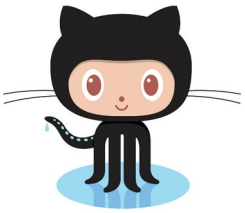
Diseño de Interfaces Web

UD – 2

Uso de estilos

Contenidos



Accesibilidad

ÍNDICE

1. INTRODUCCIÓN

Las hojas de estilos en cascada o CSS son un **conjunto de reglas** usadas para definir la presentación de un documento escrito en HTML. Le indican al navegador cómo se debe visualizar el contenido de la página web. El organismo encargado de regular su especificación es el World Wide Web Consortium (W3C). El estándar actual es CSS3 y es totalmente compatible con las versiones anteriores. Las **ventajas** que ofrece la utilización de hojas de estilo son:

- *Separar la estructura de las páginas y su contenido (en HTML), del formato del texto y de la página (en CSS).*
- *Ampliar las posibilidades de formato y de presentación de una página web.*
- *Unificar el diseño de páginas web definiéndolo una sola vez y aplicándolo tantas veces como se quiera.*
- *Reutilizar las mismos hojas de estilo para diferentes documentos HTML.*

2. INCLUIR ESTILOS CSS EN UN DOCUMENTO HTML.

Para crear un documento web con estilos CSS se parte de un documento HTML con los contenidos. Para especificar los estilos, se pueden utilizar diferentes caminos

2.1 hoja de estilos externa

Una hoja de estilos externa es un documento de texto con **extensión .css** en el que hay definidos los estilos de un documento web. Es aconsejable su uso, ya que permite su reutilización y facilita el mantenimiento del sitio web.

```
/ *fichero con los estilos -> misEstilos.css * /


h1{
  color: #FFFFFF ;
  background-color: #999999;
}


p{
  font-family: Georgia, Helvetica;
}
```

Para enlazar un documento CSS a una página web se utiliza la etiqueta <link> que se especifica en la cabecera <head> de la página.

```
<link rel="stylesheet" href="misEstilos.css" />
```

2.2 Incrustar una hoja de estilos en HTML

Una hoja de estilos puede incrustarse en el documento HTML con el elemento <style>, el cual se suele definir en la cabecera <head> del documento.

```
<head>
   <style type="text/css" media="screen">
      body {background: url(foo.gif) red; color: black;}
      p me {background-color: yellow; color: black;}
     .Notas {margin-left: 5em; margin-right: 5em;}
   </style>
</head>
```

2.3 Importar una hoja de estilos

Una hoja de estilos externa puede ser importada con la regla **@import de CSS**. Se puede especificar tanto al inicio de un fichero .css, lo más habitual, como al principio del elemento <style>.

```
<style type="text/css" media="screen, projection">
   @import  url(http://miDominio.com/css/estilos.css);
   @import url(/css/estilos2.css);
   p {background-color: yellow; color: black;}
</style >
```

2.4 Estilos en línea

Para especificar estilos en línea se utiliza el atributo **style**, que toma por valor cualquier número de propiedades CSS separadas por ";".

```
<p style="color: red; font-family: 'New Century Schoolbook', serif">
   Este párrafo muestra el texto rojo y con la fuente …
</p>
```

3. FORMATO DE UNA REGLA CSS

Para definir un estilo específico, CSS utiliza reglas que consisten en un selector y un bloque donde se declaran las diferentes propiedades que debe tener ese selector.

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

4. TABLA DE SELECTORES

Tipo de elemento	**p**	*Se aplica a todos los parrafos*
Agrupación	**p,div,href**	*Se aplica a las 3 etiquetas*
Clase	**.nombreClass** **p.nombreClass**	
Varias Clases	**p.nomClass1** **.nomClass2**	*h1 class= "nomClass1 nomClass2"*
Identificador	**#idElemento**	
Limitación de contenido	**td p**	*<td> sadfasdfasdf </td>*
Universal	***** **p ***	*Todos los hijos de p*
Atributo	**p[atributo]** **p[atributo= "valor"]** **p[lang= "es"][spell= "true"]** *…... * **p[name^="mi"]** **p[name$="mi"]** **p[name*="mi"]**	*Por atributo* *Por atributo con valor* * * *Empiece por ...* *Termine por...* *Contenga a ....*
Hijo directo	**section > p**	
Hermano contiguo	**p + img**	*// img que son hermanos inmediatos de un elemento p*
Hermano general	**p ~ img**	*// img que son hermanos no inmediatos de un elemento p*
Pseudo-elementos	**elemento::first-letter** **elemento::first-line**	*Primer letra* *Primera línea*
Peseudo-clases	**tr:first-child()** **tr:last-child()** **tr:only-child()** **tr:nth-child(n)** **tr:nth-child(*even*)** **tr:nth-child(*odd*)** **tr:empty**	*Primer hijo* *Último hijo* *Único hijo* *Hijo en posición n* *Hijos pares* *Hijos impares* *Elemento vacío(sin hijos, ni texto).*
Negación	**:not(selector)**	*Elementos q no cumplen condición*
Enlaces	**:link** **:visited**	*Enlaces no visitados* *Enlaces visitados*
Ratón	**:hover** **:active**	*Pasamos el ratón por encima del elemento* *Cuando estamos pulsando sobre el elemento.*
Formulario	**:focus** **:checked**	*Elemento que tiene el foco* *Cuando la casilla está seleccionada*

