

Definición de CSS.

CSS (siglas en inglés de **Cascading Style Sheets**), en español «Hojas de estilo en cascada», es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado.² Es muy usado para establecer el diseño visual de los documentos web, e interfaces de usuario escritas en HTML o XHTML; el lenguaje puede ser aplicado a cualquier documento XML, incluyendo XHTML, SVG, XUL, RSS, etcétera. Junto con HTML y JavaScript, CSS es una tecnología usada por muchos sitios web para crear páginas visualmente atractivas, interfaces de usuario para aplicaciones web y GUIs para muchas aplicaciones móviles (como Firefox OS).

CSS está diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de este, características tales como las capas o *layouts*, los colores y las fuentes.⁴ Esta separación busca mejorar la accesibilidad del documento, proveer más flexibilidad y control en la especificación de características presentacionales, permitir que varios documentos HTML compartan un mismo estilo usando una sola hoja de estilos separada en un archivo `.css`, y reducir la complejidad y la repetición de código en la estructura del documento.

La separación del formato y el contenido hace posible presentar el mismo documento marcado en diferentes estilos para diferentes métodos de renderizado, como en pantalla, en impresión, en voz (mediante un navegador de voz o un lector de pantalla), y dispositivos táctiles basados en el sistema Braille. También se puede mostrar una página web de manera diferente dependiendo del tamaño de la pantalla o tipo de dispositivo. Los lectores pueden especificar una hoja de estilos diferente, como una hoja de estilos CSS guardado en su computadora, para sobrescribir la hoja de estilos del diseñador.

La especificación CSS describe un esquema prioritario para determinar qué reglas de estilo se aplican si más de una regla coincide para un elemento en particular. Estas reglas son aplicadas con un sistema llamado *de cascada*, de modo que las prioridades son calculadas y asignadas a las reglas, así que los resultados son predecibles.

Funcionamiento Básico de CSS

CSS permite separar los contenidos de la página y la información sobre su aspecto. En el ejemplo anterior, dentro de la propia página HTML se crea una zona especial en la que se incluye toda la información relacionada con los estilos de la página.

Utilizando CSS, se pueden establecer los mismos estilos con menos esfuerzo y sin *ensuciar* el código HTML de los contenidos con etiquetas ``. Como se verá más adelante, la etiqueta `<style>` crea una zona especial donde se incluyen todas las reglas CSS que se aplican en la página.

Definir los estilos de esta forma ahorra miles de etiquetas y millones de atributos respecto a la solución anterior, pero sigue sin ser una solución ideal. Como los estilos CSS sólo se aplican en la página que los incluye, si queremos que las 10.000 páginas diferentes del sitio tengan el mismo aspecto, se deberían copiar 10.000 veces esas mismas reglas CSS. Más adelante se explica la solución que propone CSS para evitar este problema.

Cómo incluir CSS en un documento XHTML

Una de las principales características de CSS es su flexibilidad y las diferentes opciones que ofrece para realizar una misma tarea. De hecho, existen tres opciones para incluir CSS en un documento HTML.

a). Incluir CSS en el mismo documento HTML

Los estilos se definen en una zona específica del propio documento HTML. Se emplea la etiqueta <style> de HTML y solamente se pueden incluir en la cabecera del documento (sólo dentro de la sección <head>).

Este método se emplea cuando se define un número pequeño de estilos o cuando se quieren incluir estilos específicos en una determinada página HTML que completen los estilos que se incluyen por defecto en todas las páginas del sitio web.

El principal inconveniente es que si se quiere hacer una modificación en los estilos definidos, es necesario modificar todas las páginas que incluyen el estilo que se va a modificar.

b). Definir CSS en un archivo externo

En este caso, todos los estilos CSS se incluyen en un archivo de tipo CSS que las páginas HTML enlazan mediante la etiqueta <link>. Un archivo de tipo CSS no es más que un archivo simple de texto cuya extensión es .css. Se pueden crear todos los archivos CSS que sean necesarios y cada página HTML puede enlazar tantos archivos CSS como necesite.

Si se quieren incluir los estilos del ejemplo anterior en un archivo CSS externo, se deben seguir los siguientes pasos:

1) Se crea un archivo de texto y se le añade solamente el siguiente contenido:

```
p { color: black; font-family: Verdana; }
```

2) Se guarda el archivo de texto con el nombre estilos.css. Se debe poner especial atención a que el archivo tenga extensión .css y no .txt.

3) En la página HTML se enlaza el archivo CSS externo mediante la etiqueta <link>:

Cuando el navegador carga la página HTML anterior, antes de mostrar sus contenidos también descarga los archivos CSS externos enlazados mediante la etiqueta <link> y aplica los estilos a los contenidos de la página.

Normalmente, la etiqueta <link> incluye cuatro atributos cuando enlaza un archivo CSS:

- rel: indica el tipo de relación que existe entre el recurso enlazado (en este caso, el archivo CSS) y la página HTML. Para los archivos CSS, siempre se utiliza el valor stylesheet
- type: indica el tipo de recurso enlazado. Sus valores están estandarizados y para los archivos CSS su valor siempre es text/css
- href: indica la URL del archivo CSS que contiene los estilos. La URL indicada puede ser relativa o absoluta y puede apuntar a un recurso interno o externo al sitio web.
- media: indica el medio en el que se van a aplicar los estilos del archivo CSS. Más adelante se explican en detalle los medios CSS y su funcionamiento.

De todas las formas de incluir CSS en las páginas HTML, esta es la más utilizada con mucha diferencia. La principal ventaja es que se puede incluir un mismo archivo CSS en multitud de páginas HTML, por lo que se garantiza la aplicación homogénea de los mismos estilos a todas las páginas que forman un sitio web.

Con este método, el mantenimiento del sitio web se simplifica al máximo, ya que un solo cambio en un solo archivo CSS permite variar de forma instantánea los estilos de todas las páginas HTML que enlazan ese archivo.

Aunque generalmente se emplea la etiqueta <link> para enlazar los archivos CSS externos, también se puede utilizar la etiqueta <style>. La forma alternativa de incluir un archivo CSS externo se muestra a continuación:

En este caso, para incluir en la página HTML los estilos definidos en archivos CSS externos se utiliza una regla especial de tipo @import. Las reglas de tipo @import siempre preceden a cualquier otra regla CSS (con la única excepción de la regla @charset).

La URL del archivo CSS externo se indica mediante una cadena de texto encerrada con comillas simples o dobles o mediante la palabra reservada url(). De esta forma, las siguientes reglas @import son equivalentes:

```
@import 'css/estilos.css';  
@import "css/estilos.css";  
@import url('css/estilos.css');  
@import url("css/estilos.css");
```

c). Incluir CSS en los elementos HTML

El último método para incluir estilos CSS en documentos HTML es el peor y el menos utilizado, ya que tiene los mismos problemas que la utilización de las etiquetas .

Esta forma de incluir CSS directamente en los elementos HTML solamente se utiliza en determinadas situaciones en las que se debe incluir un estilo muy específico para un solo elemento concreto.

En el lenguaje CSS se define casi todo el aspecto visual de un sitio. Se escribe en un único archivo usualmente llamado **styles.css** y todos los archivos HTML tienen un vínculo hacia él.

Sintaxis de una regla CSS

Todo el código CSS se compone de reglas. **Una regla es el conjunto de propiedades** que se van a aplicar a un elemento determinado

Definir regla CSS

En el lenguaje CSS se define casi todo el aspecto visual de un sitio. Se escribe en un único archivo usualmente llamado styles.css y todos los archivos HTML tienen un vínculo hacia él.

Sintaxis de una regla CSS

Todo el código CSS se compone de reglas. Una regla es el conjunto de propiedades que se van a aplicar a un elemento determinado.

Una regla CSS siempre tiene un selector (el elemento al que quiero modificar) y una declaración (las características que le quiero agregar o cambiar). La declaración va encerrada entre llaves { } y dentro de ella se escriben todas las propiedades con sus valores, que modificarán al selector. Cuando hay más de una propiedad, se separan con punto y coma.

La sintaxis se ve de esta manera:

regla css

Nota: después de los dos puntos y del punto y coma pueden ir espacios o no, es indistinto. Después de la última propiedad, puede ir o no un punto y coma. Las propiedades se pueden escribir una debajo de la otra o todas en una misma línea como está en la imagen. Nosotros escribimos en una sola línea para que el código quede más compacto.

Un archivo CSS puede tener un número ilimitado de reglas CSS y dentro de cada regla podemos escribir tantas propiedades como necesitemos. Una misma regla se puede utilizar sobre varias etiquetas del HTML

Tipos de Selectores

El selector es la parte de la regla que asocia un elemento HTML con su diseño. Básicamente los selectores se pueden clasificar en 4 grupos: de etiqueta, de clase, de id y compuestos.

Al primero al que recurrimos es al selector de etiqueta, porque aprovechamos las etiquetas que ya están escritas en el html. Por ejemplo: si en el HTML tenemos la etiqueta `<body>`, para crear una regla en CSS simplemente escribo el nombre de la etiqueta y coloco las propiedades entre las llaves: `body {}`.

Pero cuando tenemos varias etiquetas iguales y sólo queremos darle diseño a una de ellas, entonces necesitamos identificarla de alguna manera y tenemos dos opciones: a esa etiqueta en el HTML le podemos agregar el atributo `class=""` por ejemplo: `<article class="destacado">` y en ese caso en el CSS el selector comienza con un punto: `.destacado {}` o el atributo `id=""` por ejemplo: `<article id="destacado">` entonces el selector comienza con numeral: `#destacado {}`

La diferencia entre un estilo de clase y uno de id es que en un archivo HTML puede haber varias etiquetas con el mismo class, en cambio no puede haber dos etiquetas con el mismo id.

Muchas veces el selector simple no nos alcanza para identificar un elemento en particular. Por ejemplo si queremos definir un diseño para la etiqueta `<a>` del menú principal pero no queremos que afecte el diseño de las otras etiquetas `<a>` del sitio. Entonces usamos un selector compuesto. En este caso el selector descendente, por ejemplo: `nav a {}`. Con este selector estamos diciendo que la regla se aplica a todas las etiquetas `<a>` que se encuentren dentro de una etiqueta `<nav>`. Entonces esa regla no afecta al resto de los enlaces del sitio.

Elemento de CSS

Elementos modificables con CSS O mejor dicho, atributos de elementos que pueden modificarse con CSS. Son muchos (casi todos) los aspectos que pueden modificarse con CSS. Dado el carácter general de esta guía, aquí veremos solamente los más utilizados.

Generalmente se actúa sobre los siguientes:

- Texto o contenido
- Tamaño del bloque
- Color del fondo del bloque
- Bordes de bloque
- Estilo de los bordes
- Color de los bordes
- Márgenes de bloque
- Espaciado interno del bloque
- Posicionamiento del bloque
- Gráficos y bloques flotantes
- Visibilidad del bloque
- Listas
- Enlaces
- CSS + JavaScript
- Efectos especiales
- Tablas

Y puede que te preguntes qué es un bloque. Como bloque puede entenderse todo lo comprendido dentro de elementos con cierre, como `<body> </body>`, `<p> </p>`, `<form> </form>`, `<table> </table>`, `<td> </td>`, `<div> </div>`, etc., y al aplicar estilos con CSS no es buena idea escribir nada fuera de estos elementos. En lugar de ser un nombre de elemento HTML, un bloque también puede tener un nombre definido por el programador, por ejemplo "menu", y generalmente son éstos los que contienen en su interior a los anteriores o a otros bloques definidos por el usuario. El número de bloques en una página puede ser muy elevado (más de mil), y dependerá de la capacidad del navegador utilizado.

Veamos cómo funciona cada cosa:

Los Márgenes de bloque son el espacio comprendido entre el bloque y el borde de la ventana activa del navegador. Se controla con el atributo `margin`.

Los Bordes de bloque, sin aplicar estilos, no son visibles, y es como un cuadro imaginario que envuelve todo el contenido del bloque. Su atributo de control es `border`.

Espaciado interno del bloque es la distancia entre el borde del bloque y su contenido. Es el atributo `padding` (en inglés significa algo así como "acolchado")

El Estilo y color de los bordes, son evidentes: son las líneas que delimitan el bloque, que pueden dibujarse de varias formas en cuanto a tipo de trazo, grosor y color. Estos dos atributos también pueden actuar sobre los bordes de otros elementos contenidos dentro de un bloque, como formularios, tablas, gráficos, etc.

El Color del fondo se controla con las mismas instrucciones que las de la página HTML: `background`. Los parámetros de color, al igual que en HTML, se pueden escribir con su nombre (en inglés) o con la notación RGB en hexadecimal.

El Tamaño son las dimensiones del bloque. Si no se indican dimensiones, por defecto, el bloque ocupará todo el ancho de la ventana, y de alto lo que su contenido precise. Se utilizan dos atributos para controlarlo: `width` para el ancho y `height` para el alto.

Estilo del texto

Veamos los parámetros disponibles para dar estilo al texto. Como ya sabes, las unidades de medida aplicables a todos ellos son varias, pero para mayor claridad, en todos los ejemplos utilizaremos solamente el píxel: `px`. Sea por ejemplo la siguiente página:

```
P {color:red;
  font-size:20px;
  font-family:Courier;
  font-weight:bold;
  font-style:italic;
  line-height:30px;
  letter-spacing:5px;
  text-decoration:underline;
  text-transform:capitalize;
  text-align:left;
  text-indent:30px;
}
```

Se obtiene:

Texto fuera de párrafo.
Segunda línea fuera de párrafo.

