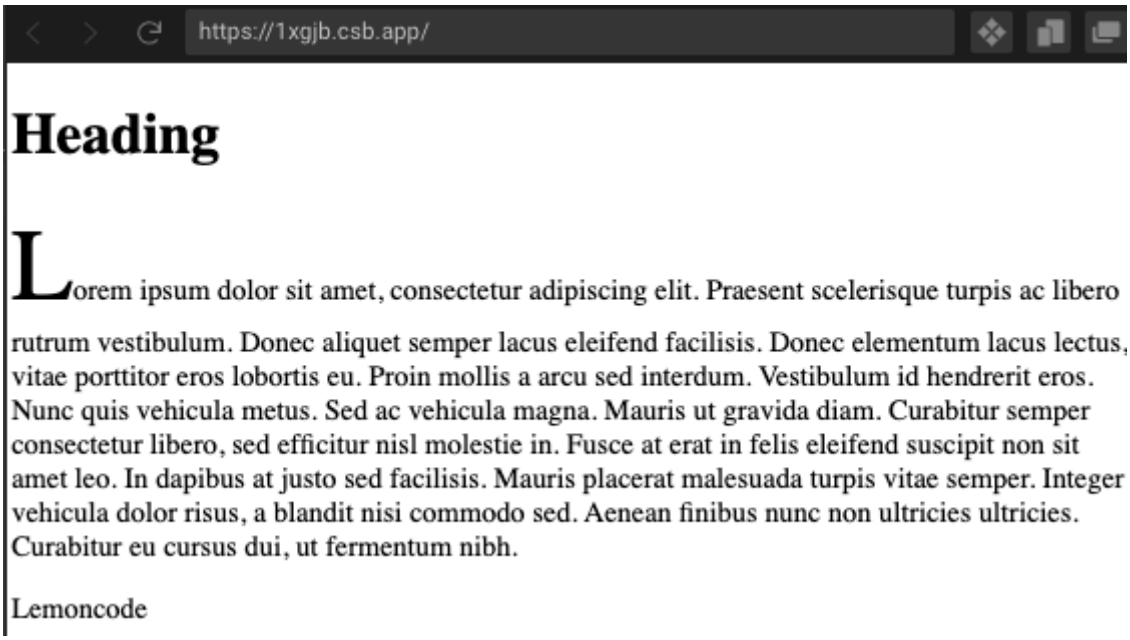


Vamos a crear un selector en el que vamos a seleccionar el parrafo *p* y de ahí el pseudo elemento del *first-letter* y ahí para la primera letra vamos a darle un font-size del 350%

./src/miestilo.css

```
p::first-letter {
  font-size: 350%;
}
```

Fíjate ahora como la primera letra del párrafo de se ve mucho más grande:



Fíjate que esa *L* no es un elemento, por eso se le llama pseudo elemento.

Si quisiéramos podríamos cambiar el aspecto te un texto que seleccionemos con el ratón, puedo hacer algo así como:

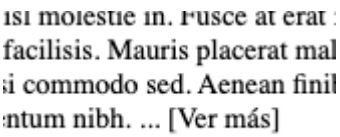
./src/miestilo.css

```
p::first-letter {
  font-size: 350%;
}

+ p::selection {
+  background-color: darkorange;
+  color: white;
+ }
```

También podemos decirle que después del párrafo nos añada un contenido del tipo "...[Ver más]"

```
- p::selection {
-  background-color: darkorange;
-  color: white;
- }

+ p:after {
+  content: "... [Ver más]"
+  cursor: pointer;
+ }
```

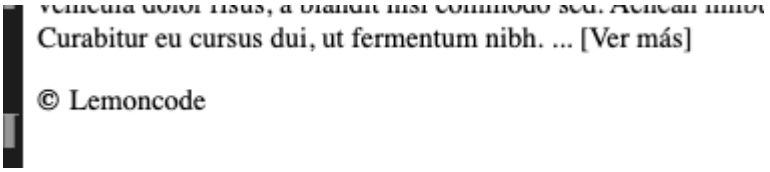
De esta manera nos aparece al final del párrafo este enlace y además se nos cambia el cursor indicando que podemos tener una acción adicional si pincháramos con el ratón.



Mucho cuidado cuando usemos esto, a nivel de accesibilidad, un lector no iba a tenerlo en cuenta, y personas invidentes podrían tener problemas para manejar la aplicación.

Vamos a ver ahora un ejemplo con el *div* que hemos definido, al final al que lo hemos identificado por la clase *marca*, en esta caso antes de añadir el texto *lemoncode* vamos a añadir un símbolo de *copyright*

```
p:after {
  content: "... [Ver más]"
  cursor: pointer;
}

+ div.marca::before {
+   content: "©";
+   padding-right: 6px;
+   font-weight: bold;
+ }
```



Otros selectores

Tenemos los selectores múltiples, es una agrupación de múltiples selectores separados por comas (esto ya lo hemos cubierto, aplicas el estilo a la lista de selectores que hemos indicado).

Combinaciones: combinaciones de dos o más selectores (por ejemplo: párrafos situados después de títulos).

Ejemplos de combinaciones:

Selector	Ejemplo	Descripción
Selectores descendientes (descendant selector)	div p	Selecciona todos los p dentro de un div
Selectores de hijos (child selector)	h1 > p	Selecciona todos los p descendiente directos de un h1
Hermano adyacente (adjacent sibling)	p + img	selecciona los img que sean hermanos inmediatos de un p
Hermano general (general sibling)	h1 ~ p	Selecciona los p que son hermano de un h1

Vamos a ver esto con varios ejemplos.

Borramos el contenido del body de nuestro *index.html*, y el css de *miestilo.css*

Vamos añadir el siguiente contenido a nuestro *body*:

./index.html

```
<body>
+  <h1>Heading 1</h1>
+  <input type="text"/>
+  <div class="container">
+    <p>Primer elemento</p>
+    <p>Segundo elemento</p>
+  </div>
+  <div class="container2">
+    <p>Tercer elemento</p>
+    <p>Cuarto elemento</p>
+  </div>
</body>
```

Y arrancamos jugando con CSS:

Primer caso, vamos a seleccionar todos los parrafos

./src/miestilo.css

```
p {
  color: red;
}
```

Con esto nos salen todos los párrafos con el texto en rojo:

(todos los párrafos rojos)[./content/todosprojos.png]

Si hacemos lo siguiente:

./src/miestilo.css

```
- p {
+ div p {
  color: red;
}
```

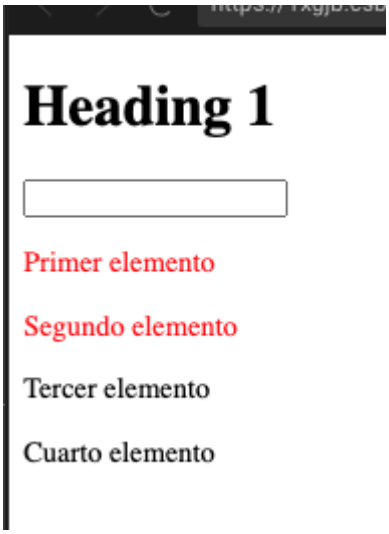
Nos sigue poniendo todos los párrafos en rojo, porque están dentro de *divs*.

Pero si por ejemplo le añadimos al *div* el selector de clase *.container* ya solo nos muestra los *p* que están debajo del *div* con la clase *container*

./src/miestilo.css

```
- div p {
+ div.container p {
  color: red;
}
```

El resultado:



Vamos a introducir un pequeño cambio en el HTML:

./index.html

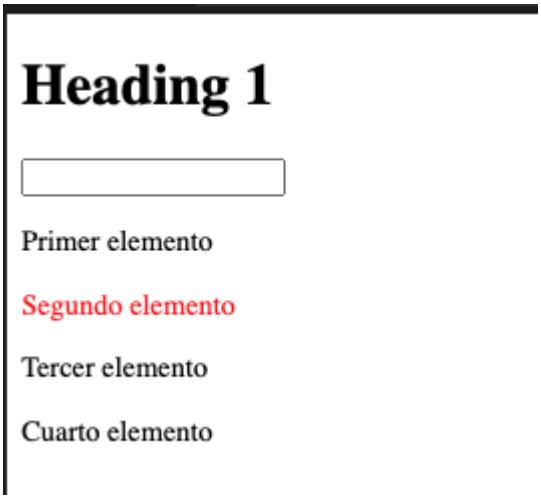
```
  <body>
    <h1>Heading 1</h1>
    <input type="text" />
    <div class="container">
+     <section>
        <p>Primer elemento</p>
+     </section>
      <p>Segundo elemento</p>
    </div>
-    <div class="container2">
+     <section class="container">
        <p>Tercer elemento</p>
        <p>Cuarto elemento</p>
+     </section>
-    </div>
  </body>
```

Si os fijáis seguimos teniendo los párrafos dentro del primer *div* con la clase *container* marcados en rojo esto es porque busca descendiente, pero lo puedo acotar usando el símbolo > para que sólo seleccione a los hijos directos:

./src/miestilo.css

```
- div.container p {
+ div.container > p {
  color: red;
}
```

Resultado:



Vamos a modificar el HTML:

index.html

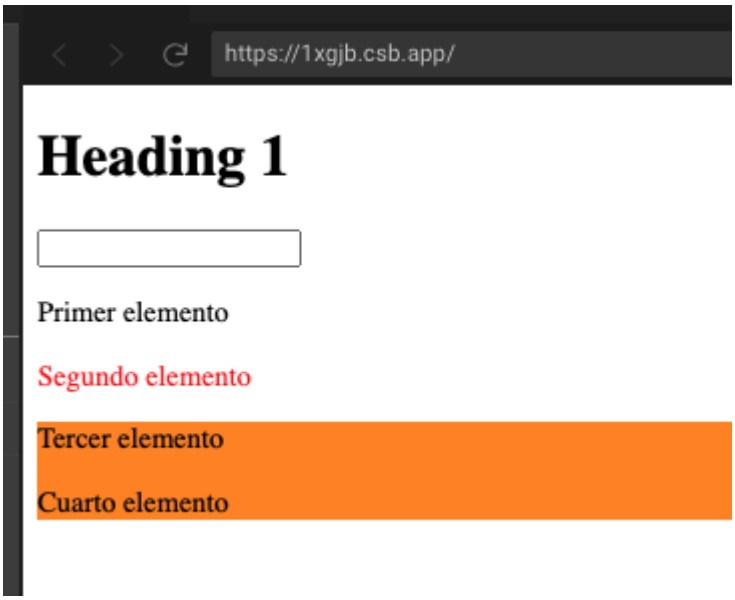
```
  <body>
    <h1>Heading 1</h1>
    <input type="text" />
    <div class="container">
      <section>
        <p>Primer elemento</p>
      </section>
      <p>Segundo elemento</p>
    </div>
    <section class="container2">
      <p>Tercer elemento</p>
      <p>Cuarto elemento</p>
    </section>
+   <section class="container">
+     <p>Quinto elemento</p>
+     <p>Sexto elemento</p>
+   </section>
    </div>
  </body>
```

Vamos ahora a jugar con los hermanos inmediatos (adyacentes), en este caso vamos a decirle que nos seleccione de los *div* con la clase *container* los hermanos inmediatos que sean *section* y le vamos a poner un color de fondo

./src/miestilo.css

```
div.container > p {
  color: red;
}

+ div.container + section {
+  background-color: darkorange;
+ }
```

Esto me selecciona el hermano contiguo de tipo section que tengo pegado a un *div* con clase *container* es decir el *section* con class *container2* (mira el bloque marcado en naranja).