Texto De Párrafo.

Segunda Línea De Párrafo

Fíjate en la gran diferencia de estilo que hay entre las dos primeras líneas y las dos siguientes. Analicemos cómo funciona: Las dos primeras líneas de texto, en color verde, tienen todos sus valores por defecto, excepto el color que lo reciben de la etiqueta BODY definida en la css. Las dos siguientes, en rojo, reciben todas sus características de la etiqueta P, también definida en la css. Como puedes ver, hay una gran cantidad de atributos que actúan sobre el estilo de ese texto.

color:red; Este ya lo conocemos. Define el color del texto. El color deseado puede escribirse directamente, en inglés, o puede utilizarse el sistema RGB en hexadecimal. Por ejemplo, este mismo color rojo en RGB sería: #FF0000; Ciertos valores, como el rojo, pueden escribirse de forma abreviada: #F00; Otra forma de declarar el color es mediante la función rgb() que puede parametrizarse en decimal o en tantos por ciento. He aquí un ejemplo con las cuatro formas del color rojo:

```
color: #ff0000
color: #f00
color: rgb(255,0,0)
color: rgb(100%, 0%, 0%)
```

Recuerda que en el índice tienes unas tablas con los nombres y códigos de los colores.

font-size:20px; También muy conocido: define el tamaño de los caracteres. Además del tamaño definido por el usuario, expresado en cualquiera de las unidades de medida conocidas, puede tener alguno de los siguientes valores absolutos: xx-large x-large large medium small x-small xx-small.

font-family:Courier; Indica el nombre (o nombres) del tipo de letra que se va a emplear. En el ejemplo se ha utilizado el tipo "Courier", pero pueden escribirse varios separados por comas, lo que indica al navegador que si no existe en la máquina el primer tipo, utilice el segundo, y si tampoco, el tercero, etc. Por ejemplo: font-family:Courier,Verdana,Arial; Cuando definas tipos de letra, recuerda que muchos programas y algunos drivers de impresora instalan sus propias fuentes en el sistema sin avisarte, y puede que en tu máquina haya tipos que no son estándar en Windows. No se deben utilizar tipos extraños que difícilmente se encontrarán en las máquinas de los clientes, es decir, el tipo de fuente indicado en la hoja simplemente le indica al navegador qué fuente debe usar, NO se la sirve. En la versión 3 de CSS parece que se podrá indicar al cliente dónde obtener una fuente que no tenga instalada.

font-weight:bold; Aquí se especifica el peso o grosor de la fuente. Pueden emplearse literales como lighter, normal (por defecto) o bold. También se pueden utilizar números entre 100 y 900, escritos de 100 en 100 (no sirve 190, pero sí 200). Los pesos no tienen los mismos resultados en todas las máquinas, ya que depende mucho de la calidad de su pantalla, de su resolución, del navegador que emplea... En cualquier caso, no deben hacerse combinaciones extrañas, como definir un size muy pequeño y darle peso 900, ya que lo único que conseguirás es un borrón perfectamente ilegible.

`font-style:italic;` Solamente tiene dos posibilidades: normal (por defecto) o italic que es el empleado en el ejemplo que estamos analizando, y que hace que el texto tenga cierta inclinación.

`line-height:30px;` Sirve para establecer la distancia entre líneas consecutivas de un mismo párrafo. Además de en píxels o alguna de las unidades de medida que ya conocemos, puede indicarse con un simple número que viene a indicar, aproximadamente, cuantas anchuras de línea se tomarán como unidad de medida para separar una línea de otra. Así, por ejemplo, si pones 0 la segunda línea se superpone a la primera.

`letter-spacing:5px;` Establece la separación entre los caracteres de la línea.

`word-spacing:5px;` Establece la separación entre las palabras de la línea.

`text-decoration:underline;` Con este parámetro, que no influye en el tamaño, puedes acompañar al texto de una delgada línea decorativa que puede estar en tres posiciones distintas, como underline (el típico subrayado debajo de la línea), through (en el centro de la línea -tachado-) o overline (encima del texto). Para que no aparezca se utiliza none (por defecto). El valor none se puede utilizar para eliminar el efecto de subrayado que ponen otros elementos, como ocurre en los links.

`text-transform:capitalize;` Curioso efecto que provee de cuatro posibilidades: capitalize que convierte en mayúscula la primera letra de cada palabra del texto, como puedes ver en el ejemplo, donde el texto original no es así. uppercase para convertir todas las letras a mayúsculas y lowercase para lo contrario, es decir, convertir todas las letras a minúsculas. El valor por defecto es none, que como ya habrás supuesto, deja el texto tal como está escrito.

`text-align:left;` Alinea el texto según convenga. Con left a la izquierda (por defecto), right a la derecha, center centrado, y justify justificado, es decir, igualando todas las líneas en longitud a izquierda o derecha.

`text-indent:30px` Produce que la primera línea del párrafo se escriba adentrada (indentada) un cierto espacio hacia la derecha o hacia la izquierda, dependiendo de la alineación activa.

Este sería un resumen sobre la declaración font:

Y este sobre el resto:

Tenemos también otras instrucciones que pueden modificar el aspecto de determinadas zonas en lugar de actuar sobre todo el bloque. Son los pseudo-elementos. Por ejemplo:

```
p :first-letter{
  color:red;
}
p :first-line{
  color:green;
}
p :after{
  content:"Hola";
}
p :before{
```



```

    content:"Por ejemplo:";
}
Como puedes ver, es fácilmente deducible lo que hace cada uno:
:first-letter actuará sobre la primera letra del párrafo.
:first-line lo hará sobre la primera línea.
:after content:"Hola"; actuará sobre el texto que haya después de la palabra "Hola" y
:before content:"Por ejemplo:"; antes de la frase "Por ejemplo:"

```

Bordes de un bloque

Con css se pueden definir los bordes de un bloque, que por defecto son invisibles. Por tanto, lo primero que hay que definir es el estilo del borde. Además del estilo, se puede definir su color y grosor, y todo ello puede hacerse globalmente, sobre los cuatro lados del bloque, o cada uno por separado. Las unidades de medida y los nombres o códigos de colores que pueden utilizarse, son los mismos que ya se han visto anteriormente para los textos. Para definir el grosor de los bordes tenemos:

- border-left-width:unidad borde izquierdo
- border-right-width:unidad borde derecho
- border-top-width:unidad borde superior
- border-bottom-width:unidad borde inferior
- border-width:unidad los cuatros bordes

Donde unidad es una de las ya conocidas, por ejemplo 1px. También pueden utilizarse constantes tales como thin para fino (2px), medium para medio (4px), y thick para grueso (6px).

Ningun estilo de bordes (excepto solid) se lleva bien con el parámetro de grosor, que implícitamente establece su dimensionado más conveniente. Tenemos:

- border-left-style:estilo borde izquierdo
- border-right-style:estilo borde derecho
- border-top-style:estilo borde superior
- border-bottom-style:estilo borde inferior
- border-style:estilo los cuatros bordes

Donde estilo puede ser:

- none (por defecto)
- solid
- double
- ridge
- groove
- inset
- outset
- dotted
- dashed

Por último, los parámetros de color de los bordes:

- border-left-color:color borde izquierdo
- border-right-color:color borde derecho
- border-top-color:color borde superior
- border-bottom-color:color borde inferior
- border-color:color los cuatros bordes

Donde color puede ser un nombre de color, en inglés, o su código RGB en hexadecimal, como ya sabes.

Las combinaciones de todos estos parámetros son infinitas, y lo mejor es ir haciendo pruebas. Recuerda que cada navegador interpreta todo esto a su manera, y algunos no lo interpretan en absoluto. He aquí aplicado al ejemplo anterior sobre texto:

Si escribimos la siguiente hoja de estilo estilo.css:

```
BODY {color:green }

P {color:red;
  font-size:20px;
  font-family:Courier;
  font-weight:bold;
  font-style:italic;
  line-height:30px;
  letter-spacing:5px;
  text-decoration:underline;
  text-transform:capitalize;
  text-align:left;
  text-indent:30px;

  border-width:2px;
  border-color:blue;
  border-style:solid;
}
```

Medios CSS.

Se llaman medios a las distintas tecnologías que hay para presentar un documento de html. El más común es la pantalla de ordenador, pero el documento también puede imprimirse, verse en el móvil, abrirse con un dispositivo braille, con un sintetizador de voz. etc.

El lenguaje CSS permite crear distintas hojas de estilo para un mismo documento, de forma que puede aplicarse cada una de ellas a un medio distinto, así de una misma página puede haber una versión para pantalla, otra para imprimir, otra para móviles, otra para lenguaje braille, etc.

Definir el documento

Si queremos que el documento no se vea igual en un medio que en otro, debemos crear diferentes hojas de estilo para cada medio. Estas pueden estar enlazadas en el documento principal de dos formas:

mediante el atributo "media" en la etiqueta link que enlaza la hoja de estilos. En este ejemplo se enlaza a una hoja de estilos que indica una versión para imprimir:

```
<link rel="stylesheet" type="text/css" href="imprimir.css" media="print" />
```

mediante la regla `@ import`. Enlazamos también a una hoja de estilos externa de la siguiente manera:

```
@import url("estilo.css") print;
```

mediante la regla `@media`, insertando el código en la misma página.

La etiqueta `link` es la forma clásica de enlazar archivos externos. Mediante el atributo `media` indicamos el tipo de medio al que va dirigido. La página se abrirá con ese estilo sólo en los medios que le indiquemos. Podemos indicar más de uno, siempre que los escribamos separados por comas:

La regla `@import`

La regla `@import` es otra forma de enlazar un archivo externo. observese que tiene que estar dentro del código CSS, con lo que se enlaza una hoja de estilos con otra hoja de estilos.

Cualquier regla `@import` debe preceder a todas las reglas especificadas en una hoja de estilo. por tanto estas reglas siempre debe estar en primer lugar de la hoja, solo puede ser precedida por otra regla `@import`.

Observese que tampoco tiene llaves. Escribimos `@import` e inmediatamente después la dirección del archivo que queremos aplicar, mediante la formula: `url("ruta_archivo.css")`, aunque también podemos escribir directamente la ruta entre comillas. Después escribiremos el tipo de medio al que se aplica la hoja. Podemos escribir la regla `@import` de cualquiera de estas dos formas:

```
@import "ruta_fichero.css" print;
```

```
@import url("ruta_fichero.css") print;
```

Es importante escribir la regla en una sola línea y acabarla siempre con el punto y coma (;), para indicar claramente donde empieza y acaba cada regla.

para especificar más de un tipo de medio para una hoja, los pondremos seguidos y separados por comas:

```
@import url("ruta_fichero.css") print, braille, embossed ;
```

Si no especificamos ningún medio se entiende que la hoja sirve para todos los medios.

La regla `@media`

La regla `@media` permite insertar el código CSS en la misma página que estamos. su sintaxis es la siguiente:

```
@media tipo_de_medio { -Código CSS para este medio- }
```

Donde pone "tipo de medio se especifica el tipo de medio, si hay más de uno pueden ponerse separados por comas. Entre llaves pondremos todo el código CSS que aplicamos a ese tipo de medio. Ejemplo:

```
@media tv, projection {  
  body: { background-color: lime; }  
  h1 { font-size: 2em; color: purple; text-align: center; }  
  h2 { font-size: 1.5em; color: blue; text-align: center; }  
  p { font-size: 1em; color: olive; text-align: justify; }  
}
```

Tipos de medios

Los tipos de medios indican los dispositivos para mostrar la página. En la hoja de estilos podemos escribir los siguientes:

`all` : Para todos los dispositivos. es el valor por defecto.

`braille` : Para los dispositivos táctiles de braille.

`embossed` : Para las impresoras de páginas braille.

`handheld` : Para teléfonos móviles y dispositivos de mano (pantalla pequeña y ancho de banda limitada).

`print` : Para impresoras y documentos vistos en pantalla en modo "vista previa a impresión".

`projection` : Para proyectores y presentaciones proyectadas.

screen : Para las pantallas de ordenadores en color. (PC, Mac, ...)

speech : Para los sintetizadores de voz.

tty : Para medios que usan una rejilla de caracteres de espacio fijo (Ej: teletipos, terminales, dispositivos portátiles con monitor de baja resolución, etc.).

tv : Para televisores y dispositivos similares. (resolución baja, con color y sonido).

Estilo para imprimir

Lo normal al crear una página es definir un estilo general para todos los medios, y algunos más específicos para otros medios. Cualquier página que quiera tener un poco de calidad debe ofrecer una versión para imprimir al usuario, por lo que he elegido este medio para ver cómo puede diferenciarse el estilo de un medio a otro.

En las versiones para imprimir se suelen eliminar contenidos superfluos, se modifican o eliminan algunas imágenes y los colores de fondo y se modifican los contenidos de texto para facilitar su lectura. veamos cómo se hace esto.

ocultar contenidos superfluos

Hay partes de la página que no tiene sentido imprimirlas, tales como menús de navegación, imágenes de fondo, pie de página, partes de la cabecera, etc. Ocultar estas partes que no se van a imprimir es tan fácil como aplicarles la propiedad display: none. Ejemplo:

```
#cabecera, #menu, #lateral, #pie {display: none; }
```

corregir la estructura de la página

Al quitar algunos bloques, el cuerpo principal puede quedar desajustado respecto de la anchura de la página. Debemos pues ajustar el ancho del contenido a imprimir al ancho de la página. colocar o corregir los márgenes si es necesario, y reajustar el contenido de los elementos que se van a imprimir.

modificar el tipo y colores de letra

Debemos asegurarnos de que el tipo de letra es fácilmente legible, si hace falta lo cambiamos, y es conveniente que el color del texto sea en negro, ya que aunque la mayoría de impresoras admiten los colores, con páginas con el texto en negro, permitimos al usuario ahorrar costes.

Imprimir los enlaces

En un medio impreso no tiene sentido poner enlaces, pero sí que podemos indicar la dirección del enlace, sobre todo cuando es una página externa. para ello podemos usar la propiedad content vista en la página anterior de la siguiente forma:

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

En este ejemplo se imprime la ruta del enlace (el valor del atributo href) después de su referencia. Si queremos que se imprima antes usaremos el seudoelemento :before, pudiendo también incluir algún texto (tal como se ve en el uso de la propiedad content en el apartado anterior).

El inconveniente de usar el código anterior es que se imprimen también los enlaces internos, lo cual no tiene sentido dar la referencia. para ello la solución consiste en dar a los enlaces externos en el código html un atributo class y hacer referencia a ellos en el selector de la regla CSS. Ejemplo:

todos los enlaces externos deberán llevar el atributo:

```
<a class="externo" href="http: ..." > ...</a>
```

Y la regla CSS a aplicar será:

```
.externo:after {content: attr(href); }
```

Debemos comprobar cómo quedará nuestra página impresa, a cada paso que demos, para darle el aspecto deseado. No hace falta imprimirla cada vez que hagamos un cambio; basta con mirar en el navegador en "archivo/vista preliminar" para ver como queda nuestra página impresa.

Comentarios CSS.

Bien, para comentar en CSS lo que debemos hacer es lo siguiente: Primero abriremos el comentario con una `» / »` seguido de un asterisco `*`. A partir de ahí podremos escribir el comentario. Para cerrar el mismo, debemos escribir los caracteres anteriores pero en orden inverso.

Sintaxis de la definición de las propiedades CSS

A lo largo de los próximos capítulos, se incluyen las definiciones formales de la mayoría de propiedades de CSS. La definición formal se basa en la información recogida en el estándar oficial y se muestra en forma de tabla.

Una de las principales informaciones de cada definición es la lista de posibles valores que admite la propiedad. Para definir la lista de valores permitidos se sigue un formato que es necesario detallar.

Si el valor permitido se indica como una sucesión de palabras sin ningún carácter que las separe (paréntesis, comas, barras, etc.) el valor de la propiedad se debe indicar tal y como se muestra y con esas palabras en el mismo orden.

Si el valor permitido se indica como una sucesión de valores separados por una barra simple (carácter `|`) el valor de la propiedad debe tomar uno y sólo uno de los valores indicados. Por ejemplo, la notación `<porcentaje> | <medida> | inherit` indica que la propiedad solamente puede tomar como valor la palabra reservada `inherit` o un porcentaje o una medida.

Si el valor permitido se indica como una sucesión de valores separados por una barra doble (símbolo `||`) el valor de la propiedad puede tomar uno o más valores de los indicados y en cualquier orden.

Por ejemplo, la notación `<color> || <estilo> || <medida>` indica que la propiedad puede tomar como valor cualquier combinación de los valores indicados y en cualquier orden. Se podría establecer un color y un estilo, solamente una medida o una medida y un estilo. Además, el orden en el que se indican los valores es indiferente. Opcionalmente, se pueden utilizar paréntesis para agrupar diferentes valores.

Por último, en cada valor o agrupación de valores se puede indicar el tipo de valor: opcional, obligatorio, múltiple o restringido.

El carácter `*` indica que el valor ocurre cero o más veces; el carácter `+` indica que el valor ocurre una o más veces; el carácter `?` indica que el valor es opcional y por último, el carácter `{número_1, número_2}` indica que el valor ocurre al menos tantas veces como el valor indicado en `número_1` y como máximo tantas veces como el valor indicado en `número_2`.

Por ejemplo, el valor `[<family-name> ,]*` indica que el valor de tipo `<family_name>` seguido por una coma se puede incluir cero o más veces. El valor `<url>? <color>` significa que la URL es opcional y el color obligatorio y en el orden indicado. Por último, el valor `[<medida> | thick | thin] {1,4}` indica que se pueden escribir entre 1 y 4 veces un valor que sea o una medida o la palabra `thick` o la palabra `thin`.

No obstante, la mejor forma de entender la notación formal para las propiedades de CSS es observar la definición de cada propiedad y volver a esta sección siempre que sea necesario.

Qué es un selector

Un selector CSS es la primera parte de una regla CSS. Es un patrón de elementos y otros En un artículo anterior explicamos términos que indican al navegador qué elementos HTML se seleccionan para aplicarles una regla que incluye los valores de las propiedades CSS. El elemento o los elementos seleccionados por el selector se denominan sujeto del selector.

```
h1 {  
  color: blue;  
  background-color: yellow;  
}  
  
p {  
  color: red;  
}
```

En artículos anteriores ya has visto algunos selectores y has aprendido que hay diversas maneras de organizar el documento. Por ejemplo, seleccionando un elemento, como h1, o seleccionando una clase, como .special.

En CSS, los selectores se definen en la especificación de selectores de CSS correspondiente; al igual que cualquier otro elemento de CSS, es necesario que los navegadores los admitan para que funcionen. La mayoría de los selectores que encontrarás se definen en especificación de nivel 3 de selectores, que es una especificación consolidada y, por lo tanto, la mayoría de navegadores admitirán esos selectores.

Listas de selectores

Si más de un elemento utiliza el mismo CSS, puedes combinar los selectores en una lista de selectores para que la regla se aplique a cada uno de los selectores individuales. Por ejemplo, si tengo el mismo CSS para un h1 y para una clase .special, los puedo escribir como dos reglas separadas.

Cuando agrupas los selectores de esta manera, si alguno de los selectores no es válido, el navegador sencillamente ignora toda la regla.

Tipos de selectores

Hay diferentes agrupaciones de selectores, y conocer qué tipo de selector necesitas te ayudará a encontrar la herramienta adecuada para tu trabajo. En estos subartículos vamos a ver los diferentes grupos de selectores con más detalle.

Selectores de tipo, de clase y de ID

Este grupo incluye selectores que delimitan un elemento HTML, como por ejemplo un <h1>.

```
h1 { }
```

También incluye selectores que delimitan una clase:

```
.box { }
```

o un ID:

```
#unique { }
```

Selectores de atributo

Este grupo de selectores te proporciona diferentes formas de seleccionar elementos según la presencia de un atributo determinado en un elemento:

```
a[title] { }
```

O incluso hacer una selección basada en la presencia de un atributo que tiene un valor particular asignado:

```
a[href="https://example.com"] { }
```

Las pseudoclases y los pseudoelementos

Este grupo de selectores incluye pseudoclases, que aplican estilo a ciertos estados de un elemento. La pseudoclase `:hover`, por ejemplo, selecciona un elemento solo cuando se le pasa el ratón por encima.

```
a: hover { }
```

Copy to Clipboard

También incluye pseudoelementos, que seleccionan una parte determinada de un elemento en vez del elemento en sí. Por ejemplo, `::first-line` siempre selecciona la primera línea del texto que se encuentra dentro de un elemento (`<p>`, en el ejemplo siguiente), y actúa como si un elemento `` hubiera delimitado la primera línea, seleccionado y aplicado estilo.

```
p::first-line { }
```

Combinadores

El último grupo de selectores combina otros selectores con el fin de delimitar elementos de nuestros documentos. El ejemplo siguiente selecciona los párrafos que son hijos directos del elemento `<article>` utilizando el operador de combinación hijo (`>`):

```
article > p { }
```

Echa un vistazo a esta tabla de referencia de selectores que contiene enlaces directos a los distintos tipos de selectores que se explican en nuestra sección de aprendizaje o en MDN, o bien sigue adelante e inicia tu viaje para averiguar cómo funcionan en Selectores de tipo, de clase y de ID.

- [Anterior](#)
- [Overview: Building blocks](#)
- [Siguiente](#)

Tabla de referencia de selectores

La tabla que te mostramos a continuación proporciona una descripción general de los selectores que puedes utilizar junto con enlaces a diversas páginas de esta guía que te mostrarán cómo utilizar cada tipo de selector. También hemos incluido un enlace a la página MDN de cada selector para poder comprobar la información sobre los navegadores que lo admiten. Puedes utilizarlo como referencia

para volver cuando necesites buscar los selectores a medida que avanzas con la materia o mientras experimentas con CSS por tu cuenta.

Selectores Básicos

a) Selector universal

Se utiliza para seleccionar todos los elementos de la página. El siguiente ejemplo elimina el margen y el relleno de todos los elementos HTML (por ahora no es importante fijarse en la parte de la declaración de la regla CSS):

```
* {  
  margin: 0;  
  padding: 0;  
}
```

El selector universal se indica mediante un asterisco (*). A pesar de su sencillez, no se utiliza habitualmente, ya que es difícil que un mismo estilo se pueda aplicar a todos los elementos de una página.

No obstante, sí que se suele combinar con otros selectores y además, forma parte de algunos hacks muy utilizados, como se verá más adelante.

b) Selector de tipo o etiqueta

Selecciona todos los elementos de la página cuya etiqueta HTML coincide con el valor del selector. El siguiente ejemplo selecciona todos los párrafos de la página:

```
p {  
  ...  
}
```

Para utilizar este selector, solamente es necesario indicar el nombre de una etiqueta HTML (sin los caracteres < y >) correspondiente a los elementos que se quieren seleccionar.

El siguiente ejemplo aplica diferentes estilos a los titulares y a los párrafos de una página HTML:

```
h1 {  
  color: red;  
}
```

```
h2 {  
  color: blue;  
}
```

```
p {  
  color: black;  
}
```

Si se quiere aplicar los mismos estilos a dos etiquetas diferentes, se pueden encadenar los selectores.

En el siguiente ejemplo, los títulos de sección h1, h2 y h3 comparten los mismos estilos:

```
h1 {  
  color: #8A8E27;  
  font-weight: normal;  
  font-family: Arial, Helvetica, sans-serif;  
}  
h2 {  
  color: #8A8E27;  
  font-weight: normal;
```



```
font-family: Arial, Helvetica, sans-serif;
}
h3 {
color: #8A8E27;
font-weight: normal;
font-family: Arial, Helvetica, sans-serif;
}
```

En este caso, CSS permite agrupar todas las reglas individuales en una sola regla con un selector múltiple. Para ello, se incluyen todos los selectores separados por una coma (,) y el resultado es que la siguiente regla CSS es equivalente a las tres reglas anteriores:

```
h1, h2, h3 {
color: #8A8E27;
font-weight: normal;
font-family: Arial, Helvetica, sans-serif;
}
```

En las hojas de estilo complejas, es habitual agrupar las propiedades comunes de varios elementos en una única regla CSS y posteriormente definir las propiedades específicas de esos mismos elementos. El siguiente ejemplo establece en primer lugar las propiedades comunes de los títulos de sección (color y tipo de letra) y a continuación, establece el tamaño de letra de cada uno de ellos:

```
h1, h2, h3 {
color: #8A8E27;
font-weight: normal;
font-family: Arial, Helvetica, sans-serif;
}
h1 { font-size: 2em; }
h2 { font-size: 1.5em; }
h3 { font-size: 1.2em; }
```

c) Selector descendente

Selecciona los elementos que se encuentran dentro de otros elementos. Un elemento es descendiente de otro cuando se encuentra entre las etiquetas de apertura y de cierre del otro elemento.

El selector del siguiente ejemplo selecciona todos los elementos `` de la página que se encuentren dentro de un elemento `<p>`:

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

Si el código HTML de la página es el siguiente:

```
<p>
...
<span>texto1</span>
...
<a href="">...<span>texto2</span></a>
...
</p>
```

El selector `p span` selecciona tanto `texto1` como `texto2`. El motivo es que en el selector descendente, un elemento no tiene que ser descendiente directo del otro. La única condición es que un elemento debe estar dentro de otro elemento, sin importar el nivel de profundidad en el que se encuentre.

Al resto de elementos `` de la página que no están dentro de un elemento `<p>`, no se les aplica la regla CSS anterior.

Los selectores descendentes permiten aumentar la precisión del selector de tipo o etiqueta. Así, utilizando el selector descendente es posible aplicar diferentes estilos a los elementos del mismo tipo.

El siguiente ejemplo amplía el anterior y muestra de color azul todo el texto de los `` contenidos dentro de un `<h1>`:

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

```
h1 span { color: blue; }
```

Con las reglas CSS anteriores:

Los elementos `` que se encuentran dentro de un elemento `<p>` se muestran de color rojo.

Los elementos `` que se encuentran dentro de un elemento `<h1>` se muestran de color azul.

El resto de elementos `` de la página, se muestran con el color por defecto aplicado por el navegador.