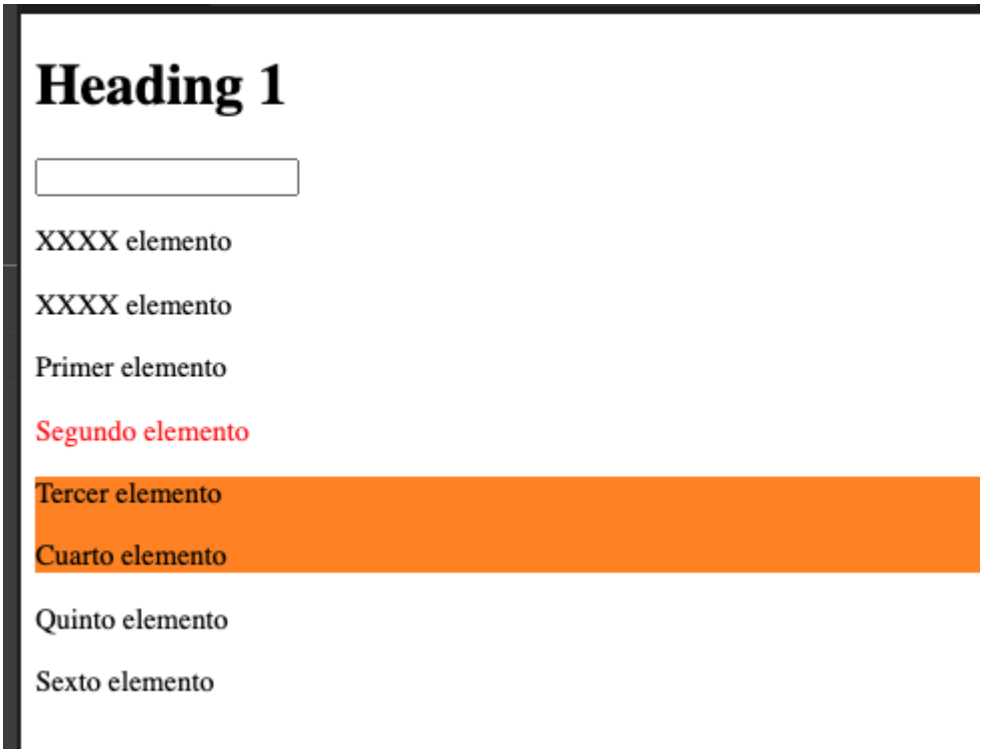
Oye y si tengo un hermano justo encima del div ¿Es también adyacente? modifiquemos el HTML:

./src/index.html

```
  <body>
    <h1>Heading 1</h1>
    <input type="text" />
+   <section class="container">
+     <p>XXXX elemento</p>
+     <p>XXXX elemento</p>
+   </section>
    <div class="container">
      <section>
        <p>Primer elemento</p>
      </section>
      <p>Segundo elemento</p>
    </div>
    <section class="container2">
      <p>Tercer elemento</p>
      <p>Cuarto elemento</p>
    </section>
    <section class="container">
     <p>Quinto elemento</p>
```

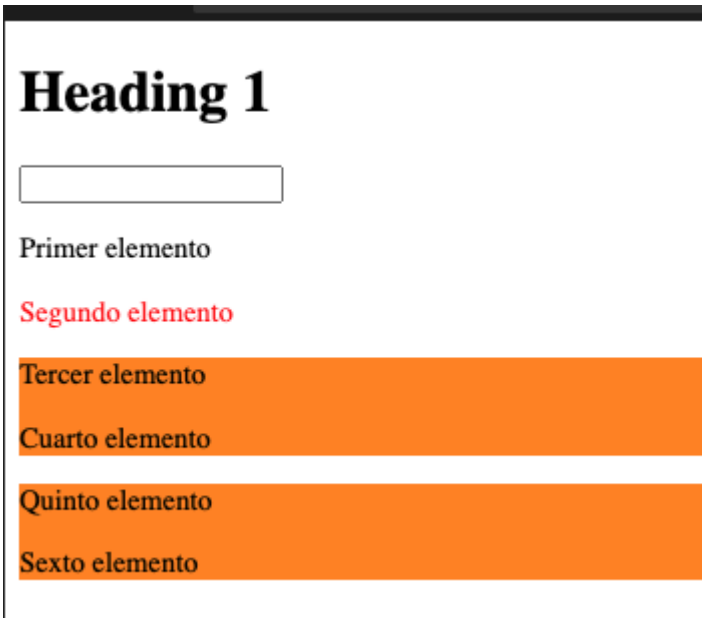
```
      <p>Sexto elemento</p>
    </section>
    </div>
  </body>
```

En este caso no se selecciona la nueva sección *¿Por qué? ¡si es adyacente!* Por que a la hora de aplicar esos estilos, mira desde donde lleva montado el HTML hacía abajo, algo que ya haya renderizado no se para a volver reevaluarlo.



Si quisiéramos que fueran todos los hermanos, pondríamos:

```
- div.container + section {
+ div.container ~ section {
  background-color: darkorange;
 }
```

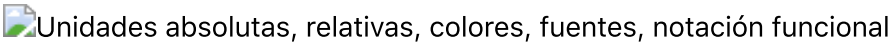


Especificando valores de CSS

Hasta ahora hemos estado jugando a meter valores en CSS sin tener en cuenta con que tipo estamos trabajando, que si números, que si pixeles, que si valores en hexadecimal, que si constantes... vamos a ver tipos de datos básicos que acepta CSS.

Tipo de datos	Descripción
	Es un número entero como 1024 0 -55
	Representa un número decimal; puede tener o no un punto de separación decimal
	Es un con una unidad asociada. es una categoría general que incluye los tipos , y
	Representa una fracción de otro valor. Son siempre relativos a otra cantidad

Especificando valores CSS:



En unidades absolutas podemos indicar por ejemplo el width en pixeles, centímetros picas..., cuando hacemos diseño responsivo no se suele trabajar con unidades absolutas.

Las unidades relativas, nos permite dar una visualización similar sin depender del tamaño del viewport, por ejemplo si ponemos tres cajas una al lado de otra y decirle que cada una ocupe el 33% del espacio disponible del viewport (el espacio que tenemos en el navegador), también podemos hacerlo al espacio que da el padre.

El *em* es el tamaño de fuente relativo al navegador, y el *rem* el tamañano relativo de fuente del *elemento root*, si al elemento root de nuestro documento le damos un tamaño de fuente todas las demás fuentes deberíamos de indicarle que fuera *rem*.

Para los colores podemos usar contantes (black, green), o en formato hexadecimal, rgb, rgba (con opacidad).

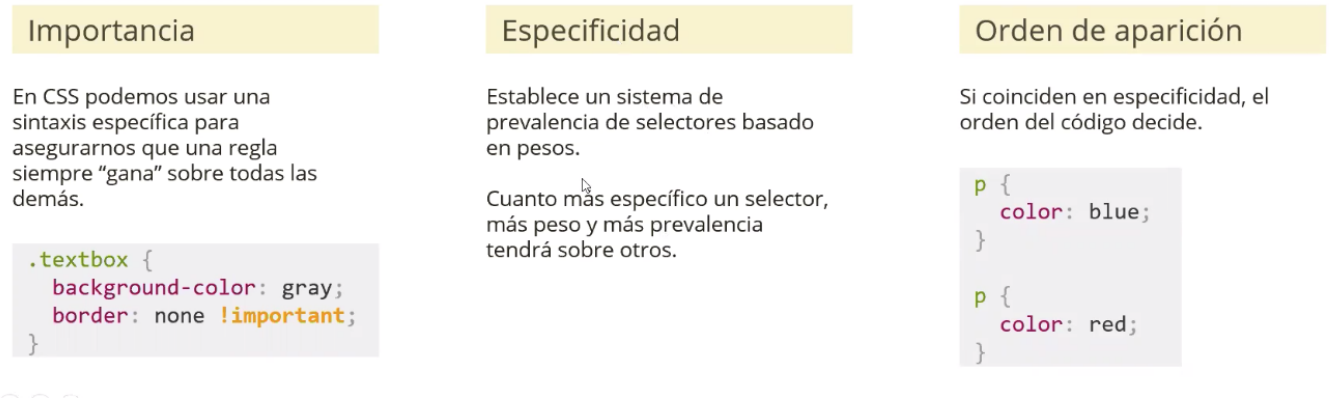
Las fuentes las podemos tener tal cual, si tienen espacios las ponemos entre comillas dobles.

También tenemos funciones, por ejemplo podemos decirle con *calc* que el tamaño de un elemento es el *calc* del 100% menos 50 pixeles, es decir todo lo que puede ocupar menos cincuenta pixeles, *translate* permite rotarlo, *url* podemos aplicar una imagen de fondo a un div.

Concepto de cascada

En las secciones previas, hemos visto que podemos aplicarle más de un selector a un elemento HTML, ¿Cómo resolvemos en caso de conflicto? ¿Se borran las propiedades del selector anterior y gana el que queda? ¿Sólo sobrescribe las que cambia? Veamos:

- Un elemento se puede ver afectado por más de una regla CSS.
- Si esa regla tiene propiedades diferentes, ninguna choca y se aplican las dos.
- Si hay reglas que entren en conflicto, el navegador tiene que elegir cual poner (si varias reglas colisionan).
- Cascada = el orden de las reglas CSS importa, así como lo específico que seamos a la hora de generar esos selectores, esto atiende a tres factores: importancia, especificad, y orden de aparición.



> A tener en cuenta hay que evitar usar *!important* en la medida de lo posible, ya que ¿Qué pasa si pongo un !important y después quiero reescribir eso... pongo otro *!important*? De hecho si nos encontramos esto en una librería de terceros estamos fastidiados.

Importancia

El concepto de importancia, tenemos:

- El CSS de autor que es el CSS que escribimos nosotros.

- El CSS de usuario (esto no lo tienen ya algunos navegadores), por ejemplo un usuario de un navegador podría tener su hoja de estilo global para todas las páginas, imagínate un usuario que ve que la fuente de los párrafos suele ser muy pequeña, y quiere que todas las webs cuando carguen tengan la fuente más grande por defecto.
- El CSS de agente de usuario: el navegador tiene unos estilos por defecto (cada navegador tiene el suyo), si echamos un ojo a una página por defecto, podemos ver ciertas fuentes, espaciado, tamaños de headings...