La sintaxis formal del selector descendente se muestra a continuación:

```
selector1 selector2 selector3 ... selectorN
```

Los selectores descendentes siempre están formados por dos o más selectores separados entre sí por espacios en blanco. El último selector indica el elemento sobre el que se aplican los estilos y todos los selectores anteriores indican el lugar en el que se debe encontrar ese elemento.

En el siguiente ejemplo, el selector descendente se compone de cuatro selectores:

```
p a span em { text-decoration: underline; }
```

Los estilos de la regla anterior se aplican a los elementos de tipo `` que se encuentren dentro de elementos de tipo ``, que a su vez se encuentren dentro de elementos de tipo `<a>` que se encuentren dentro de elementos de tipo `<p>`.

No debe confundirse el selector descendente con la combinación de selectores:

```
/* El estilo se aplica a todos los elementos "p", "a", "span" y "em" */
```

```
p, a, span, em { text-decoration: underline; }
```

```
/* El estilo se aplica solo a los elementos "em" que se encuentran dentro de "p a span" */
```

```
p a span em { text-decoration: underline; }
```

Se puede restringir el alcance del selector descendente combinándolo con el selector universal. El siguiente ejemplo, muestra los dos enlaces de color rojo:

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

```
<p><a href="#">Enlace</a></p>
```

```
<p><span><a href="#">Enlace</a></span></p>
```

Sin embargo, en el siguiente ejemplo solamente el segundo enlace se muestra de color rojo:

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

```
<p><a href="#">Enlace</a></p>
```

```
<p><span><a href="#">Enlace</a></span></p>
```

La razón es que el selector `p * a` se interpreta como todos los elementos de tipo `<a>` que se encuentren dentro de cualquier elemento que, a su vez, se encuentre dentro de un elemento de tipo `<p>`. Como el primer elemento `<a>` se encuentra directamente bajo un elemento `<p>`, no se cumple la condición del selector `p * a`.

d) Selector de clase

Si se considera el siguiente código HTML de ejemplo:

```
<body>
```

```
<p>Lorem ipsum dolor sit amet...</p>
```

```
<p>Nunc sed lacus et est adipiscing accumsan...</p>
```

```
<p>Class aptent taciti sociosqu ad litora...</p>
```

```
</body>
```

¿Cómo se pueden aplicar estilos CSS sólo al primer párrafo? El selector universal (`*`) no se puede utilizar porque selecciona todos los elementos de la página. El selector de tipo o etiqueta (`p`) tampoco se puede utilizar porque seleccionaría todos los párrafos. Por último, el selector descendente (`body p`) tampoco se puede utilizar porque todos los párrafos se encuentran en el mismo sitio.

Una de las soluciones más sencillas para aplicar estilos a un solo elemento de la página consiste en utilizar el atributo class de HTML sobre ese elemento para indicar directamente la regla CSS que se le debe aplicar:

```
<body>
  <p class="destacado">Lorem ipsum dolor sit amet...</p>
  <p>Nunc sed lacus et est adipiscing accumsan...</p>
  <p>Class aptent taciti sociosqu ad litora...</p>
</body>
```

A continuación, se crea en el archivo CSS una nueva regla llamada destacado con todos los estilos que se van a aplicar al elemento. Para que el navegador no confunda este selector con los otros tipos de selectores, se prefija el valor del atributo class con un punto (.) tal y como muestra el siguiente ejemplo:

```
.destacado { color: red; }
```

El selector .destacado se interpreta como "cualquier elemento de la página cuyo atributo class sea igual a destacado", por lo que solamente el primer párrafo cumple esa condición.

Este tipo de selectores se llaman selectores de clase y son los más utilizados junto con los selectores de ID que se verán a continuación. La principal característica de este selector es que en una misma página HTML varios elementos diferentes pueden utilizar el mismo valor en el atributo class:

```
<body>
  <p class="destacado">Lorem ipsum dolor sit amet...</p>
  <p>Nunc sed lacus et <a href="#" class="destacado">est adipiscing</a> accumsan...</p>
  <p>Class aptent taciti <em class="destacado">sociosqu ad</em> litora...</p>
</body>
```

Los selectores de clase son imprescindibles para diseñar páginas web complejas, ya que permiten disponer de una precisión total al seleccionar los elementos. Además, estos selectores permiten reutilizar los mismos estilos para varios elementos diferentes.

A continuación se muestra otro ejemplo de selectores de clase:

```
.aviso {
  padding: 0.5em;
  border: 1px solid #98be10;
  background: #ff6feda;
}
.error {
  color: #930;
  font-weight: bold;
}
<span class="error">...</span>
<div class="aviso">...</div>
```

El elemento tiene un atributo class="error", por lo que se le aplican las reglas CSS indicadas por el selector .error. Por su parte, el elemento <div> tiene un atributo class="aviso", por lo que su estilo es el que definen las reglas CSS del selector .aviso.

En ocasiones, es necesario restringir el alcance del selector de clase. Si se considera de nuevo el ejemplo anterior:

```
<body>
  <p class="destacado">Lorem ipsum dolor sit amet...</p>
  <p>Nunc sed lacus et <a href="#" class="destacado">est adipiscing</a> accumsan...</p>
  <p>Class aptent taciti <em class="destacado">sociosqu ad</em> litora...</p>
</body>
```

¿Cómo es posible aplicar estilos solamente al párrafo cuyo atributo class sea igual a destacado? Combinando el selector de tipo y el selector de clase, se obtiene un selector mucho más específico:

```
p.destacado { color: red }
```

El selector `p.destacado` se interpreta como "aquellos elementos de tipo `<p>` que dispongan de un atributo `class` con valor `destacado`". De la misma forma, el selector `a.destacado` solamente selecciona los enlaces cuyo atributo `class` sea igual a `destacado`.

De lo anterior se deduce que el atributo `.destacado` es equivalente a `*.destacado`, por lo que todos los diseñadores obvian el símbolo `*` al escribir un selector de clase normal.

No debe confundirse el selector de clase con los selectores anteriores:

```
/* Todos los elementos de tipo "p" con atributo class="aviso" */
```

```
p.aviso { ... }
```

```
/* Todos los elementos con atributo class="aviso" que estén dentro  
de cualquier elemento de tipo "p" */
```

```
p .aviso { ... }
```

```
/* Todos los elementos "p" de la página y todos los elementos con  
atributo class="aviso" de la página */
```

```
p, .aviso { ... }
```

Por último, es posible aplicar los estilos de varias clases CSS sobre un mismo elemento. La sintaxis es similar, pero los diferentes valores del atributo `class` se separan con espacios en blanco. En el siguiente ejemplo:

```
<p class="especial destacado error">Párrafo de texto...</p>
```

Al párrafo anterior se le aplican los estilos definidos en las reglas `.especial`, `.destacado` y `.error`, por lo que en el siguiente ejemplo, el texto del párrafo se vería de color rojo, en negrita y con un tamaño de letra de 15 píxel:

```
.error { color: red; }
```

```
.destacado { font-size: 15px; }
```

```
.especial { font-weight: bold; }
```

```
<p class="especial destacado error">Párrafo de texto...</p>
```

Si un elemento dispone de un atributo `class` con más de un valor, es posible utilizar un selector más avanzado:

```
.error { color: red; }
```

```
.error.destacado { color: blue; }
```

```
.destacado { font-size: 15px; }
```

```
.especial { font-weight: bold; }
```

```
<p class="especial destacado error">Párrafo de texto...</p>
```

En el ejemplo anterior, el color de la letra del texto es azul y no rojo. El motivo es que se ha utilizado un selector de clase múltiple `.error.destacado`, que se interpreta como "aquellos elementos de la página que dispongan de un atributo `class` con al menos los valores `error` y `destacado`".

e) Selectores de ID

En ocasiones, es necesario aplicar estilos CSS a un único elemento de la página. Aunque puede utilizarse un selector de clase para aplicar estilos a un único elemento, existe otro selector más eficiente en este caso.

El selector de ID permite seleccionar un elemento de la página a través del valor de su atributo `id`. Este tipo de selectores sólo seleccionan un elemento de la página porque el valor del atributo `id` no se puede repetir en dos elementos diferentes de una misma página.

La sintaxis de los selectores de ID es muy parecida a la de los selectores de clase, salvo que se utiliza el símbolo de la almohadilla (`#`) en vez del punto (`.`) como prefijo del nombre de la regla CSS:

```
#destacado { color: red; }
```

```
<p>Primer párrafo</p>
```

```
<p id="destacado">Segundo párrafo</p>
```

<p>Tercer párrafo</p>

En el ejemplo anterior, el selector #destacado solamente selecciona el segundo párrafo (cuyo atributo id es igual a destacado).

La principal diferencia entre este tipo de selector y el selector de clase tiene que ver con HTML y no con CSS. Como se sabe, en una misma página, el valor del atributo id debe ser único, de forma que dos elementos diferentes no pueden tener el mismo valor de id. Sin embargo, el atributo class no es obligatorio que sea único, de forma que muchos elementos HTML diferentes pueden compartir el mismo valor para su atributo class.

De esta forma, la recomendación general es la de utilizar el selector de ID cuando se quiere aplicar un estilo a un solo elemento específico de la página y utilizar el selector de clase cuando se quiere aplicar un estilo a varios elementos diferentes de la página HTML.

Al igual que los selectores de clase, en este caso también se puede restringir el alcance del selector mediante la combinación con otros selectores. El siguiente ejemplo aplica la regla CSS solamente al elemento de tipo <p> que tenga un atributo id igual al indicado:

```
p#aviso { color: blue; }
```

A primera vista, restringir el alcance de un selector de ID puede parecer absurdo. En realidad, un selector de tipo p#aviso sólo tiene sentido cuando el archivo CSS se aplica sobre muchas páginas HTML diferentes.

En este caso, algunas páginas pueden disponer de elementos con un atributo id igual a aviso y que no sean párrafos, por lo que la regla anterior no se aplica sobre esos elementos.

No debe confundirse el selector de ID con los selectores anteriores:

```
/* Todos los elementos de tipo "p" con atributo id="aviso" */
```

```
p#aviso { ... }
```

```
/* Todos los elementos con atributo id="aviso" que estén dentro  
de cualquier elemento de tipo "p" */
```

```
p #aviso { ... }
```

```
/* Todos los elementos "p" de la página y todos los elementos con  
atributo id="aviso" de la página */
```

```
p, #aviso { ... }
```

2.1.6. Combinación de selectores básicos

CSS permite la combinación de uno o más tipos de selectores para restringir el alcance de las reglas CSS. A continuación se muestran algunos ejemplos habituales de combinación de selectores.

Selectores Avanzado

Utilizando solamente los selectores básicos de la sección anterior, es posible diseñar prácticamente cualquier página web. No obstante, CSS define otros selectores más avanzados que permiten simplificar las hojas de estilos.

Desafortunadamente, el navegador Internet Explorer 6 y sus versiones anteriores no soportaban este tipo de selectores avanzados, por lo que su uso no era común hasta hace poco tiempo. Si quieres consultar el soporte de los selectores en los distintos navegadores, puedes utilizar las siguientes referencias:

Utilizando solamente los selectores básicos de la sección anterior, es posible diseñar prácticamente cualquier página web. No obstante, CSS define otros selectores más avanzados que permiten simplificar las hojas de estilos.

Desafortunadamente, el navegador Internet Explorer 6 y sus versiones anteriores no soportaban este tipo de selectores avanzados, por lo que su uso no era común hasta hace poco tiempo. Si quieres

consultar el soporte de los selectores en los distintos navegadores, puedes utilizar las siguientes referencias:

Lista completa de los selectores que soporta cada versión de cada navegador

Test online en el que puedes comprobar los selectores que soporta el navegador con el que realizas el test

a) Selector de hijos

Se trata de un selector similar al selector descendente, pero muy diferente en su funcionamiento. Se utiliza para seleccionar un elemento que es hijo directo de otro elemento y se indica mediante el "signo de mayor que" (>):

```
p > span { color: blue; }
```

```
<p><span>Texto1</span></p>
```

```
<p><a href="#"><span>Texto2</span></a></p>
```

En el ejemplo anterior, el selector `p > span` se interpreta como "cualquier elemento `` que sea hijo directo de un elemento `<p>`", por lo que el primer elemento `` cumple la condición del selector. Sin embargo, el segundo elemento `` no la cumple porque es descendiente pero no es hijo directo de un elemento `<p>`.

El siguiente ejemplo muestra las diferencias entre el selector descendente y el selector de hijos:

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

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

```
<p><a href="#">Enlace1</a></p>
```

```
<p><span><a href="#">Enlace2</a></span></p>
```

El primer selector es de tipo descendente y por tanto se aplica a todos los elementos `<a>` que se encuentran dentro de elementos `<p>`. En este caso, los estilos de este selector se aplican a los dos enlaces.

Por otra parte, el selector de hijos obliga a que el elemento `<a>` sea hijo directo de un elemento `<p>`. Por lo tanto, los estilos del selector `p > a` no se aplican al segundo enlace del ejemplo anterior.

b) Selector adyacente

El selector adyacente se emplea para seleccionar elementos que en el código HTML de la página se encuentran justo a continuación de otros elementos. Su sintaxis emplea el signo `+` para separar los dos elementos:

```
elemento1 + elemento2 { ... }
```

Si se considera el siguiente código HTML:

```
<body>
```

```
<h1>Titulo1</h1>
```

```
<h2>Subtítulo</h2>
```

```
...
```

```
<h2>Otro subtítulo</h2>
```

```
...
```

```
</body>
```

La página anterior dispone de dos elementos `<h2>`, pero sólo uno de ellos se encuentra inmediatamente después del elemento `<h1>`. Si se quiere aplicar diferentes colores en función de esta circunstancia, el selector adyacente es el más adecuado:

```
h2 { color: green; }
```

```
h1 + h2 { color: red }
```

Las reglas CSS anteriores hacen que todos los `<h2>` de la página se vean de color verde, salvo aquellos `<h2>` que se encuentran inmediatamente después de cualquier elemento `<h1>` y que se muestran de color rojo.

Técnicamente, los elementos que forman el selector adyacente deben cumplir las dos siguientes condiciones:

elemento1 y elemento2 deben ser elementos hermanos, por lo que su elemento padre debe ser el mismo.

elemento2 debe aparecer inmediatamente después de elemento1 en el código HTML de la página.

El siguiente ejemplo es muy útil para los textos que se muestran como libros:

```
p + p { text-indent: 1.5em; }
```

En muchos libros, suele ser habitual que la primera línea de todos los párrafos esté indentada, salvo la primera línea del primer párrafo. Con el selector `p + p`, se seleccionan todos los párrafos de la página que estén precedidos por otro párrafo, por lo que no se aplica al primer párrafo de la página.

c) Selector de atributos

El último tipo de selectores avanzados lo forman los selectores de atributos, que permiten seleccionar elementos HTML en función de sus atributos y/o valores de esos atributos.

Los cuatro tipos de selectores de atributos son:

`[nombre_atributo]`, selecciona los elementos que tienen establecido el atributo llamado `nombre_atributo`, independientemente de su valor.

`[nombre_atributo=valor]`, selecciona los elementos que tienen establecido un atributo llamado `nombre_atributo` con un valor igual a `valor`.

`[nombre_atributo~=valor]`, selecciona los elementos que tienen establecido un atributo llamado `nombre_atributo` y al menos uno de los valores del atributo es `valor`.

`[nombre_atributo|=valor]`, selecciona los elementos que tienen establecido un atributo llamado `nombre_atributo` y cuyo valor es una serie de palabras separadas con guiones, pero que comienza con `valor`. Este tipo de selector sólo es útil para los atributos de tipo `lang` que indican el idioma del contenido del elemento.

A continuación se muestran algunos ejemplos de estos tipos de selectores:

```
/* Se muestran de color azul todos los enlaces que tengan  
un atributo "class", independientemente de su valor */
```

```
a[class] { color: blue; }
```

```
/* Se muestran de color azul todos los enlaces que tengan
```



```

    un atributo "class" con el valor "externo" */
a[class="externo"] { color: blue; }
/* Se muestran de color azul todos los enlaces que apunten
    al sitio "http://www.ejemplo.com" */
a[href="http://www.ejemplo.com"] { color: blue; }
/* Se muestran de color azul todos los enlaces que tengan
    un atributo "class" en el que al menos uno de sus valores
    sea "externo" */
a[class~="externo"] { color: blue; }
/* Selecciona todos los elementos de la página cuyo atributo
    "lang" sea igual a "en", es decir, todos los elementos en inglés */
*[lang=en] { ... }
/* Selecciona todos los elementos de la página cuyo atributo
    "lang" empiece por "es", es decir, "es", "es-ES", "es-AR", etc. */
*[lang|= "es"] { color : red }

```

Agrupación de Reglas.

Cuando se crean archivos CSS complejos con decenas o cientos de reglas, es habitual que los estilos que se aplican a un mismo selector se definan en diferentes reglas:

```

h1 { color: red; }
...
h1 { font-size: 2em; }
...
h1 { font-family: Verdana; }

```

Las tres reglas anteriores establecen el valor de tres propiedades diferentes de los elementos <h1>. Antes de que el navegador muestre la página, procesa todas las reglas CSS de la página para tener en cuenta todos los estilos definidos para cada elemento.

Cuando el selector de dos o más reglas CSS es idéntico, se pueden agrupar las declaraciones de las reglas para hacer las hojas de estilos más eficientes:

```

h1 {
    color: red;
    font-size: 2em;
    font-family: Verdana;
}

```

El ejemplo anterior tiene el mismo efecto que las tres reglas anteriores, pero es más eficiente y es más fácil de modificar y mantener por parte de los diseñadores. Como CSS ignora los espacios en blanco y las nuevas líneas, también se pueden agrupar las reglas de la siguiente forma:

```

h1 { color: red; font-size: 2em; font-family: Verdana; }

```

Si se quiere reducir al máximo el tamaño del archivo CSS para mejorar ligeramente el tiempo de carga de la página web, también es posible indicar la regla anterior de la siguiente forma:

```

h1 {color:red;font-size:2em;font-family:Verdana;}

```

Herencias

La herencia en CSS es el mecanismo mediante el cual determinadas propiedades de un elemento padre se transmiten a sus hijos. ... Por ejemplo, los márgenes no se heredan porque es poco probable que un elemento hijo necesite los mismos márgenes que su padre.

Colisiones de estilos.

Es posible que tengamos a veces varias reglas que apliquen el mismo estilo a un determinado elemento . Determinar todas las declaraciones que se aplican al elemento para el medio CSS seleccionado. ... Ordenar las declaraciones según su origen y su prioridad (palabra clave !

Unidades de Medidas

Las medidas en CSS se emplean, entre otras, para definir la altura, anchura y márgenes de los elementos y para establecer el tamaño de letra del texto. Todas las medidas se indican como un valor numérico entero o decimal seguido de una unidad de medida (sin ningún espacio en blanco entre el número y la unidad de medida).

CSS divide las unidades de medida en dos grupos: absolutas y relativas. Las medidas relativas definen su valor en relación con otra medida, por lo que para obtener su valor real, se debe realizar alguna operación con el valor indicado. Las unidades absolutas establecen de forma completa el valor de una medida, por lo que su valor real es directamente el valor indicado.

Si el valor es 0, la unidad de medida es opcional. Si el valor es distinto a 0 y no se indica ninguna unidad, la medida se ignora completamente, lo que suele ser uno de los errores más habituales de los diseñadores que empiezan con CSS. Algunas propiedades permiten indicar medidas negativas, aunque habitualmente sus valores son positivos. Si el valor decimal de una medida es inferior a 1, se puede omitir el 0 de la izquierda (0.5em es equivalente a .5em).

a) Unidades absolutas

Una medida indicada mediante unidades absolutas está completamente definida, ya que su valor no depende de otro valor de referencia. A continuación se muestra la lista completa de unidades absolutas definidas por CSS y su significado:

in, pulgadas ("inches", en inglés). Una pulgada equivale a 2.54 centímetros.

cm, centímetros.

mm, milímetros.

pt, puntos. Un punto equivale a 1 pulgada/72, es decir, unos 0.35 milímetros.

pc, picas. Una pica equivale a 12 puntos, es decir, unos 4.23 milímetros.

A continuación se muestran ejemplos de utilización de unidades absolutas:

```
/* El cuerpo de la página debe mostrar un margen de media pulgada */  
body { margin: 0.5in; }
```

```
/* Los elementos <h1> deben mostrar un interlineado de 2 centímetros */
```

```
h1 { line-height: 2cm; }
```

```
/* Las palabras de todos los párrafos deben estar separadas 4 milímetros entre si */
```

```
p { word-spacing: 4mm; }
```

```
/* Los enlaces se deben mostrar con un tamaño de letra de 12 puntos */
```

```
a { font-size: 12pt }
```

```
/* Los elementos <span> deben tener un tamaño de letra de 1 pica */
```

`span { font-size: 1pc }`

La principal ventaja de las unidades absolutas es que su valor es directamente el valor que se debe utilizar, sin necesidad de realizar cálculos intermedios. Su principal desventaja es que son muy poco flexibles y no se adaptan fácilmente a los diferentes medios.

De todas las unidades absolutas, la única que suele utilizarse es el punto (pt). Se trata de la unidad de medida preferida para establecer el tamaño del texto en los documentos que se van a imprimir, es decir, para el medio print de CSS, tal y como se verá más adelante.

b) Unidades relativas

Las unidades relativas, a diferencia de las absolutas, no están completamente definidas, ya que su valor siempre está referenciado respecto a otro valor. A pesar de su aparente dificultad, son las más utilizadas en el diseño web por la flexibilidad con la que se adaptan a los diferentes medios.

A continuación se muestran las tres unidades de medida relativas definidas por CSS y la referencia que toma cada una para determinar su valor real:

em, (no confundir con la etiqueta `` de HTML) relativa respecto del tamaño de letra del elemento.

ex, relativa respecto de la altura de la letra x ("equis minúscula") del tipo y tamaño de letra del elemento.

px, (píxel) relativa respecto de la resolución de la pantalla del dispositivo en el que se visualiza la página HTML.

Las unidades em y ex no han sido creadas por CSS, sino que llevan décadas utilizándose en el campo de la tipografía. Aunque no es una definición exacta, la unidad 1em equivale a la anchura de la letra M ("eme mayúscula") del tipo y tamaño de letra del elemento.

La unidad em hace referencia al tamaño en puntos de la letra que se está utilizando. Si se utiliza una tipografía de 12 puntos, 1em equivale a 12 puntos. El valor de 1ex se puede aproximar por 0.5 em.

Si se considera el siguiente ejemplo:

```
p { margin: 1em; }
```

La regla CSS anterior indica que los párrafos deben mostrar un margen de anchura igual a 1em. Como se trata de una unidad de medida relativa, es necesario realizar un cálculo matemático para determinar la anchura real de ese margen.

La unidad de medida em siempre hace referencia al tamaño de letra del elemento. Por otra parte, todos los navegadores muestran por defecto el texto de los párrafos con un tamaño de letra de 16 píxel. Por tanto, en este caso el margen de 1em equivale a un margen de anchura 16px.

A continuación se modifica el ejemplo anterior para cambiar el tamaño de letra de los párrafos:

```
p { font-size: 32px; margin: 1em; }
```

El valor del margen sigue siendo el mismo en unidades relativas (1em) pero su valor real ha variado porque el tamaño de letra de los párrafos ha variado. En este caso, el margen tendrá una anchura de 32px, ya que 1em siempre equivale al tamaño de letra del elemento.

Si se quiere reducir la anchura del margen a 16px pero manteniendo el tamaño de letra de los párrafos en 32px, se debe utilizar la siguiente regla CSS:

```
p { font-size: 32px; margin: 0.5em; }
```

El valor 0.5em se interpreta como "la mitad del tamaño de letra del elemento", ya que se debe multiplicar por 0.5 su tamaño de letra ($32\text{px} \times 0.5 = 16\text{px}$). De la misma forma, si se quiere mostrar un margen de 8px de anchura, se debería utilizar el valor 0.25em, ya que $32\text{px} \times 0.25 = 8\text{px}$.

La gran ventaja de las unidades relativas es que siempre mantienen las proporciones del diseño de la página. Establecer el margen de un elemento con el valor 1em equivale a indicar que "el margen del elemento debe ser del mismo tamaño que su letra y debe cambiar proporcionalmente".

En efecto, si el tamaño de letra de un elemento aumenta hasta un valor enorme, su margen de 1em también será enorme. Si su tamaño de letra se reduce hasta un valor diminuto, el margen de 1em también será diminuto. El uso de unidades relativas permite mantener las proporciones del diseño cuando se modifica el tamaño de letra de la página.

El funcionamiento de la unidad ex es idéntico a em, salvo que en este caso, la referencia es la altura de la letra x minúscula, por lo que su valor es aproximadamente la mitad que el de la unidad em.

Por último, las medidas indicadas en píxel también se consideran relativas, ya que el aspecto de los elementos dependerá de la resolución del dispositivo en el que se visualiza la página HTML. Si un elemento tiene una anchura de 400px, ocupará la mitad de una pantalla con una resolución de 800x600, pero ocupará menos de la tercera parte en una pantalla con resolución de 1440x900.

Las unidades de medida se pueden mezclar en los diferentes elementos de una misma página, como en el siguiente ejemplo:

```
body { font-size: 10px; }
```

```
h1 { font-size: 2.5em; }
```

En primer lugar, se establece un tamaño de letra base de 10 píxel para toda la página. A continuación, se asigna un tamaño de 2.5em al elemento <h1>, por lo que su tamaño de letra real será de $2.5 \times 10\text{px} = 25\text{px}$.

Como se vio en los capítulos anteriores, el valor de la mayoría de propiedades CSS se hereda de padres a hijos. Así por ejemplo, si se establece el tamaño de letra al elemento <body>, todos los elementos de la página tendrán el mismo tamaño de letra, salvo que indiquen otro valor.

Sin embargo, el valor de las medidas relativas no se hereda directamente, sino que se hereda su valor real una vez calculado. El siguiente ejemplo muestra este comportamiento:

```
body {  
  
  font-size: 12px;
```

```
text-indent: 3em;

}
```

```
h1 { font-size: 15px }
```

La propiedad `text-indent`, como se verá en los próximos capítulos, se utiliza para tabular la primera línea de un texto. El elemento `<body>` define un valor para esta propiedad, pero el elemento `<h1>` no lo hace, por lo que heredará el valor de su elemento padre. Sin embargo, el valor heredado no es 3em, sino 36px.