En cuanto a orden, el CSS de autor tiene preferencia sobre el de usuario, y el de usuario sobre el agente de usuario.

Especificidad

Establece un sistema de prevalencia de selectores basado en pesos.

Cuanto más específico es un selector más peso y prevalencia tendrá sobre otros.

Niveles de peso:

- A: Número de estilos aplicados mediante un atributo *style* (si existe, simpre valdrá 1).
- B: Número de apariciones de un ID en un selector.
- C: Número de apariciones de una clase, pseudoclase o atributo en el selector.
- D: Número de apariciones de un elemento o pseudoelemento en el selector.

Ejemplos:

Selector	A	B	C	D	Total (peso)
h1	0	0	0	1	0001
h1 + p::first-line	0	0	0	3	0003
li > a[title*="lemoncode"]| > .selected	0	0	2	2	0022
#miID	0	1	0	0	0100
<a style="color:red;">	1	0	0	0	1000

Para ayudarnos con estos calculos podemos usar la Calculadora de especificidad Keegan

Vamos a ver un ejemplo de como funciona esto:

En el HTML vamos a meter el siguiente:

./content/index.html

```
<body>
  <h1>Heading 1</h1>
  <h2>Heading 2</h2>
  <div>A div</div>
  <div>
    <p>A paragraph inside a div</p>
```

```
  </div>

  <p>Otro párrafo</p>
</body>
```

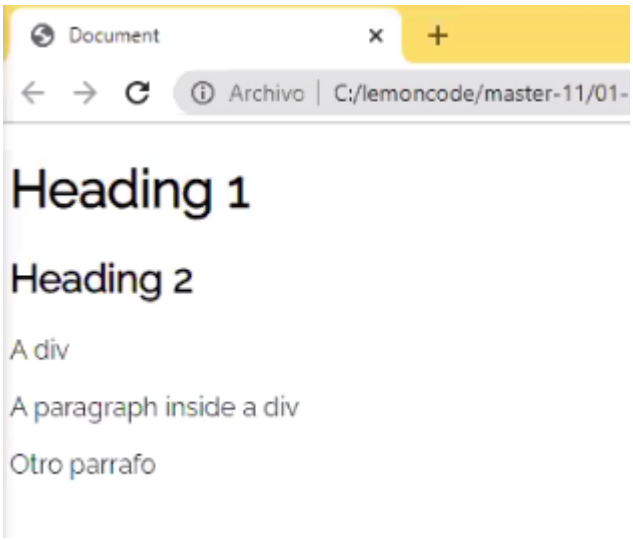
Vamos a jugar con el css y ver como se resuelven conflictos:

Borramos el contenido del fichero *.css*

Primero vamos a importarnos una fuente de google, y vamos a ponerla como fuente por defecto en el body.

```
@import "https://fonts.googleapis.com/css2?family=Raleway:wght@400&disp";

body {
  font-family: "Raleway", sans-serif;
}
```

Ahora podemos ver como ha aplicado esta fuente a todos los elementos que hay debajo del body.



Vamos a darle estilo en los *p*

```
p {
  background-color: green;
}
```

Esto va a hacer que todos los párrafos aparezcan con el color verde, vamos a añadir otra regla de estilo en el que el párrafo va a tener otro color de fondo, en este caso gris (*grey*), así pues esta segunda regla al tener el mismo peso, aplica la regla de, el último que llega gana.

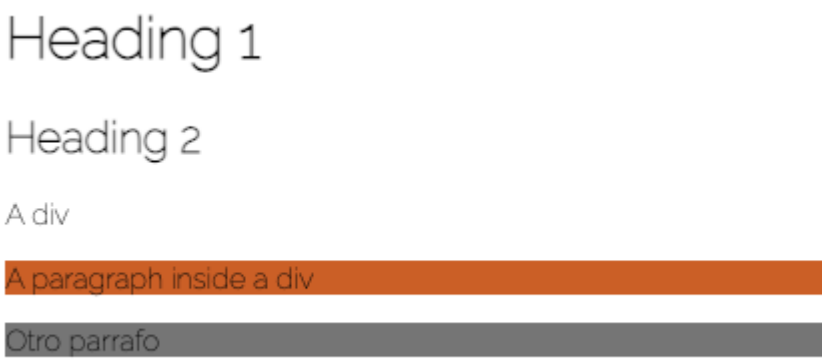
```
p {
  background-color: green;
```

```
}

+ p {
+  background-color: grey;
+}
```

¿Que ocurre si ponemos esta regla?

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

p {
  background-color: green;
}

 p {
  background-color: grey;
}
```

Esta regla ya es más específica (tiene dos selectores de elemento), afecta sólo a los párrafos dentro de un div y en ese caso prevalecería sobre las otras dos que sólo tienen un *p*.




```
<body>
  <section class="container">
    <p>I am a &ltp&gt</p>
    <div>
      <p class="red">I am red?</p>
      <p class="red" id="lemoncode">
        I amp a &ltp&gt inside a div, but I have an id!
      </p>
    </div>
  </section>
</body>
```

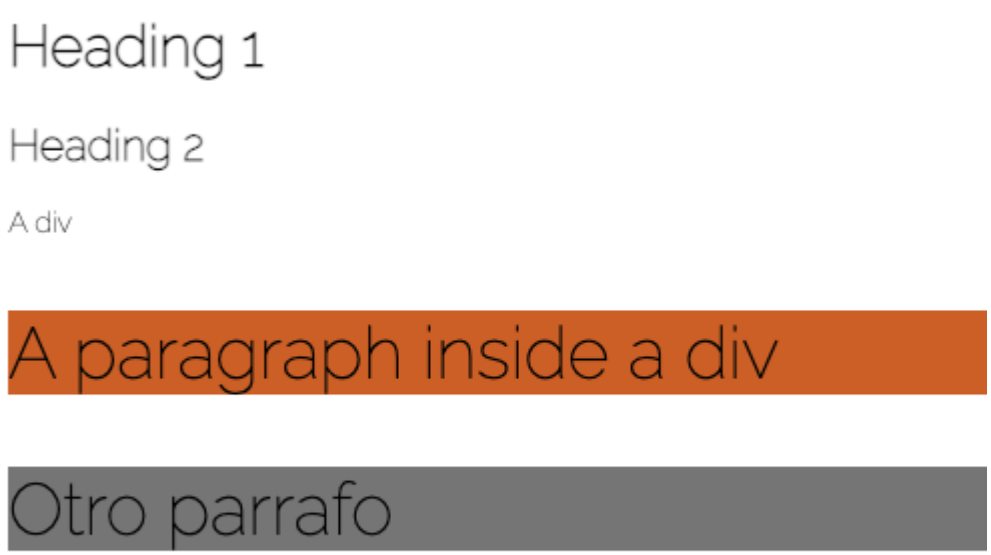
Si pinchamos con el botón derecho sobre el párrafo naranja y elegimos del menú "inspect", en la parte derecha podemos ver el área de *styles* y allí vemos la "lucha" de estilos para aplicar al párrafo:

Lucha de estilos, dev tools

¿Qué pasa si en el estilo del *p* que pinta de verde (el que se ignora), ponemos una fuente de por ejemplo 40 pixeles?

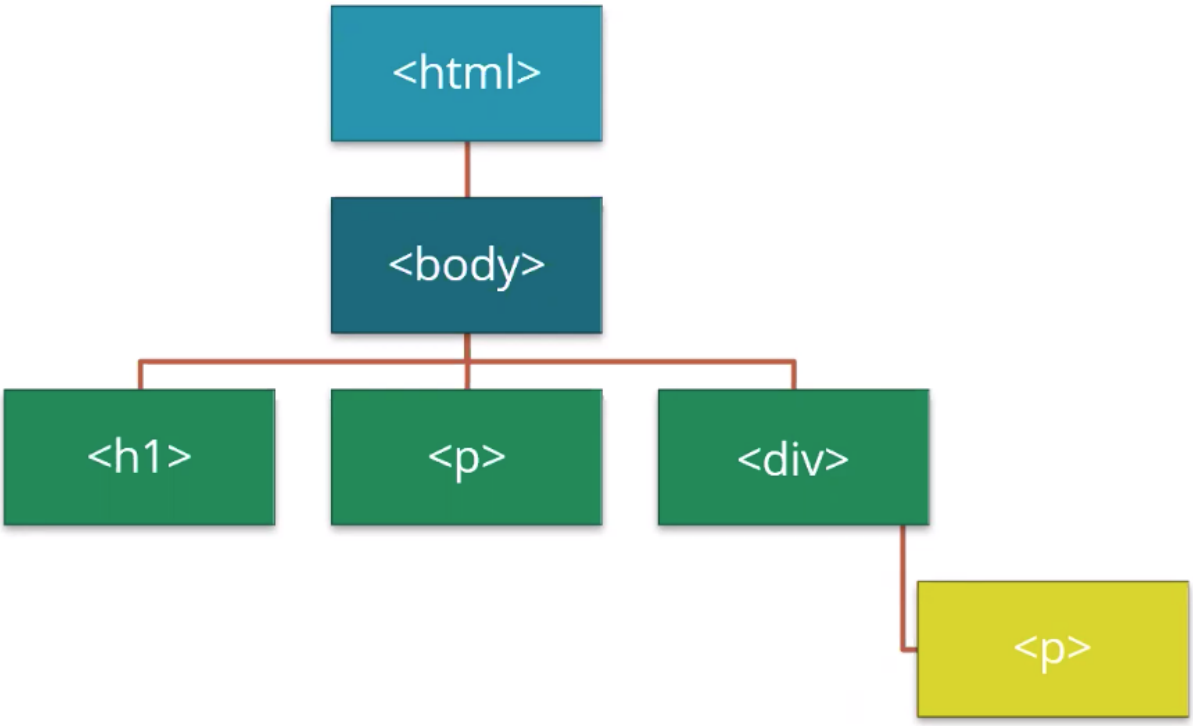
```
div p {
  background-color: chocolate;
}

p {
  background-color: green;
+ font-size: 40px;
}

p {
  background-color: grey;
}
```

Pues que el font size se aplica a todos los parrafos ¿Y eso? Si decíamos que esa clase se sobrescribía... resulta que la propiedad *font-size* no la sobrescribe ninguna de las que va delante, por lo que prevalece.



Concepto de herencia

Cuando hemos creado los estilos, hemos visto como la fuente definida en el *body* aplicaba a todos los elementos que teníamos dentro (*div* y *p*), sin embargo hay otras propiedades que no se heredan, por ejemplo a un *div* le añadimos un *border* (un recuadro), esa propiedad no la hereda el *p* hijo que tenga, esto tiene su lógica, porque si se heredara tendríamos que ir quitando el *border* a cada elemento hijo.



La herencia fluye de padre a hijos, tenemos tres formas definir la herencia:

- *inherit*: hereda el valor de la propiedad del elemento padre.
- *initial*: establece el valor que tenía la propiedad inicialmente.
- *unset*: Hereda el valor de la propiedad del elemento padre y en caso de no existir, su valor inicial.

inherit	Hereda el valor de la propiedad del elemento padre
initial	Establece el valor que tenía la propiedad inicialmente
unset	Hereda el valor de la propiedad del elemento padre, y en caso de no existir, su valor inicial

Veámoslo con un ejemplo, borramos el *html* añadimos el siguiente:

./index.html

```
<body>
  <div>
```

```
    Soy el div
    <p>
      A paragraph inside a div
      <em>emphasis</em>
      <strong>strong</strong>
    </p>
  </div>
</body>
```

Del *css* vamos a dejar sólo la selección de fuente:

./src/miestilo.css

```
@import "https://fonts.googleapis.com/css2?family=Raleway:wght@400&disp";

body {
  font-family: "Raleway", sans-serif;
}

- div p {
-  background-color: chocolate;
- }
-
- p {
-  background-color: green;
-  font-size: 40px;
- }
-
- p {
-  background-color: grey;
- }
```

Tenemos el siguiente resultado:

Soy el div

A paragraph inside a div *emphasis* **strong**

Ahora lo que tenemos son las propiedades del user agent y las heredadas por el *body* que hemos definido: el *font-family*, es decir esta propiedad hereda de padre a hijo.

Si metemos un borde al body, no se heredaría de padre a hijo, sólo aplica al *body*.

./src/miestilo.css

```
body {
+ border: solid red;
```

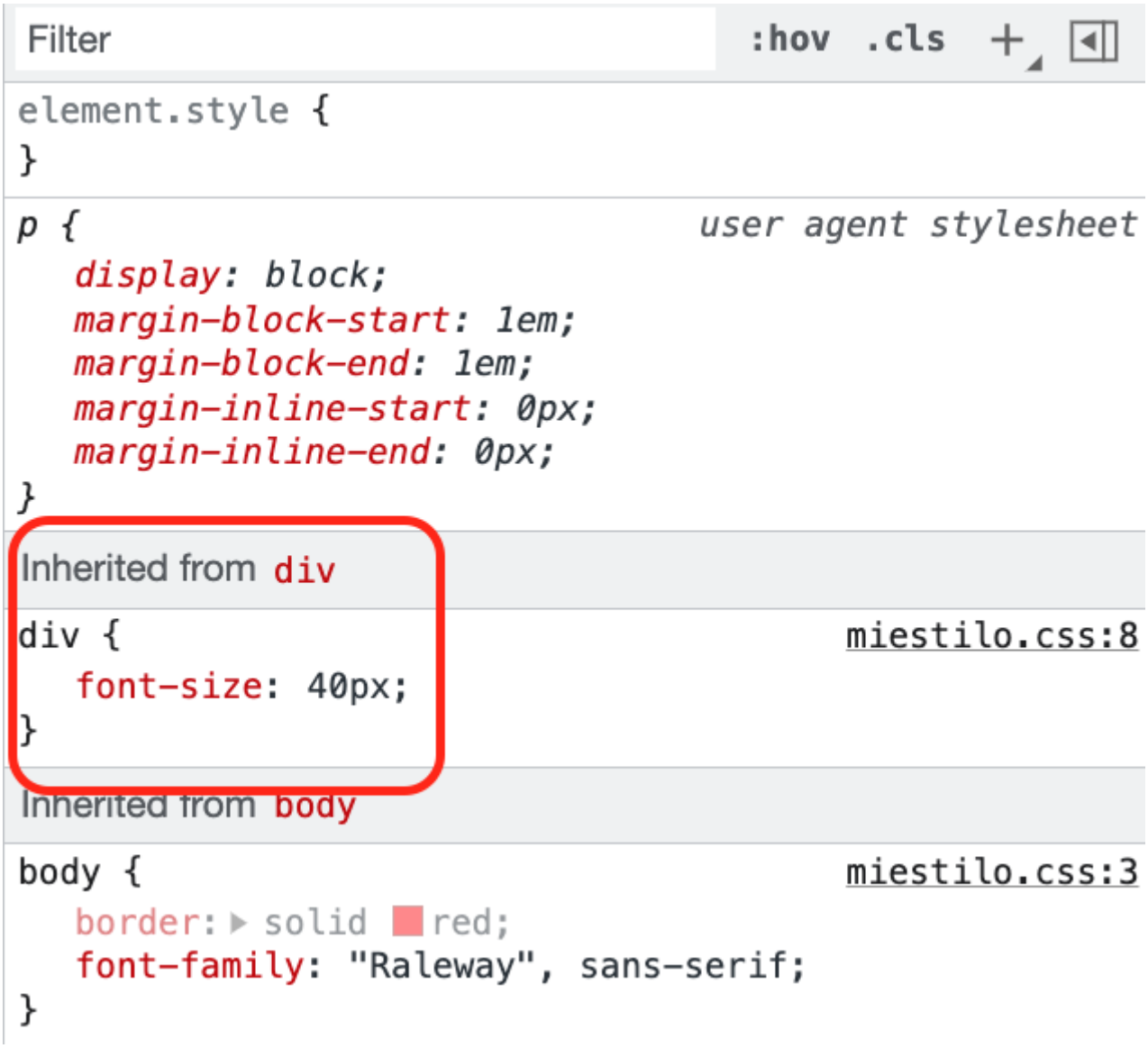
```
    font-family: "Raleway", sans-serif;
  }
```

Soy el div

A paragraph inside a div *emphasis* **strong**

Vamos a seguir jugando, por ejemplo vamos a cambiar para todos los *divs* el *font-size* y vamos a decirle que sea de 40 pixeles. En los *p* que hay dentro de los *divs* se aplica el tamaño de la fuente (lo hereda del div padre)

Si inspeccionamos el elemento con las *devtools* del navegador podemos ver como nos indica que hereda el tamaño de fuente del estilo aplicado al *div* padre.



Vamos ahora a indicarle que no coja la del padre, si no la inicial:

./content/miestilo.css

```
```

```
body {
  border: solid red;
  font-family: "Raleway", sans-serif;
}

div {
  font-size: 40px;
}

+ div p {
+  font-size: initial;
+ }
```

De esta manera le indicamos que ignore la fuente que hereda del padre y aplique la que tenía previo a la herencia:

Un tema curioso es que puedo heredar una propiedad que por defecto no se hereda, por ejemplo teníamos el *body* con un border, y ¿si le decimos al *div* que hay justo debajo que herede esa propiedad?

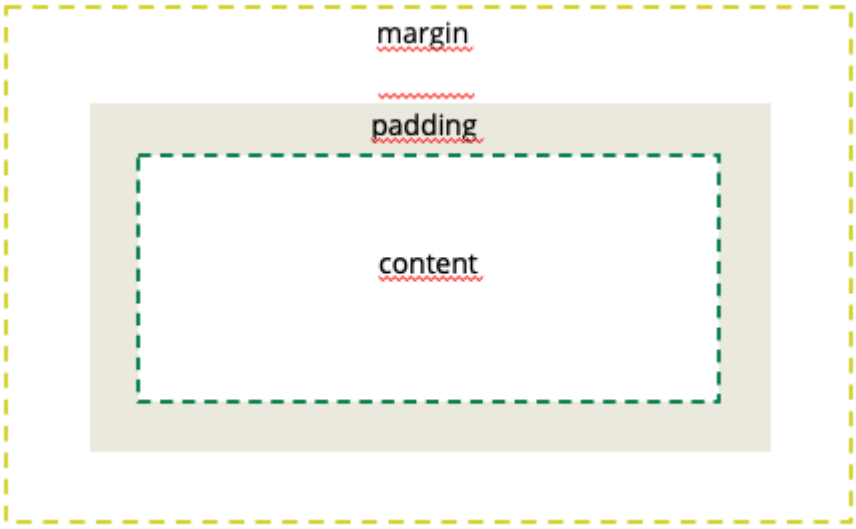
```
body {
  border: solid red;
  font-family: "Raleway", sans-serif;
}