Si se heredara el valor 3em, al multiplicarlo por el valor de `font-size` del elemento `<h1>` (que vale 15px) el resultado sería $3\text{em} \times 15\text{px} = 45\text{px}$. No obstante, como se ha comentado, los valores que se heredan no son los relativos, sino los valores ya calculados.

Por lo tanto, en primer lugar se calcula el valor real de 3em para el elemento `<body>`: $3\text{em} \times 12\text{px} = 36\text{px}$. Una vez calculado el valor real, este es el valor que se hereda para el resto de elementos.

c) Porcentajes

El porcentaje también es una unidad de medida relativa, aunque por su importancia CSS la trata de forma separada a em, ex y px. Un porcentaje está formado por un valor numérico seguido del símbolo % y siempre está referenciado a otra medida. Cada una de las propiedades de CSS que permiten indicar como valor un porcentaje, define el valor al que hace referencia ese porcentaje.

Los porcentajes se pueden utilizar por ejemplo para establecer el valor del tamaño de letra de los elementos:

```
body { font-size: 1em; }
```

```
h1 { font-size: 200%; }
```

```
h2 { font-size: 150%; }
```

Los tamaños establecidos para los elementos `<h1>` y `<h2>` mediante las reglas anteriores, son equivalentes a 2em y 1.5em respectivamente, por lo que es más habitual definirlos mediante em.

Los porcentajes también se utilizan para establecer la anchura de los elementos:

```
div#contenido { width: 600px; }
```

```
div.principal { width: 80%; }
```

```
<div id="contenido">
```

```
  <div class="principal">
```

```
    ...
```

```
  </div>
```

</div>

En el ejemplo anterior, la referencia del valor 80% es la anchura de su elemento padre. Por tanto, el elemento <div> cuyo atributo class vale principal tiene una anchura de $80\% \times 600\text{px} = 480\text{px}$.

Colores en CSS

En esta lección se comentan las propiedades CSS y los códigos de colores definidos en la recomendación CSS 3 Color, publicada en agosto de 2021.

Estas propiedades permiten establecer el color del texto y la opacidad/transparencia de un elemento.

En esta lección se explica cómo expresar colores mediante códigos numéricos (RGB, RGBA, HSL y HSLA) o mediante nombres de colores (básicos o extendidos). En la lección Esquemas de colores se explican alguna de las formas de obtener combinaciones de colores visualmente atractivas.

Color

La propiedad color permite establecer el color del texto. Las formas de expresar colores se comentan en la lección CSS: Colores.

Opacidad (transparencia): opacity

Opacity

La propiedad opacity permite hacer que un elemento sea parcialmente transparente. El valor de esta propiedad debe ser un valor decimal entre 0 y 1 (el valor 0 hace que el elemento sea completamente transparente y el valor 1 hace que el elemento sea completamente opaco).

Si un elemento parcialmente transparente se superpone a otro elemento de otro color, el navegador mezcla los colores de los elementos, como muestra el siguiente ejemplo:

Códigos de colores RGB

colores RGB

Los códigos de colores RGB ya formaban parte de la recomendación CSS 1 (diciembre de 1996).

Las pantallas de ordenador consiguen los colores mezclando tres colores básicos (rojo, verde y azul). Cada color admite 256 niveles de intensidad, lo que hace un total de $256 \times 256 \times 256 = 16.777.216$ colores distintos. Para hacer referencia a un color concreto, basta con indicar las intensidades de cada uno de los tres colores básicos.

Los códigos RGB se pueden expresar de distintas formas:

- **rgb(rojo, verde, azul)**, donde rojo, verde y azul son números enteros desde 0 a 255.
- **rgb(rojo, verde, azul)**, donde rojo, verde y azul son porcentajes desde 0% hasta 100%.
- **#RRGGBB**, donde RR, GG y BB son números hexadecimales desde 00 hasta FF.
- **#RGB**, donde R, G y B son números hexadecimales desde 0 hasta F (el navegador transforma esos números en números de dos cifras repitiendo el valor, es decir, F se convierte en FF, 6 en 66, etc).

El color negro se consigue con la ausencia de cualquier color, así que su código RGB es `rgb(0, 0, 0)`, `rgb(0%, 0%, 0%)`, `#000000` o `#000`.

El color blanco se consigue mezclando los tres colores con la máxima intensidad, así que su código RGB es `rgb(255, 255, 255)`, `rgb(100%, 100%, 100%)`.

Colores RGBA

Los códigos de colores RGBA se incorporaron a CSS en la primera edición (junio de 2011) de la recomendación CSS 3 Color.

Estos códigos incorporan la transparencia alpha a los códigos RGB mediante un cuarto valor que indica la transparencia del elemento. La transparencia se expresa como un número decimal entre 0 y 1, en el que el 0 significa completamente transparente y el 1 completamente opaco.

Los códigos RGBA se pueden expresar de distintas formas:

- **`rgba(rojo, verde, azul, transparencia)`**, donde rojo, verde y azul son números enteros desde 0 a 255 y transparencia un número decimal entre 0 y 1.
- **`rgba(rojo, verde, azul, transparencia)`**, donde rojo, verde y azul son porcentajes desde 0% hasta 100% y transparencia un número decimal entre 0 y 1.

El cuadro siguiente permite generar colores RGBA moviendo los deslizadores. Para poder apreciar el funcionamiento de la transparencia, el cuadro muestra dos rectángulos de color, en el que el color del rectángulo situado por delante permite transparencia:

Códigos de colores HSL

Colores HSL

Los códigos de colores HSL se incorporaron a CSS en la primera edición (junio de 2011) de la recomendación CSS 3 Color.

En el modelo de color HSL el color se define mediante tres valores:

- Hue (Matiz) es un entero entre 0 y 360 y recorre todos los colores

0360Hue (Matiz) Saturation (Saturación) es un porcentaje que define la intensidad del color
Lightness (Luminosidad) es un porcentaje que indica la claridad u oscuridad del color
`{color: hsl(153, 80%, 40%);`

El modelo HSL es más intuitivo que el modelo RGB y su uso probablemente se generalice en el futuro.

Códigos de colores HSLA

Colores HSLA
















Los códigos de colores HSLA se incorporaron a CSS en la primera edición (junio de 2011) de la recomendación CSS 3 Color.

Estos códigos incorporan la transparencia alpha a los códigos HSL mediante un cuarto valor que indica la transparencia del elemento. La transparencia se expresa como en el caso de los códigos RGBA con un número decimal entre 0 y 1, en el que el 0 significa completamente transparente y el 1 completamente opaco.

Nombres de colores HTML 3.2

Colores HTML 3.2

En la recomendación HTML 3.2 (aprobada el 14 de enero de 1997) se incluyeron dieciséis nombres de colores: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white y yellow. Estos colores son los colores de la paleta VGA de Windows. La tabla e ilustración siguientes muestran estos 16 colores, sus nombres y códigos RGB.

Color	Nombre	Código RGB	Color	Nombre	Código RGB
	blue	#0000FF		navy	#000080
	fuchsia	#FF00FF		purple	#800080
	red	#FF0000		maroon	#800000
	yellow	#FFFF00		olive	#808000
	lime	#00FF00		green	#008000
	aqua	#00FFFF		teal	#008080
	black	#000000		gray	#808080
	white	#FFFFFF		silver	#C0C0C0

Colores SVG / X11

Los códigos de colores SVG (o X11) se incorporaron a CSS en la primera edición (junio de 2011) de la recomendación CSS 3 Color, aunque los navegadores los aceptaban desde mucho antes. Se muestran a continuación los 147 colores, sus nombres y códigos RGB y HSL.

Los colores del sistema se introdujeron en la recomendación CSS 2 y hacen referencia a los colores utilizados por el interfaz del sistema operativo.

Los navegadores interpretaban e interpretan correctamente estos nombres de colores, pero en estos apuntes estos colores ya no se comentan en esta lección, sino en la lección de elementos obsoletos, porque en la primera edición (junio de 2011) de la recomendación CSS 3 Color estos nombres pasaron a considerarse obsoletos, en favor de la propiedad `appearance`.

El problema es que la propiedad `appearance` no está incluida en ninguna recomendación ya aprobada. Esta propiedad se encuentra definida únicamente para elementos de formularios en la futura recomendación CSS 4: Interfaz de usuario básico, actualmente (octubre de 2021) en elaboración, pero parece que esta propiedad tiene serios problemas de incompatibilidad entre navegadores.

Mientras los navegadores los admitan, los nombres de colores de sistema se pueden utilizar, pero nada garantizada que los navegadores dejen de aceptarlos en un futuro.

Modelo de Caja de CSS

El **modelo de caja CSS** es un módulo CSS que define cajas rectangulares, incluyendo sus rellenos y márgenes, que son generadas para los elementos y que se disponen de acuerdo al modelo de formato visual.

Propiedades

Propiedades que controlan el flujo del contenido en una caja.

- [`box-decoration-break` \(en-US\)](#)
- [`box-sizing`](#)
- [`overflow`](#)
- [`overflow-x` \(en-US\)](#)
- [`overflow-y`](#)

Propiedades que controlan el tamaño de una caja.

- [`height`](#)
- [`width`](#)
- [`max-height`](#)
- [`max-width` \(en-US\)](#)
- [`min-height`](#)
- [`min-width`](#)

Propiedades que controlan los márgenes de una caja.

- [`margin`](#)
- [`margin-bottom`](#)
- [`margin-left` \(en-US\)](#)
- [`margin-right`](#)
- [`margin-top` \(en-US\)](#)

Propiedades que controlan los rellenos de una caja

- [`padding`](#)

- [padding-bottom](#)
- [padding-left \(en-US\)](#)
- [padding-right \(en-US\)](#)
- [padding-top](#)

Otras propiedades

- [box-shadow](#)
- [visibility](#)

Position

La propiedad position de CSS especifica cómo un elemento es posicionado en el documento. Las propiedades top, right, bottom, y left determinan la ubicación final de los elementos posicionados.

Tipos de posicionamiento

- Un **elemento posicionado** es un elemento cuyo valor computado de position es relative, absolute, fixed, o sticky. (En otras palabras, cualquiera excepto static).
- Un **elemento posicionado relativamente** es un elemento cuyo valor computado de position es relative. Las propiedades top y bottom especifican el desplazamiento vertical desde su posición original; las propiedades left y right especifican su desplazamiento horizontal.
- Un **elemento posicionado absolutamente** es un elemento cuyo valor computado de position es absolute o fixed. Las propiedades top, right, bottom, y left especifican el desplazamiento desde los bordes del bloque contenedor del elemento. (El bloque contenedor es el ancestro relativo al cual el elemento está posicionado). Si el elemento tiene márgenes, se agregarán al desplazamiento. el elemento establece un nuevo contexto de formato de bloque para su contenido
- Un **elemento posicionado fijamente** es un elemento cuyo valor de position computado es sticky. Es tratado como un elemento posicionado relativamente hasta que su bloque contenedor cruza un límite establecido (como por ejemplo dando a top cualquier valor distinto de auto), dentro de su flujo principal (o el contenedor dentro del cual se mueve), desde el cual es tratado como "fijo" hasta que alcance el borde opuesto de su bloque contenedor.

La mayoría de las veces, los elementos absolutamente posicionados que tienen su height y width establecidos en auto son ajustados hasta acomodarse a su contenido. Sin embargo, elementos non-replaced y absolutamente posicionados se pueden crear para llenar el espacio vertical disponible, especificando tanto top como bottom, y dejando height sin especificar (es decir, auto). De igual manera se pueden utilizar para llenar el espacio horizontal disponible especificando tanto left como right, y dando a width el valor de auto.

Principales propiedades CSS

El lenguaje CSS que estamos usando actualmente es la versión 2.1 que es estándar y la 3.0 que todavía no está terminada de definir pero ya se está usando. La versión 3.0 agrega nuevas propiedades, que se suman a las anteriores. A continuación describo las principales propiedades:

Font-family:

Define la familia tipográfica. Es conveniente poner una lista de dos o tres tipografías separadas por coma, porque si el usuario no tiene instalada la tipografía que nosotros elegimos, el navegador opta por mostrar la siguiente que debería ser una similar, si tampoco la tiene instalada, mostrará la tipografía por defecto.

Tenemos 3 opciones para definir la elección de la familia tipográfica: 1) usar una de las fuentes más comunes (web safe fonts) que muy probablemente tenga instalada el usuario (Arial, Verdana, Trebuchet, Georgia, san serif, Times), 2) subir nuestra propia fuente con una propiedad nueva que se llama `@font-face`, o 3) usar una fuente de Google Font, que en este momento es lo más recomendable. El sitio es: <http://www.google.com/fonts/>.

Font-size:

Define el tamaño de la fuente y el valor se puede escribir en pixels o en ems. En este momento se recomienda usar ems. Los dos son valores relativos, el pixel es un valor relativo a la resolución de la pantalla, pero el em es relativo al tamaño de la fuente definida por el usuario. Si el usuario no cambió la configuración, el valor por defecto de los textos en todos los navegadores es de 16px. Entonces 1em = 16px.

Color:

Define el color de la tipografía. Los colores se pueden escribir de 3 formas distintas: con sistema hexadecimal, por ejemplo: #FF0000 (es rojo). Con los nombres de los colores (más limitado) por ejemplo: black, red, green. O usando RGB, esta paleta permite agregar el canal alfa para hacer transparencias.