div {
+  margin:40px;
+ border: inherit;
  font-size: 40px;
}
```

Fíjate que el div hereda la propiedad *border* del *body* y lo muestra.

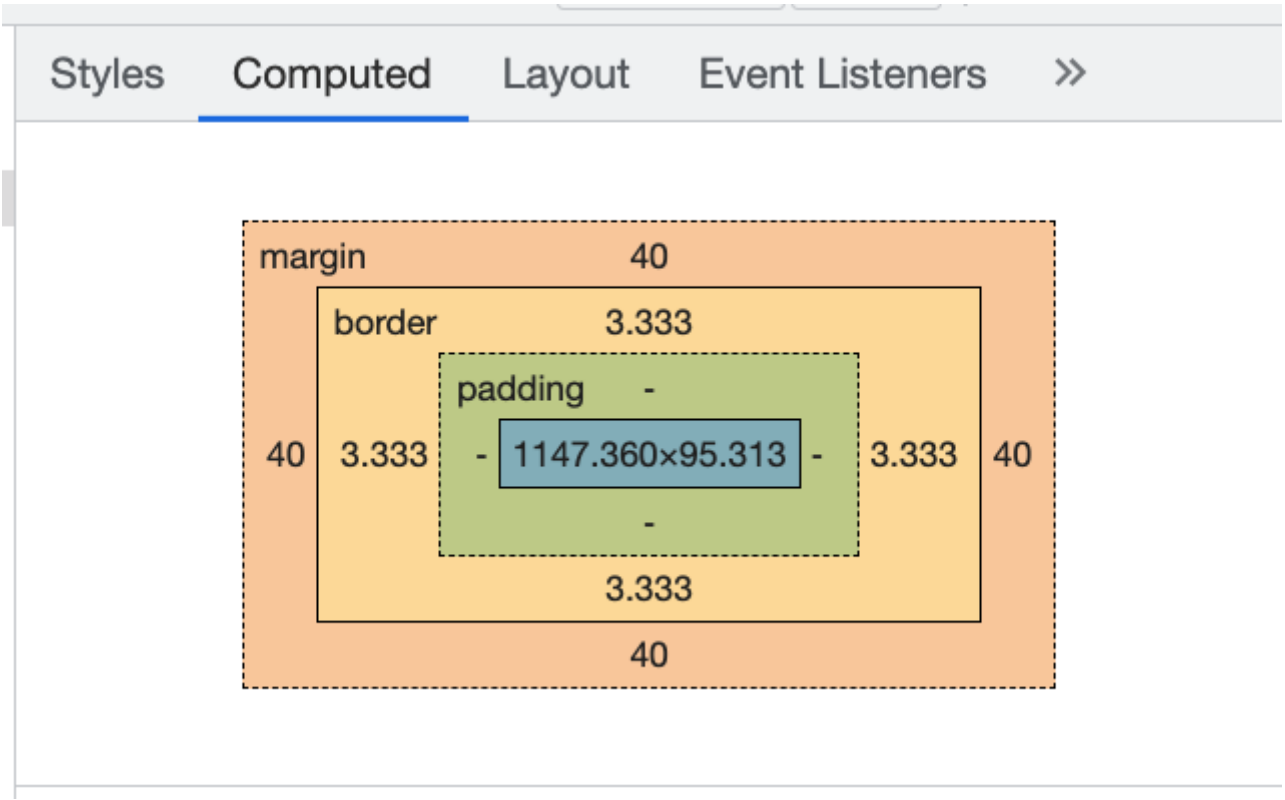
Listado propiedades no heredan https://www.sitepoint.com/css-inheritance-introduction/

Modelo de caja

Cualquier elemento del documento tiene una estructura de una caja rectangular.



Si te fijas esto lo puedes ver con las devtools:



¿Que significa esto? Que todos los elementos HTML tienen:

- Un Margen: espacio que hay desde los elementos hermanos hasta el borde.
- Un Padding: espacio que hay desde el borde al contenido del elemento.
- Contenido: El contenido en si.

Ojo a la hora de calcular tamaños en pixeles, hay una propiedad CSS que es *box-sizing*, que si está en *content box* (valor por defecto) sólo toma el contenido para calcular el ancho y alto del contenido, si no queremos este comportamiento podemos cambiar a *borderbox* (incluir *border* y *padding:* en el ancho y alto del elemento), cuando trabajamos con hojas de estilos de terceros suelen resetear los valores por defecto a *borderbox*, pincha aquí para saber más sobre borderbox.

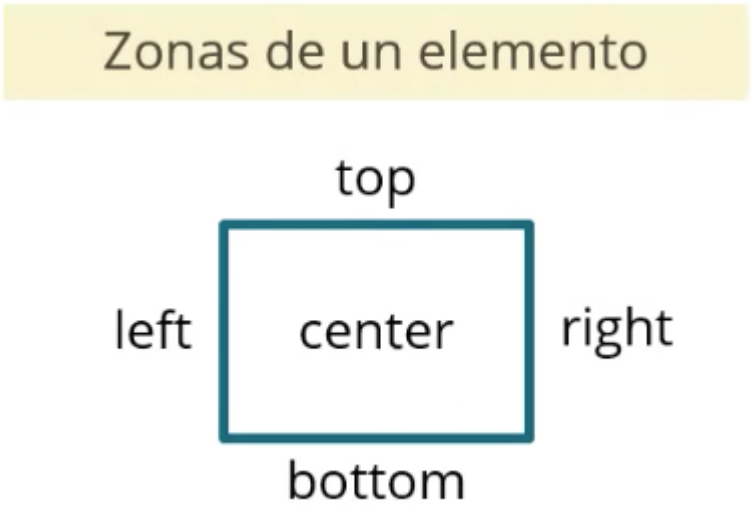
Cuando se limpian todas las propiedades se suele hacer de estas dos maneras:

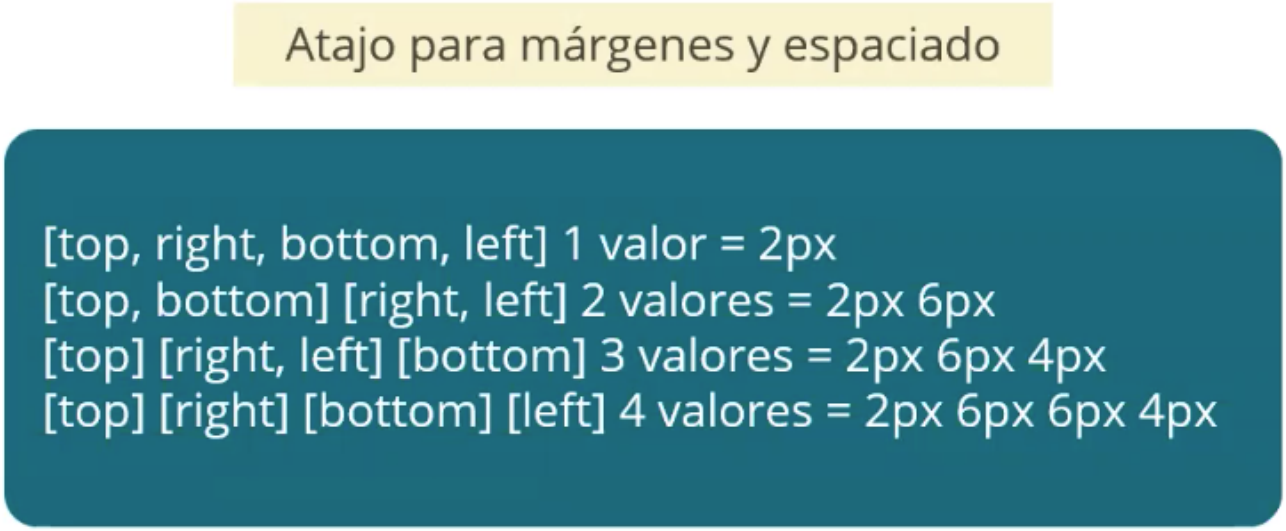
```
* {
  box-sizing: border-box;
}
```

```
html {
  box-sizing: border-box;
}
```

Lo ideal es elegir una opción para todo y ser consistente.

A tener en cuenta, podemos tocar sólo el padding del lado izquierdo, o el de abajo, o si son distintos definirlos de una vez usando atajos.





Por ejemplo podemos escribir lo siguiente:

```
ul li {
  padding-left: 5px; /*Estamos diciendo que el padding izdo va a ser de 5 pixeles*/
  margin: 0 3px; /* Decimos que top y bottom son 0 y right y left 3*/
  border-bottom: 3px solid black; /* top,right, left, bottom 3px y borde solido de color negro*/
}
```

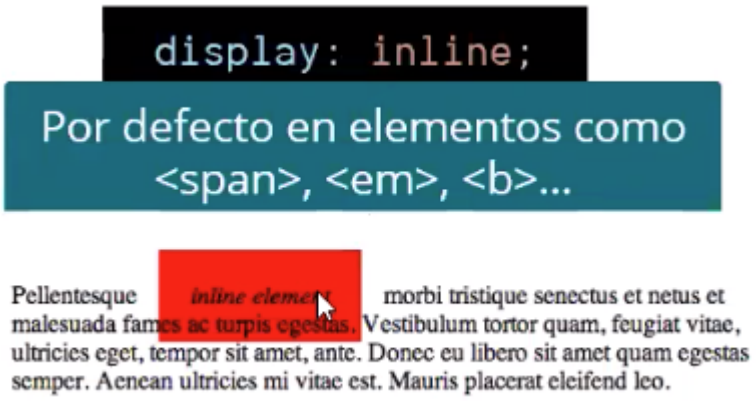
Vamos a ver ahora la propiedad display, esta propiedad controla el comportamiento de la caja tanto externo (como fluye en el documento) como interno (como se organizan los elementos hijos).

Tenemos:

- *display: inline*.
- *display: block*.
- *display: inline-block*.

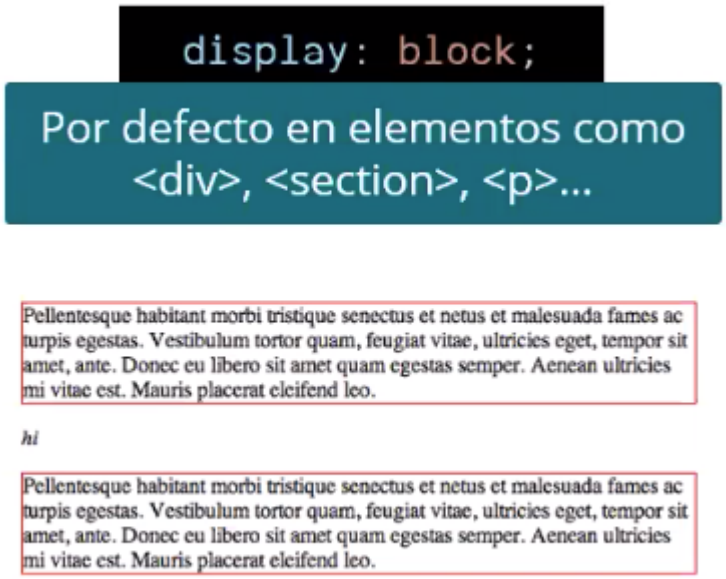
display: inline

Es el comportamiento que tenemos por defecto en elementos como **, **, *<b>*, es decir apila elementos uno al lado del otro (si no caben más caben a la siguiente línea).



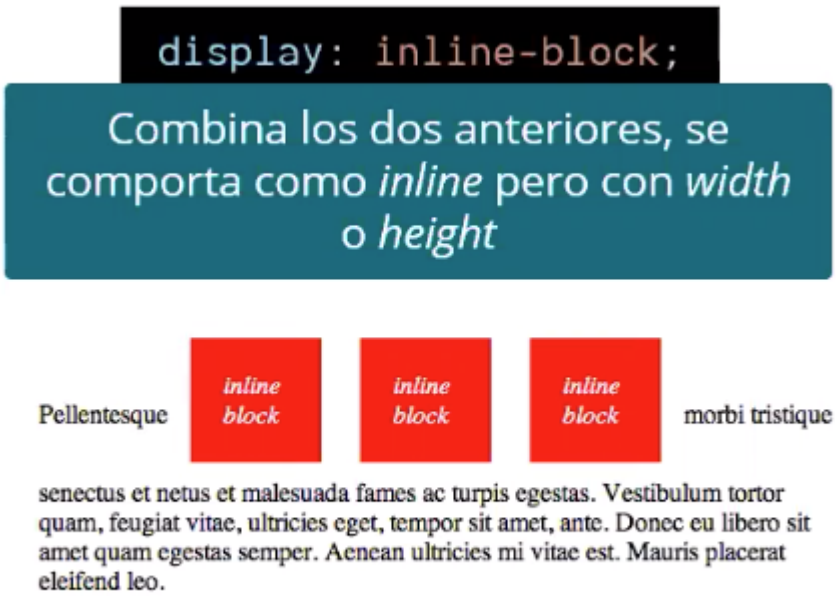
display: block

Es el comportamiento que tenemos por defecto en elemento como * *, *<section>*, *<p>*, es decir caen los elementos uno debajo del otro.



display: inline-block

Combina los dos anteriores, se comporta como *inline* pero con *width* o *height* (es decir en un span no le puedo dar ni ancho ni alto es lo que tome ese elemento, pero a ese *p* lo pongo en modo *inline-block* y pongo un width o un height si le puedo dar ancho y alto).



Vamos a jugar un poco con estos conceptos, borramos el contenido del *body* que tenemos en el HTML y del *css* sólo dejamos la fuente y el *body*.

Arrancamos creando dos párrafos:

index.html

```
<body>
+ <p>Hola, es <b>texto</b> importante</p>
+ <p>Hola, es <b>texto</b> importante</p>
</body>
```

Si visualizamos éste, vemos que un parrafo cae debajo del otro ya que por defecto se aplica el display *block*.

Hola, es **texto** importante

Hola, es **texto** importante

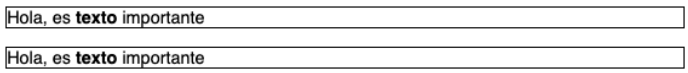
Podemos por ejemplo jugar con las propiedades, le vamos añadir un borde y decirle que el *width* sea del 50%

./src/miestilo.css

```
p {
  border: 1px solid black;
  display: block;
```

```
  width: 50%;
}
```

Vemos que el *width* si lo tiene en cuenta:



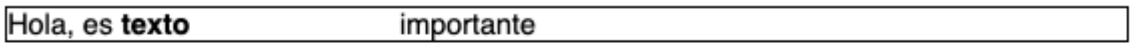
Vamos ahora a intentar poner el texto en negrita con un ancho de 150

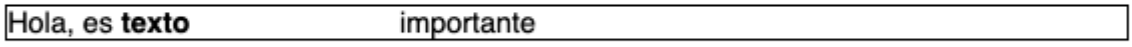
```
p {
  border: 1px solid black;
  dispay: block;
  width: 50%;
}

+ b {
+  width: 150px;
+ }
```

Este nuevo estilo que hemos metido no hace nada ¿Por qué? Porque por defecto el *b* se muestra como *display: inline* si queremos que tenga en cuenta el ancho lo tenemos que cambiar a *display: inline-block*

```
b {
+  display: inline-block;
  width: 150px;
}
```

Ahora si que se puede ver ese espacio

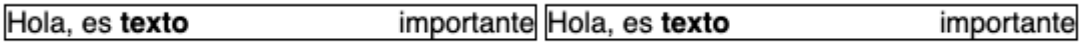




Si al estilo del párrafo le cambiaramos el estilo de *display* de bloque a *inline* los párrafos se apilarían uno al lado del otro:

```
p {
  border: 1px solid black;
-  display: block;
+  display: inline;
  width: 50%;
}
```

El resultado:



Posicionamiento

Para extraer un elemento del flujo normal del documento usamos la propiedad *position*, valores:

- *Static*: Se posiciona de acuerdo al flujo normal del documento (top, right, bottom, left y z-index no tiene efecto). Es el valor por defecto.
- *Relative*: Se posiciona de acuerdo al flujo normal del documento y luego es desplazado en relación a si mismo.
- *Absolute*: Están fuera del flujo normal y su posición final es determinada en base al primer padre posicionado por los valores top, right, bottom y left (el primer padre tiene que ser *relative* si no se iría hasta el body).
- *Fixed*: posicionamiento fijo, igual que el absoluto pero no se mueven aunque hagamos scroll.

Vamos a por unos ejemplos:

Limpiamos HTML y CSS, volvemos a la carga.

En el *index.html*, vamos a poner un montón de *lorem ipsum* para forzar a que haya scroll (para generar párrafos de *lorem ipsum* puedes usar esta tool: https://www.lipsum.com/), en está guía pegamos 5, pega tantos párrafos como te hagan falta para que te salga scroll.

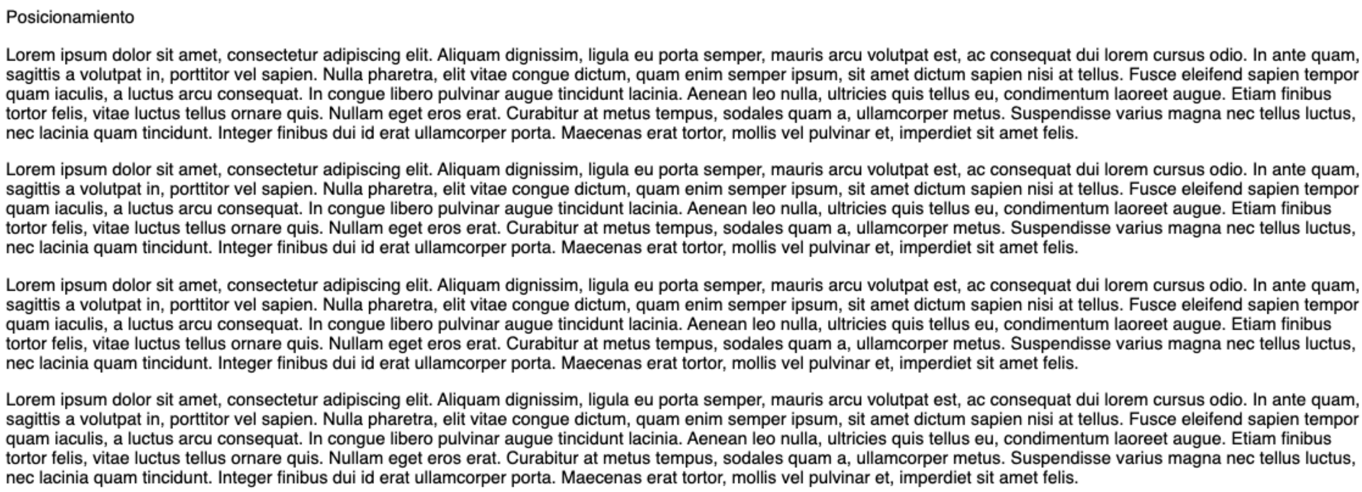
index.html

```
<body>
  <p>Posicionamiento</p>
  <div class="box"></div>
  <div class="box" id="element"><div>
  <div class="box"></div>
  <p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam
dignissim, ligula eu porta semper, mauris arcu volutpat est, ac consequat
dui lorem cursus odio. In ante quam, sagittis a volutpat in, porttitor vel
sapien. Nulla pharetra, elit vitae congue dictum, quam enim semper ipsum,
sit amet dictum sapien nisi at tellus. Fusce eleifend sapien tempor quam
iaculis, a luctus arcu consequat. In congue libero pulvinar augue
tincidunt lacinia. Aenean leo nulla, ultricies quis tellus eu, condimentum
laoreet augue. Etiam finibus tortor felis, vitae luctus tellus ornare
quis. Nullam eget eros erat. Curabitur at metus tempus, sodales quam a,
ullamcorper metus. Suspendisse varius magna nec tellus luctus, nec lacinia
quam tincidunt. Integer finibus dui id erat ullamcorper porta. Maecenas
erat tortor, mollis vel pulvinar et, imperdiet sit amet felis.
  </p>
  <p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam
```

```
dignissim, ligula eu porta semper, mauris arcu volutpat est, ac consequat
dui lorem cursus odio. In ante quam, sagittis a volutpat in, porttitor vel
sapien. Nulla pharetra, elit vitae congue dictum, quam enim semper ipsum,
sit amet dictum sapien nisi at tellus. Fusce eleifend sapien tempor quam
iaculis, a luctus arcu consequat. In congue libero pulvinar augue
tincidunt lacinia. Aenean leo nulla, ultricies quis tellus eu, condimentum
laoreet augue. Etiam finibus tortor felis, vitae luctus tellus ornare
quis. Nullam eget eros erat. Curabitur at metus tempus, sodales quam a,
ullamcorper metus. Suspendisse varius magna nec tellus luctus, nec lacinia
quam tincidunt. Integer finibus dui id erat ullamcorper porta. Maecenas
erat tortor, mollis vel pulvinar et, imperdiet sit amet felis.
  </p>
  <p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam
dignissim, ligula eu porta semper, mauris arcu volutpat est, ac consequat
dui lorem cursus odio. In ante quam, sagittis a volutpat in, porttitor vel
sapien. Nulla pharetra, elit vitae congue dictum, quam enim semper ipsum,
sit amet dictum sapien nisi at tellus. Fusce eleifend sapien tempor quam
iaculis, a luctus arcu consequat. In congue libero pulvinar augue
tincidunt lacinia. Aenean leo nulla, ultricies quis tellus eu, condimentum
laoreet augue. Etiam finibus tortor felis, vitae luctus tellus ornare
quis. Nullam eget eros erat. Curabitur at metus tempus, sodales quam a,
ullamcorper metus. Suspendisse varius magna nec tellus luctus, nec lacinia
quam tincidunt. Integer finibus dui id erat ullamcorper porta. Maecenas
erat tortor, mollis vel pulvinar et, imperdiet sit amet felis.
  </p>
  <p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam
dignissim, ligula eu porta semper, mauris arcu volutpat est, ac consequat
dui lorem cursus odio. In ante quam, sagittis a volutpat in, porttitor vel
sapien. Nulla pharetra, elit vitae congue dictum, quam enim semper ipsum,
sit amet dictum sapien nisi at tellus. Fusce eleifend sapien tempor quam
iaculis, a luctus arcu consequat. In congue libero pulvinar augue
tincidunt lacinia. Aenean leo nulla, ultricies quis tellus eu, condimentum
laoreet augue. Etiam finibus tortor felis, vitae luctus tellus ornare
quis. Nullam eget eros erat. Curabitur at metus tempus, sodales quam a,
ullamcorper metus. Suspendisse varius magna nec tellus luctus, nec lacinia
quam tincidunt. Integer finibus dui id erat ullamcorper porta. Maecenas
erat tortor, mollis vel pulvinar et, imperdiet sit amet felis.
  </p>
  <p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam
dignissim, ligula eu porta semper, mauris arcu volutpat est, ac consequat
dui lorem cursus odio. In ante quam, sagittis a volutpat in, porttitor vel
sapien. Nulla pharetra, elit vitae congue dictum, quam enim semper ipsum,
sit amet dictum sapien nisi at tellus. Fusce eleifend sapien tempor quam
iaculis, a luctus arcu consequat. In congue libero pulvinar augue
tincidunt lacinia. Aenean leo nulla, ultricies quis tellus eu, condimentum
laoreet augue. Etiam finibus tortor felis, vitae luctus tellus ornare
quis. Nullam eget eros erat. Curabitur at metus tempus, sodales quam a,
ullamcorper metus. Suspendisse varius magna nec tellus luctus, nec lacinia
quam tincidunt. Integer finibus dui id erat ullamcorper porta. Maecenas
erat tortor, mollis vel pulvinar et, imperdiet sit amet felis.
```