Width:

Define el ancho de un elemento, el valor se puede escribir en pixels, ems o porcentaje.

Max-width o min-width:

Definen el ancho máximo o mínimo de un elemento. Muy importante en sitios adaptables

Height:

Define el alto de un elemento, el valor se puede escribir en pixels, ems o porcentaje.

Max-height o min-height:

Definen el alto máximo o mínimo de un elemento. Muy importante en sitios adaptables

Padding:

Es la distancia desde el borde de un elemento hasta su contenido.

Margin:

Es la distancia entre un elemento y otro (desde el borde de un elemento hacia afuera)

Border:

Define el borde de un elemento, su color, su estilo y grosor.

Background:

Define los fondos de un objeto. El fondo puede ser una imagen o un color. El color puede ser pleno o degradado. La imagen se puede repetir formando una trama (es lo que ocurre por defecto) o se puede especificar que no repita y que se coloque en determinada posición.

Interface de Usuario

La **interfaz de usuario** o UI (User **Interface**) es un concepto **que** abarca arquitectura de información, patrones y diferentes elementos visuales **que** nos permiten interactuar de forma eficaz con sistemas operativos y software de diversos dispositivos

-Box sizing (tamaño de la caja)

La propiedad [CSS](#) establece cómo se calcula el ancho y alto total de un elemento.**box-sizing**

De forma predeterminada, en el modelo de cuadro CSS , el widthy heightque asigna a un elemento se aplica solo al cuadro de contenido del elemento. Si el elemento tiene algún borde o relleno, este se agrega al widthy heightpara llegar al tamaño del cuadro que se representa en la pantalla. Esto significa que cuando establece widthy height, tiene que ajustar el valor que da para permitir cualquier borde o relleno que se pueda agregar. Por ejemplo, si tiene cuatro cuadros con width: 25%;, si alguno tiene relleno izquierdo o derecho o un borde izquierdo o derecho, no se ajustarán de forma predeterminada en una línea dentro de las restricciones del contenedor principal.

La box-sizingpropiedad se puede utilizar para ajustar este comportamiento:

- content-boxle da el comportamiento predeterminado de tamaño de cuadro CSS. Si establece el ancho de un elemento en 100 píxeles, el cuadro de contenido del elemento tendrá 100 píxeles de ancho y el ancho de cualquier borde o relleno se agregará al ancho renderizado final, lo que hará que el elemento tenga más de 100 píxeles.

- border-box le dice al navegador que tenga en cuenta cualquier borde y relleno en los valores que especifique para el ancho y la altura de un elemento. Si establece el ancho de un elemento en 100 píxeles, esos 100 píxeles incluirán cualquier borde o relleno que haya agregado, y el cuadro de contenido se reducirá para absorber ese ancho adicional. Por lo general, esto hace que sea mucho más fácil cambiar el tamaño de los elementos. box-sizing: border-boxes el estilo predeterminado que usan los navegadores para los elementos [<table>](#), [<select>](#) y [<button>](#), y para los [<input>](#) elementos cuyo tipo es [radio](#), [checkbox](#), [reset](#), [button](#), [submit](#), [color](#) o [search](#).

Nota: A menudo es útil configurar box-sizing para border-box diseñar elementos. Esto hace que lidiar con los tamaños de los elementos sea mucho más fácil y, en general, elimina una serie de dificultades con las que puede tropezar al diseñar su contenido. Por otro lado, cuando se usa position: relative o position: absolute, el uso de box-sizing: content-box permite que los valores de posicionamiento sean relativos al contenido e independientes de los cambios en los tamaños de borde y relleno, lo que a veces es deseable.

Sintaxis

box-sizing: border-box;
box-sizing: content-box;

/* Global values */

box-sizing: inherit;
box-sizing: initial;
box-sizing: revert;
box-sizing: revert-layer;
box-sizing: unset;

Copiar al portapapeles

La box-sizing propiedad se especifica como una sola palabra clave elegida de la lista de valores a continuación.

Valores

content-box

Este es el valor inicial y predeterminado según lo especificado por el estándar CSS. Las propiedades [width](#) y [height](#) incluyen el contenido, pero no incluyen el relleno, el borde o el margen. Por ejemplo, `.box {width: 350px; border: 10px solid black;}` representa un cuadro de 370 px de ancho.

Aquí, las dimensiones del elemento se calculan como: *ancho = ancho del contenido* y *alto = alto del contenido*. (Los bordes y el relleno no se incluyen en el cálculo).

border-box

Las propiedades [width](#) y [height](#) incluyen el contenido, el relleno y el borde, pero no incluyen el margen. Tenga en cuenta que el relleno y el borde estarán dentro de la

caja. Por ejemplo, `.box {width: 350px; border: 10px solid black;}` representa un cuadro de 350 px de ancho, con un área para el contenido de 330 px de ancho. El cuadro de contenido no puede ser negativo y se reduce a 0, por lo que es imposible usarlo `border-box` para hacer que el elemento desaparezca.

Aquí las dimensiones del elemento se calculan como: *ancho = borde + relleno + ancho del contenido*, y *alto = borde + relleno + alto del contenido*.

sintaxis formal

<code>box-sizing</code>	=
<code>content-box</code>	↓
<code>border-box</code>	

Ejemplos

Tamaños de cuadro con cuadro de contenido y cuadro de borde

Este ejemplo muestra cómo diferentes `box-sizing` valores alteran el tamaño representado de dos elementos idénticos.

HTML

```
<div class="content-box">Content box</div>
<br />
<div class="border-box">Border box</div>
```

Copiar al portapapeles

CSS

```
div {
  width: 160px;
  height: 80px;
  padding: 20px;
  border: 8px solid red;
  background: yellow;
}

.content-box {
  box-sizing: content-box;
  /* Total width: 160px + (2 * 20px) + (2 * 8px) = 216px
  Total height: 80px + (2 * 20px) + (2 * 8px) = 136px
  Content box width: 160px
  Content box height: 80px */
}

.border-box {
  box-sizing: border-box;
  /* Total width: 160px
  Total height: 80px
  Content box width: 160px - (2 * 20px) - (2 * 8px) = 104px
  Content box height: 80px - (2 * 20px) - (2 * 8px) = 24px */
}
```


-Flex box (Caja Flexible)

El Módulo de Caja Flexible, comúnmente llamado flexbox, fue diseñado como un modelo unidimensional de layout, y como un método que pueda ayudar a distribuir el espacio entre los ítems de una interfaz y mejorar las capacidades de alineación. Este artículo hace un repaso de las principales características de flexbox, las que exploraremos con mayor detalle en el resto de estas guías.

Cuando describimos a flexbox como unidimensional destacamos el hecho que flexbox maneja el layout en una sola dimensión a la vez — ya sea como fila o como columna. Esto contrasta con el modelo bidimensional del Grid Layout de CSS, el cual controla columnas y filas a la vez.

Los dos ejes de flexbox

Cuando trabajamos con flexbox necesitamos pensar en términos de dos ejes — el eje principal y el eje cruzado. El eje principal está definido por la propiedad [flex-direction](#), y el eje cruzado es perpendicular a este. Todo lo que hacemos con flexbox está referido a estos dos ejes, por lo que vale la pena entender cómo trabajan desde el principio.

El eje principal

El eje principal está definido por flex-direction, que posee cuatro posibles valores:

- row
- row-reverse
- column
- column-reverse

Si elegimos row o row-reverse, el eje principal correrá a lo largo de la fila según la **dirección de la línea**.

Al elegir column o column-reverse el eje principal correrá desde el borde superior de la página hasta el final — según la **dirección del bloque**.

El eje cruzado

El eje cruzado va perpendicular al eje principal, y por lo tanto si flex-direction (del eje principal) es row o row-reverse el eje cruzado irá por las columnas.

Si el eje principal es column o column-reverse entonces el eje cruzado corre a lo largo de las filas.

Entender cuál eje es cuál es importante cuando empezamos a mirar la alineación y justificación flexible de los ítems; flexbox posee propiedades que permiten alinear y justificar el contenido sobre un eje o el otro.

Líneas de inicio y de fin

Otra área vital de entendimiento es cómo flexbox no hace suposiciones sobre la manera de escribir del documento. En el pasado, CSS estaba muy inclinado hacia el modo de escritura horizontal y de izquierda a derecha. Los métodos modernos de layout acogen la totalidad de modos de escritura así que no es necesario asumir que una línea de texto empezará arriba del documento y correrá de izquierda a derecha, con nuevas líneas dispuestas una abajo de la otra.

Puede leer más acerca de la relación que hay entre flexbox y la especificación de los Modos de Escritura en un artículo posterior, sin embargo la siguiente descripción debería ayudar para explicar porqué no se habla de izquierda y derecha ni de arriba o abajo a la hora de describir la dirección en la que fluyen los ítems flex.

En ambos casos el margen inicial del eje cruzado estará en el extremo superior del contenedor flex y el margen final en el extremo inferior, ya que ambos idiomas tiene un modo de escritura horizontal.

Después de un tiempo, pensar en inicial y final en vez de izquierda y derecha se hará natural, y será útil cuando interactúe con otros métodos de layout tales como el CSS Grid Layout que sigue los mismos patrones.

El contenedor flex

Un área del documento que contiene un flexbox es llamada **contenedor flex**. Para crear un contenedor flex, establecemos la propiedad del área del contenedor [display](#) como flex o inline-flex. Tan pronto como hacemos esto, los hijos directos de este contenedor se vuelven **ítems flex**. Como con todas las propiedades de CSS, se definen algunos valores iniciales, así que cuando creemos un contenedor flex todos los ítems flex contenidos se comportarán de la siguiente manera.

- Los ítems se despliegan sobre una fila (la propiedad flex-direction por defecto es row).
- Los ítems empiezan desde el margen inicial sobre el eje principal.
- Los ítems no se ajustan en la dimensión principal, pero se pueden contraer.
- Los ítems se ajustarán para llenar el tamaño del eje cruzado.
- La propiedad [flex-basis](#) es definida como auto.
- La propiedad [flex-wrap](#) es definida como nowrap.

El resultado es que todos los ítems se alinearán en una solo fila, usando el tamaño del contenedor como su tamaño en el eje principal. Si hay más ítems de los que caben en el contenedor, estos no pasarán más abajo si no que sobrepasarán el margen. Si hay ítems más altos que otros, todos los ítems serán ajustados en el eje cruzado para alcanzar al mayor.

Se puede ver en el ejercicio en vivo de abajo cómo luce. Intente editar el ítem o añadir ítems adicionales para así probar el comportamiento inicial de flexbox.

Cambiar flex-direction

Al añadir la propiedad [flex-direction](#) en el contenedor flex nos permite cambiar la dirección de cómo los ítems son desplegados. Colocando `flex-direction: row-reverse` se mantendrá el despliegue a lo largo de la fila, sin embargo el inicio y final quedarán al revés del original.

Si cambiamos `flex-direction` a `column` el eje principal se cambiará y los ítems aparecerán en una columna. Colocando `column-reverse` las líneas de inicio y fin serán nuevamente puestas al revés.

El ejemplo en vivo de abajo tiene `flex-direction` puesto como `row-reverse`. Pruebe los otros valores — `row`, `column` y `column-reverse` — para ver qué sucede con el contenido.

Contenedores flex Multi-línea con flex-wrap

Si bien flexbox es un modelo unidimensional, es posible lograr que nuestros ítems flex sean repartidos en varias líneas. Haciendo esto, se deberá considerar cada línea como un nuevo contenedor flex. Cualquier distribución del espacio solo sucederá dentro de esa línea, sin referenciar las líneas colaterales.

Para lograr repartirse en varias líneas añada la propiedad [flex-wrap](#) con el valor `wrap`. Cuando los ítems sean demasiados para desplegarlos en una línea, serán repartidos en la línea siguiente. El ejemplo en vivo de abajo contiene ítems que se les ha asignando un ancho, donde el ancho total de los ítems excede al del contenedor flex. Cuando `flex-wrap` se coloca como `wrap`, los ítems se repartirán. Al colocarlo como `nowrap`, el cual es el valor inicial, estos se contraerán para calzar con el contenedor ya que usan los valores iniciales de flexbox que permiten que los ítems se contraigan. Al usar `nowrap` los ítems podrían salirse del margen si estos no pudieran contraerse, o no contraerse lo suficiente para ser calzados.

La abreviatura flex-flow

Se pueden combinar las propiedades `flex-direction` y `flex-wrap` en la abreviatura [flex-flow](#). El primer valor especificado es `flex-direction` y el segundo valor es `flex-wrap`.

En el ejemplo en vivo de abajo intente cambiar el primer valor por uno de los valores permitidos para `flex-direction` - `row`, `row-reverse`, `column` o `column-reverse`, y cambie también el segundo valor por `wrap` y `nowrap`.

Propiedades aplicadas a los ítems flex

Para obtener más control sobre los ítems flex podemos apuntarlos directamente. Hacemos esto a través de tres propiedades:

- [flex-grow](#)
- [flex-shrink](#)
- [flex-basis](#)

Daremos un breve vistazo a estas propiedades en este resumen, y en un próximo artículo ahondaremos sobre su comportamiento.

Antes de darle sentido a estas propiedades debemos considerar el concepto de **espacio disponible**. Lo que hacemos cuando cambiamos el valor de alguna de estas propiedades es cambiar la forma que se distribuye el espacio disponible entre nuestros ítems. Este concepto de espacio disponible es también importante cuando veamos la alineación de ítems.

Si tenemos tres ítems con un ancho de 100 píxeles en un contenedor de 500 píxeles de ancho, entonces el espacio que se necesita para colocar nuestros ítems es de 300 píxeles. Esto deja 200 píxeles de espacio disponible. Si no cambiamos los valores iniciales entonces flexbox colocará ese espacio después del último ítem.

Si en cambio quisiéramos que los ítems crecieran para llenar ese espacio, entonces necesitaremos un método para distribuir el espacio sobrante entre los ítems. Es justo lo que harán las propiedades flex que aplicaremos a dichos ítems.

La propiedad flex-basis

Con flex-basis se define el tamaño de un ítem en términos del espacio que deja como espacio disponible. El valor inicial de esta propiedad es auto — en este caso el navegador revisa si los ítems definen un tamaño. En el ejemplo de arriba, todos los ítems tienen un ancho de 100 píxeles así que este es usado como flex-basis.

Si los ítems no tiene un tamaño entonces el tamaño de su contenido es usado como flex-basis. Y por eso, apenas declarado display: flex en el padre a fin de crear ítems flex, todos estos ítems se ubicaron en una sola fila y tomaron solo el espacio necesario para desplegar su contenido.

La propiedad flex-grow

Con la propiedad flex-grow definida como un entero positivo, los ítems flex pueden crecer en el eje principal a partir de flex-basis. Esto hará que el ítem se ajuste y tome todo el espacio disponible del eje, o una proporción del espacio disponible si otro ítem también puede crecer.

Si le damos a todos los ítems del ejemplo anterior un valor flex-grow de 1 entonces el espacio disponible en el contenedor flex será compartido igualitariamente entre estos ítems y se ajustarán para llenar el contenedor sobre el eje principal.

Podemos usar flex-grow apropiadamente para distribuir el espacio en proporciones. Si otorgamos al primer ítem un valor flex-grow de 2 y a los otros un valor de 1, entonces 2

partes serán dadas al primer ítem (100px de 200px en el caso del ejemplo de arriba) y 1 parte para cada uno de los restantes (cada uno con 50px de los 200px en total).

La propiedad flex-shrink

Así como la propiedad flex-grow se encarga de añadir espacio sobre el eje principal, la propiedad flex-shrink controla como se contrae. Si no contamos con suficiente espacio en el contenedor para colocar los ítems y flex-shrink posee un valor entero positivo, el ítem puede contraerse a partir de flex-basis. Así como podemos asignar diferentes valores de flex-grow con el fin que un ítem se expanda más rápido que otros — un ítem con un valor más alto de flex-shrink se contraerá más rápido que sus hermanos que poseen valores menores.

El tamaño mínimo del ítem tendrá que ser considerado cuando se determine un valor de contracción que pueda funcionar, esto significa que flex-shrink tiene el potencial de comportarse menos consistentemente que flex-grow. Por lo tanto, haremos una revisión más detallada de cómo este algoritmo trabaja en el artículo Controlling Ratios de los ítems sobre el eje principal.

Nota: Nótese que los valores de flex-grow y flex-shrink son proporciones. Típicamente si pusiéramos todos los ítems flex: 1 1 200px y luego quisiéramos que un ítem creciera al doble, deberíamos ponerlo con flex: 2 1 200px. Aunque igualmente podemos colocar flex: 10 1 200px y flex: 20 1 200px si quisiéramos.

Valores abreviados para las propiedades flex

Difícilmente veremos la propiedades flex-grow, flex-shrink y flex-basis usadas individualmente; si no que han sido combinadas en la abreviación [flex](#). La abreviación flex permite establecer los tres valores en este orden: flex-grow, flex-shrink, flex-basis.

El ejemplo en vivo de más abajo permite probar los diferentes valores de la abreviación flex; recuerde que el primer valor es flex-grow. Dándole un valor positivo significa que el ítem puede crecer. El segundo es flex-shrink — con un valor positivo los ítems pueden contraerse. El valor final es flex-basis; este es el valor que los ítems usan como valor base para crecer y contraerse.

Hay además algunas abreviaturas de valores que cubren la mayoría de los casos de uso. Se ven con frecuencia utilizados en tutoriales, y en muchos casos es todo lo que necesitamos usar. Los valores predefinidos son los siguientes:

- flex: initial
- flex: auto
- flex: none
- flex: <positive-number>

Fijando flex: initial el ítem se restablece con los valores iniciales de Flexbox. Es lo mismo que flex: 0 1 auto. En este caso el valor de flex-grow es 0, así que los ítems no crecerán más de su tamaño flex-basis. El valor flex-shrink es 1, así que los ítems pueden contraerse si es

necesario en vez de salirse de los márgenes. El valor de `flex-basis` es `auto`. Los ítems pueden definir un tamaño en la dimensión del eje principal, o bien obtener su tamaño por el contenido de los mismos.

Usar `flex: auto` es lo mismo que usar `flex: 1 1 auto`, es como con `flex: initial` pero en este caso los ítems pueden crecer y llenar el contenedor así como encoger si se requiere.

Al usar `flex: none` se crearán ítems `flex` totalmente inflexibles. Es como escribir `flex: 0 0 auto`. Los ítems no pueden ni crecer ni encoger pero serán colocados usando `flexbox` con `flex-basis` en `auto`.

Una abreviación que es común en tutoriales es `flex: 1` o `flex: 2` y más. Es como usar `flex: 1 1 0`. Los ítems pueden crecer o encoger con un `flex-basis` de 0.

Pruebe estas abreviaciones de valores en el ejemplo en vivo de abajo.

Alineación, justificación y distribución del espacio libre entre ítems

Una característica clave de `flexbox` es la capacidad de alinear y justificar ítems sobre los ejes principal y cruzado, y distribuir el espacio entre los ítems `flex`.

`align-items`

La propiedad [`align-items`](#) alineará los ítems sobre el eje cruzado.

El valor inicial para esta propiedad es `stretch` razón por la cual los ítems se ajustan por defecto a la altura de aquel más alto. En efecto se ajustan para llenar el contenedor `flex` — el ítem más alto define la altura de este.

En cambio definimos `align-items` como `flex-start` para que los ítems se alineen al comienzo del contenedor `flex`, `flex-end` para alinearlos al final, o `center` para alinearlos al centro. Intente esto en el ejemplo en vivo — He definido en el contenedor `flex` una altura para que se aprecie que se pueden mover libremente dentro del contenedor. Vea lo que sucede si se coloca el valor `align-items` como:

- `stretch`
- `flex-start`
- `flex-end`
- `center`

`justify-content`

La propiedad [`justify-content`](#) es usada para alinear los ítems en el eje principal, cuyo `flex-direction` define la dirección del flujo. El valor inicial es `flex-start` que alineará los ítems al inicio del margen del contenedor, pero también se podría definir como `flex-end` para alinearlos al final, o `center` para alinearlos al centro.

También podemos usar `space-between` para tomar todo el espacio sobrante después de que los ítems hayan sido colocados, y distribuir de forma pareja los ítems para que haya un espacio equitativo entre cada ítem. O bien, usamos el valor `space-around` para crear un espacio equitativo a la derecha e izquierda de cada ítem.

Pruebe con los siguientes valores de `justify-content` en el ejemplo en vivo:

- `space-evenly`
- `flex-start`
- `flex-end`
- `center`
- `space-around`
- `space-between`

-Media Query (Cónsula de Medios)

Las **media queries** (en español "consultas de medios") son útiles cuando deseas modificar tu página web o aplicación en función del tipo de dispositivo (como una impresora o una pantalla) o de características y parámetros específicos (como la resolución de la pantalla o el ancho del [viewport](#) del navegador).

Un media query es una característica de CSS. Te permite crear e implementar diseños que se adaptan a las propiedades del dispositivo que estás usando. Algunas de estas propiedades incluyen el alto y ancho de una página.

En esta breve guía, verás cómo puedes establecer múltiples propiedades de anchura en una regla de media query. Por ahora vamos a dar un vistazo a los fundamentos primero.

Cómo funcionan los **Media Queries en CSS**

Ahora que tienes una idea básica de lo que es un media query, echemos un vistazo a cómo funciona realmente esta particular característica de CSS.

Un media query básico luce así:

```
@media only screen and (max-width: 576px) {  
  // hacer algo  
}
```

```
@media only screen and (min-width: 576px) {  
  // volver hacer algo
```

}

Esto significa que los estilos que se escriban dentro de las reglas de media queries anteriores sólo funcionarán o serán efectivos en las propiedades de ancho especificadas anteriormente.

Literalmente, está diciendo que a este ancho en particular (es decir, la propiedad max-width de 576px), CSS por favor aplica los estilos que voy a escribir aquí sólo a este ancho, desde un ancho inicial de 0px.

Propiedades max-width y min-width

Hay dos cosas que debes tener en cuenta al crear media queries para diferentes tamaños de pantalla: las propiedades max-width y min-width.

Cuando se pasa una propiedad max-width a un media query, CSS lo interpreta como un ancho que comienza en cero – eso si aún no se ha establecido una propiedad de ancho mínimo. Pronto llegaremos a eso.

Una vez que la propiedad max-width tiene un valor asignado, todos los estilos dentro de esa media query particular se aplicarán a cualquier dispositivo cuyo tamaño de pantalla abaque desde 0px hasta el ancho máximo especificado.

La propiedad min-width, por otro lado, toma el valor inicial que le has asignado y aplica los estilos dentro de la regla del media query hasta que se proporcione el siguiente max-width is provided. Digamos por ejemplo:

```
@media only screen and (min-width: 576px) {  
    // aplicar los estilos aquí a partir de este ancho mínimo de  
    // 576px (dispositivos con tamaño de pantalla medio)  
}
```

Los estilos que se escriban en el media query anterior sólo serían aplicables a los dispositivos que se encuentren dentro del ancho mínimo especificado y superior.

Como establecer el rango de ancho de un Media Query

El método que acabamos de comentar de crear media queries aplicando sólo una propiedad width resuelve sólo un problema.

Al establecer "rango de ancho", tienes cierta flexibilidad a la hora de crear diseños que te brinda capacidad de respuesta en todos los anchos de dispositivos.

Establecer un "rango de ancho" particular no es diferente de la forma en que se crean los media queries. La única diferencia es la adición de más expresiones de media feature (es decir, los tamaños de ancho de pantalla).

Echa un vistazo:


```
@media only screen and (min-width: 360px) and (max-width: 768px) {  
    // hacer algo en ese rango de ancho.  
}
```

La media query anterior sólo funcionará para la expresión de características (el tamaño de pantalla del dispositivo móvil para el que está escribiendo un estilo) proporcionada anteriormente.

Toma la primera expresión de ancho proporcionada como valor inicial y la segunda como valor final.

¿Recuerdas que vimos la diferencia entre la propiedad max-width y min-width mas arriba? Simplemente le estamos diciendo al navegador que aplique los estilos CSS que escribiremos dentro de esta regla a los dispositivos móviles con tamaños de pantalla de 360px a 768px.

En términos básicos, estamos construyendo diseños que serán responsivos desde anchos de dispositivos pequeños hasta anchos medios, como tabletas o dispositivos móviles.

Puedes echar un vistazo a algunos media breakpoints que están disponibles en la documentación de Bootstrap. Prueba jugando con ellos estableciendo los rangos de tamaño de pantalla que prefieras.

Probando Media Queries con Flexbox

Has visto cómo crear una media query básica que toma múltiples expresiones de tamaño de pantalla. Y has visto la diferencia entre las propiedades max-width y min-width y cómo aplicarlas.

En esta sección, veremos cómo crear un diseño simple cuya apariencia cambie en diferentes media breakpoints (tamaños de pantalla). Empezaremos por crear el código que contendrá el diseño.

```
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <title>Ejemplo de Media query</title>  
    <link rel="stylesheet" href="style.css" />  
  </head>  
  <body>  
    <div class="contenedor">  
      <div class="cajas caja1">
```

```

    <h1>Primera Caja</h1>
    <p>
        Información en la primera caja
    </p>
</div>
<div class="cajas caja2">
    <h1>Segunda Caja</h1>
    <p>
        Información en la segunda caja
    </p>
</div>
</body>
</html>

```

El fragmento anterior mostrará dos cajas con su respectiva información cuando se apliquen los estilos. Si deseas puedes ver el ejemplo de código completo aquí.

```

.contenedor {
    display: flex;
    flex-wrap: wrap;
    width: 980px;
    margin: 0 auto;
    margin-top: 40px;
}
@media only screen and (min-width: 320px) and (max-width: 576px) {
    .contenedor {
        width: 100%;
        padding-left: 23px;
        flex-direction: column-reverse;
    }
    .cajas {
        width: 100%;
    }
}

```

Si echas un vistazo al archivo CSS, verás que la media query tiene un ancho mínimo de 320px y un ancho máximo de 576px. Esto sólo significa que todos los estilos que irán en esta regla sólo serán aplicables a los dispositivos con anchos extra-pequeños y pequeños.

El diseño de las cajas también cambia en este rango de ancho particular. Eso se debe a que el selector `.container` tiene su propiedad `flex-direction` cambiada de `row` a `column-reverse`.

Puedes decidir experimentar con los otros valores que se pueden asignar a la propiedad `flex-direction`. Compruébalos aquí.