```
  </p>
</body>
```

Si borramos los estilos que había de antes y mostramos la página se vería tal que (añadiendo párrafos para forzar scroll):


Posicionamiento

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam dignissim, ligula eu porta semper, mauris arcu volutpat est, ac consequat dui lorem cursus odio. In ante quam, sagittis a volutpat in, porttitor vel sapien. Nulla pharetra, elit vitae congue dictum, quam enim semper ipsum, sit amet dictum sapien nisi at tellus. Fusce eleifend sapien tempor quam iaculis, a luctus arcu consequat. In congue libero pulvinar augue tincidunt lacinia. Aenean leo nulla, ultricies quis tellus eu, condimentum laoreet augue. Etiam finibus tortor felis, vitae luctus tellus ornare quis. Nullam eget eros erat. Curabitur at metus tempus, sodales quam a, ullamcorper metus. Suspendisse varius magna nec tellus luctus, nec lacinia quam tincidunt. Integer finibus dui id erat ullamcorper porta. Maecenas erat tortor, mollis vel pulvinar et, imperdiet sit amet felis.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam dignissim, ligula eu porta semper, mauris arcu volutpat est, ac consequat dui lorem cursus odio. In ante quam, sagittis a volutpat in, porttitor vel sapien. Nulla pharetra, elit vitae congue dictum, quam enim semper ipsum, sit amet dictum sapien nisi at tellus. Fusce eleifend sapien tempor quam iaculis, a luctus arcu consequat. In congue libero pulvinar augue tincidunt lacinia. Aenean leo nulla, ultricies quis tellus eu, condimentum laoreet augue. Etiam finibus tortor felis, vitae luctus tellus ornare quis. Nullam eget eros erat. Curabitur at metus tempus, sodales quam a, ullamcorper metus. Suspendisse varius magna nec tellus luctus, nec lacinia quam tincidunt. Integer finibus dui id erat ullamcorper porta. Maecenas erat tortor, mollis vel pulvinar et, imperdiet sit amet felis.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam dignissim, ligula eu porta semper, mauris arcu volutpat est, ac consequat dui lorem cursus odio. In ante quam, sagittis a volutpat in, porttitor vel sapien. Nulla pharetra, elit vitae congue dictum, quam enim semper ipsum, sit amet dictum sapien nisi at tellus. Fusce eleifend sapien tempor quam iaculis, a luctus arcu consequat. In congue libero pulvinar augue tincidunt lacinia. Aenean leo nulla, ultricies quis tellus eu, condimentum laoreet augue. Etiam finibus tortor felis, vitae luctus tellus ornare quis. Nullam eget eros erat. Curabitur at metus tempus, sodales quam a, ullamcorper metus. Suspendisse varius magna nec tellus luctus, nec lacinia quam tincidunt. Integer finibus dui id erat ullamcorper porta. Maecenas erat tortor, mollis vel pulvinar et, imperdiet sit amet felis.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam dignissim, ligula eu porta semper, mauris arcu volutpat est, ac consequat dui lorem cursus odio. In ante quam, sagittis a volutpat in, porttitor vel sapien. Nulla pharetra, elit vitae congue dictum, quam enim semper ipsum, sit amet dictum sapien nisi at tellus. Fusce eleifend sapien tempor quam iaculis, a luctus arcu consequat. In congue libero pulvinar augue tincidunt lacinia. Aenean leo nulla, ultricies quis tellus eu, condimentum laoreet augue. Etiam finibus tortor felis, vitae luctus tellus ornare quis. Nullam eget eros erat. Curabitur at metus tempus, sodales quam a, ullamcorper metus. Suspendisse varius magna nec tellus luctus, nec lacinia quam tincidunt. Integer finibus dui id erat ullamcorper porta. Maecenas erat tortor, mollis vel pulvinar et, imperdiet sit amet felis.

Vamos a jugar con el *div* que esta marcado con el *id element*.

Arrancamos por definir, el selector para las clase *box*, en este caso le daremos 100 pixeles de ancho y alto, y un margen en la parte de abajo de la caja de 100 pixeles de alto, y un color de fondo azul y un borde de 3 pixeles (para poder visualizar el espacio que ocupan)

```css
.box {
  width: 100px;
  height: 100px;
  margin-bottom: 10px;
  background-color: lightblue;
  border: 3px solid darkblue;
}
```

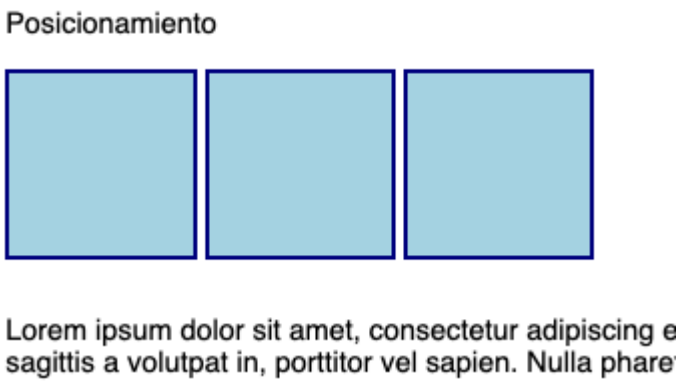
Esto se ve tal que así ahora:

Display default block

Vamos a añadirle un display *inline-block* para que fluyan de izquierda a derecha:

```diff
.box {
+ display: inline-block;
  width: 100px;
  height: 100px;
  margin-bottom: 10px;
  background-color: lightblue;
  border: 3px solid darkblue;
}
```

Con lo que se nos queda el siguiente aspecto:

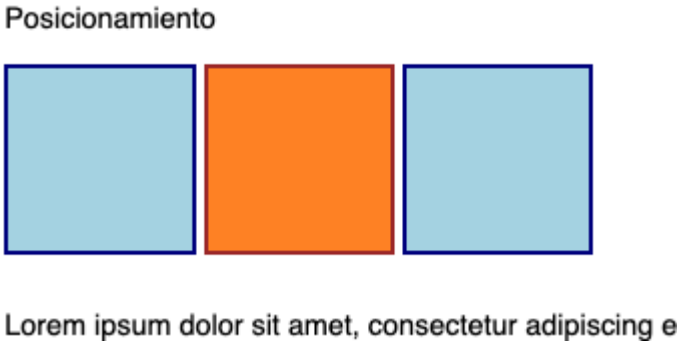


Vamos ahora a por el id *element* del div que hemos creado en el HTML:

Añadimos al CSS:

```
.box {
  display: inline-block;
  width: 100px;
  height: 100px;
  margin-bottom: 10px;
  background-color: lightblue;
  border: 3px solid darkblue;
}

+ #element {
+   background-color: darkorange;
+   border: 3px solid brown;
+ }
```

Ya tenemos el div de en medio marcado con un color distinto:



Vamos a empezar a jugar con la posición de ese elemento.

El position que tenemos por defecto es el *static* si le pongo ese estilo, el layout se queda como está.

```
 #element {
+   position: static;
    background-color: darkorange;
```

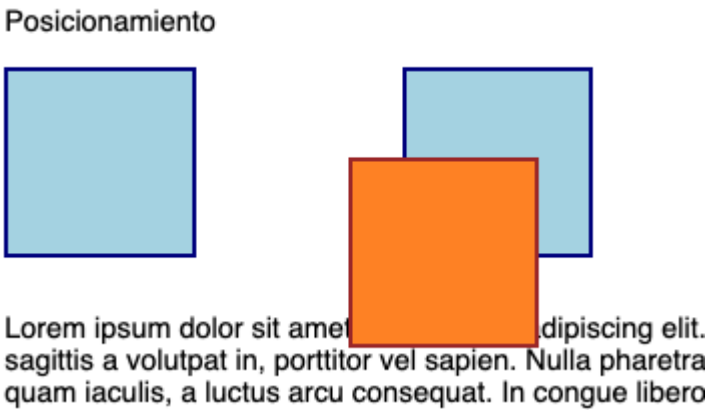
```
    border: 3px solid brown;
  }
```

Si intento moverlo usando las propiedades *top* o similares no se va a mover, ya que no se considera que es un elemento posicionado, si quisiera desplazarlo tendría que jugar con propiedades como *margin*, *padding*

¿Y si fuera un relative? Aquí el elemento sigue en el flujo normal del documento, pero se va a posicionar respecto a si mismo, aquí si podemos desplazarlo por ejemplo 50 pixeles con top y left.

¿Que hace esto? El elemento se posiciona en su sitio y después se ajusta su posición teniendo en cuenta el *top*, *left* ... que la hayamos indicado, por ejemplo:

```
 #element {
-   position: static;
+   position: relative;
+   top: 120px;
+   left: 80px;
    background-color: darkorange;
    border: 3px solid brown;
 }
```



Un tema interesante: aunque separáramos mucho este *div* los elemento hermanos respetarían el espacio original que ocupaba ese *div*, esta es una diferencia importante de usar *absolute* vs *relative*.

Si cambiamos ahora a *absolute* veremos que el elemento:

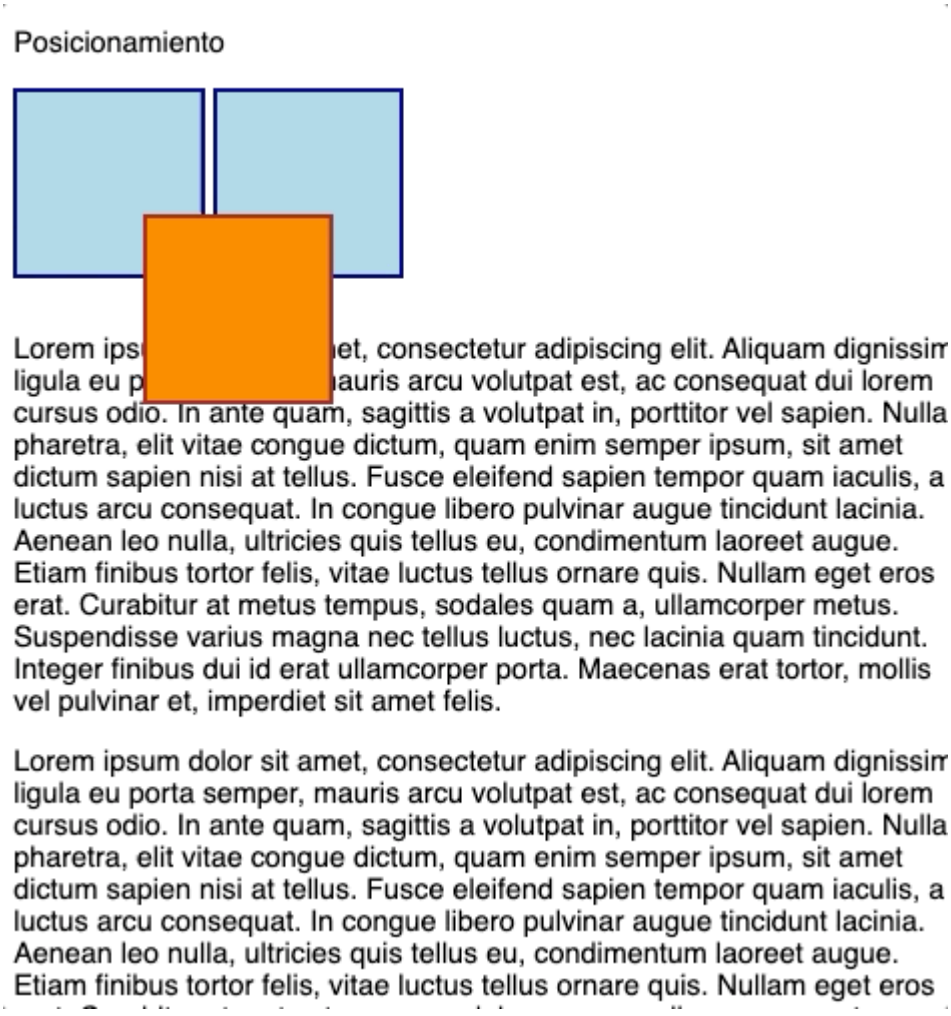
- Se mueve pero con coordenadas absolutas.
- Se desplaza si hacemos scroll.
- Toma como referencia el primer padre posicionado (en este caso sería el elemento *body*, ya que el div *container padre* no esta posicionado), si posicionaramos el div padre *container* con un *relative*, tomaría como punto de partida ese div.
- Para los hermanos ese elemento ya no está ahí y ocupan su espacio.

Veamoslo:

```
 #element {
-   position: relative;
+   position: absolute;
    top: 120px;
    left: 80px;
    background-color: darkorange;
    border: 3px solid brown;
 }
```

Vamos ahora por el *position* que nos queda: *fixed* en este caso el elemento se queda en esa posición *siempre* hagamos o no scroll (además sale del flujo normal, los hermanos ocupan su espacio).

```
 #element {
-   position: absolute;
+   position: fixed;

    top: 120px;
    left: 80px;
    background-color: darkorange;
    border: 3px solid brown;
 }
```

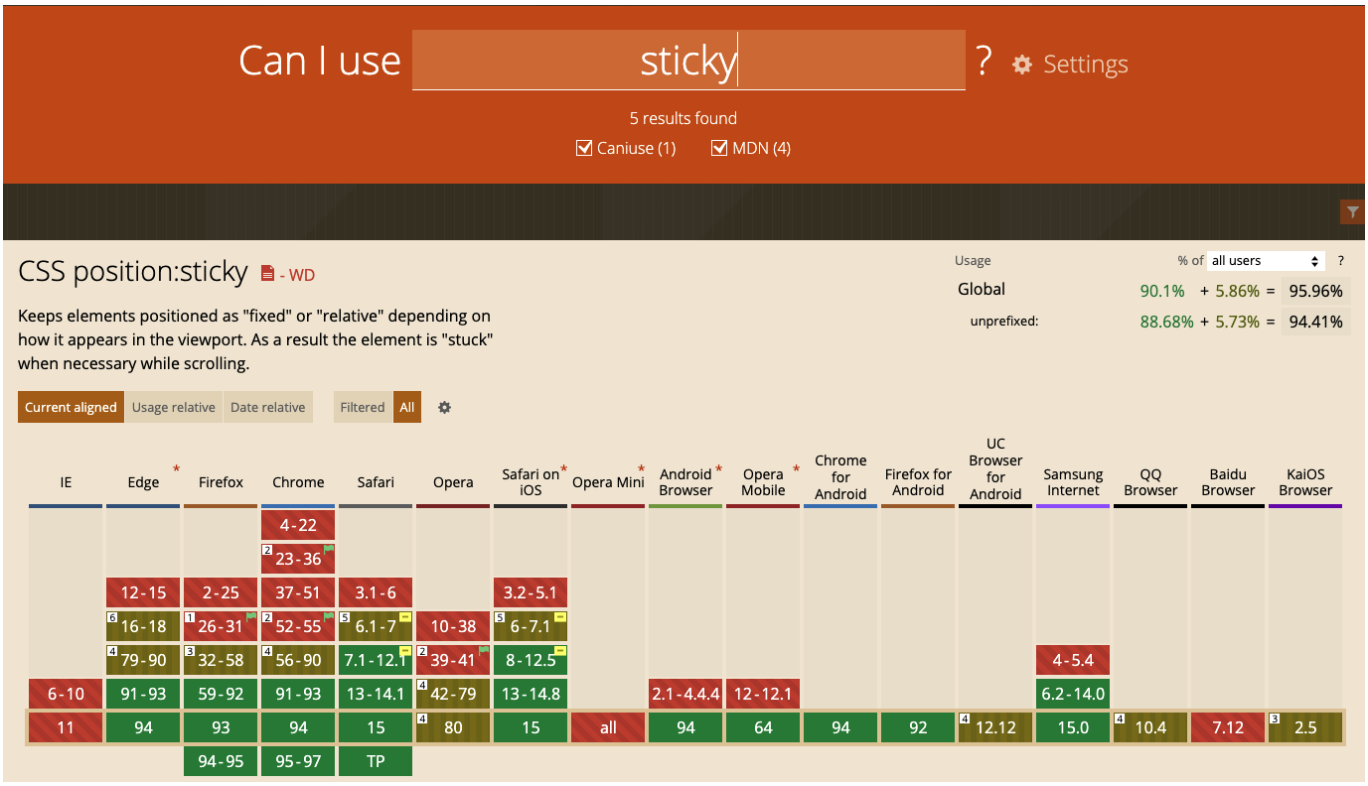


¿Que aplicación puede tener esto? Te puede valer para poner la típica cabecera que tiene el menú y los datos del usuario y queremos que este siempre visible.

Otro escenario muy interesante que habrás visto en algunos sitio es el de un elemento HTML que acompaña con el scroll, hasta que llega un punto en el que se queda fija, esto se consigue con el *position sticky*, vamos a cambiar la posición de nuestro *div* y decirle que cuando la barra de scroll llegue al tope, se quede siempre visible en la posición cero.

```
 #element {
-   position: fixed;
+   position: sticky;

-   top: 120px;
+   top: 0px;
-   left: 80px;
+   left: 50px;

    background-color: darkorange;
    border: 3px solid brown;
 }
```

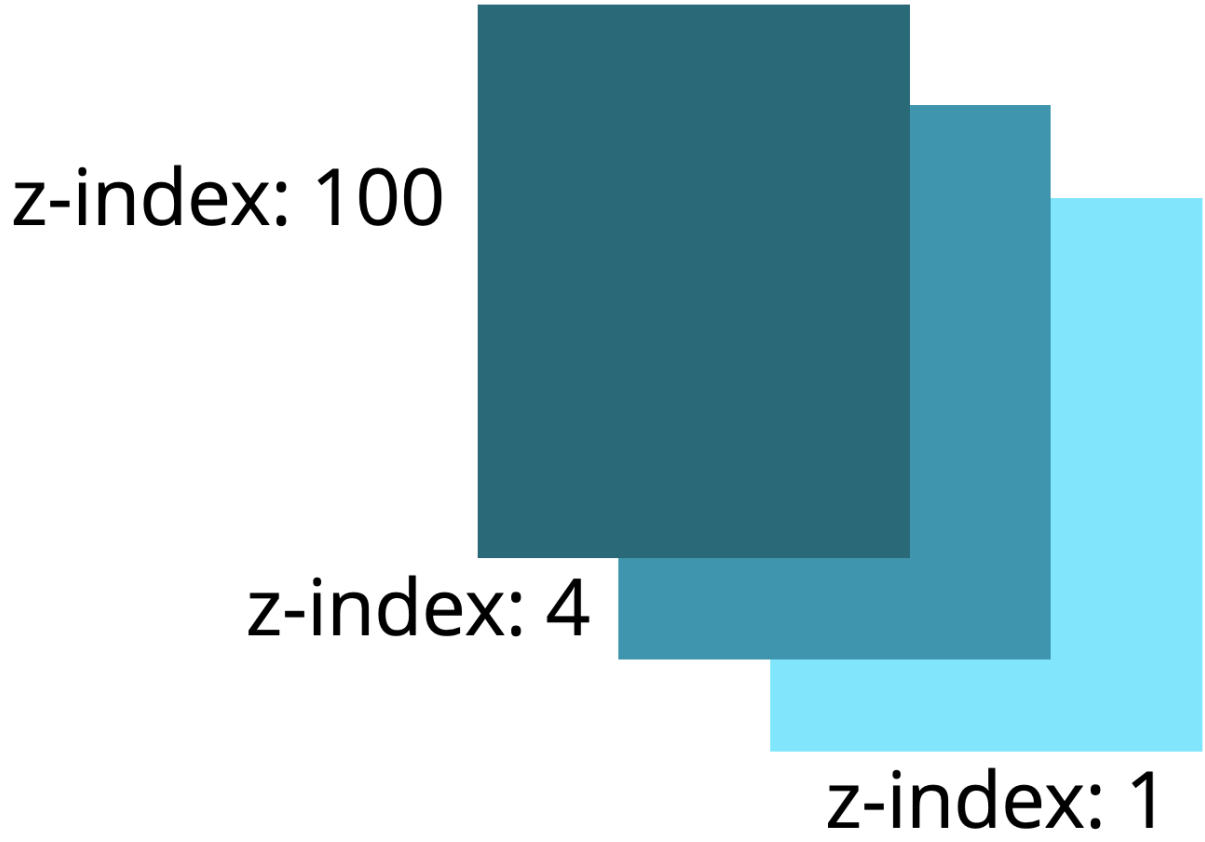
Peor hay que tener cuidado, si miramos la página "can I use" nos puede decir que soporte tiene esto para todos los navegadores.



Z-Index

Sirve para indicar el nivel de profundidad en la que está un elemento sobre los demás.

Es decir ya no es posición X o Y, si no un nuevo eje, ponemos elementos uno encima del otro. A mayor z-index más arriba aparece. Por ejemplo queremos desplegar un menú que se vea sobre el contenido de la ventana.



Vamos a por un ejemplo, añadimos:

./index.html

```
  <body>
    <p>Posicionamiento</p>
    <div class="box"></div>
    <div class="box" id="element"></div>
-    <div class="box"></div>
+    <div class="box" id="element2"></div>

    <p>
```

Vamos a cambiar el *position* de *#element* a *relative*, y le añadimos algo de margen para que se solape con el siguiente elemento.

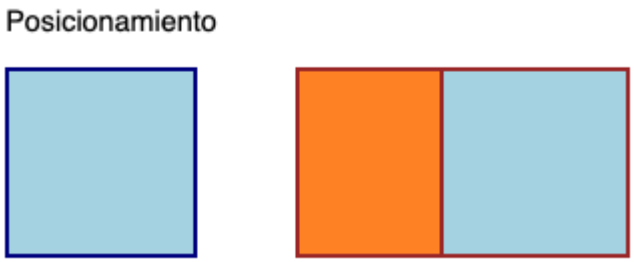
./src/miestilo.css

```
#element {
-  position: sticky;
+  position: relative;
   top: 0px;
+
   background-color: darkorange;
   border: 3px solid brown;
}
```

Añadimos el estilado para el *#element2*, lo ponemos relative y lo movemos 40 pixeles para que se vea la superposición:

```
#element {
  position: relative;
  top: 0px;
  background-color: darkorange;
  border: 3px solid brown;
}

+ #element2 {
+  border: 3px solid brown;
+  position: relative;
+  top: 0px;
+  left: 20px;
+  background-color: lightblue;
+ }
```

Fíjate que *#element2* se superpone sobre *#element1* eso es porque tienen el mismo *z-index* (valor por defecto), pero *#element2* gana al estar definido más adelante en el HTML.



Si queremos que pase al contrario, podemos darle un *z-index* mas alto al primero elemento, veamos como:

```
#element {
  position: relative;
  top: 0px;
  left: 50px;
  background-color: darkorange;
  border: 3px solid brown;
```

```
+ z-index: 2;
}

#element2 {
  border: 3px solid brown;
  position: relative;
  top: 0px;
  left: 20px;
  background-color: lightblue;
+ z-index: 1;
}
```

Posicionamiento

Cuando queremos poner algo por encima de todo (por ejemplo un modal), ponemos un z-index con un valor grande.