5. PRIORIDAD DE LAS DECLARACIONES CSS

Además de las hojas de estilos definidas por los **diseñadores**, los navegadores aplican a cada página otras dos hojas de estilos: la del **navegador** y la del **usuario**.

La hoja de estilos del navegador es la primera que se aplica y se utiliza para establecer el estilo inicial por defecto a todos los elementos HTML (*tamaños de letra iniciales, decoración del texto, márgenes entre elementos, etc.*) Además, cada usuario puede crear su propia hoja de estilos y aplicarla automáticamente a todas las páginas que visite con su navegador. Se trata de una opción muy útil para personas discapacitadas visualmente, ya que pueden aumentar el contraste y el tamaño del texto de todas las páginas para facilitar su lectura.

Orden en el que se aplican las diferentes hojas de estilos:

5.1 Important

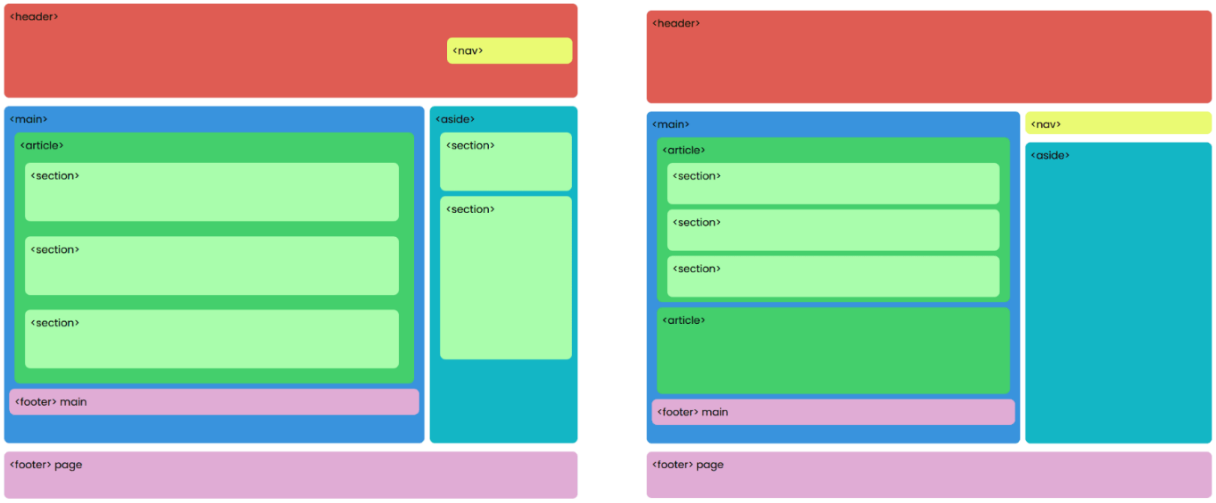
Si a una declaración CSS se le añade la palabra reservada **!important**, se aumenta su prioridad. No sólo afecta a las declaraciones simples, sino que varía la prioridad de las hojas de estilo.

6. ETIQUETAS SEMÁNTICAS

Las etiquetas semánticas ayudan a la implementación de los estándares de la *Web Semántica*, cuya finalidad es hacer el contenido de internet legible para aplicaciones informáticas.
Uno de los usos mas importantes de este tipo de etiquetas es la de marcar la información de la pagina para ser indexada por los buscadores web como Google.

Las etiquetas semánticas más importantes son:

- **header** → Representa un grupo de ayudas introductorias o de navegación. También puede contener logos, formulario de búsqueda o tabla de contenidos
- **nav** → Sección de una página que enlaza a otras páginas o partes de la misma. Una sección con links de navegación.
- **Aside** → Sección de una pagina que contiene contenido relacionado tangencialmente al de su alrededor. Por lo general se utilizan como barras laterales.
- **main** → Representa el contenido predominante de la página. Una página debe tener solo un tag <main>. Solo debe estar contenida por tags de tipo <html> <body>, o <form>.
- **article** → Representa una sección de contenido que puede ser distribuido de forma independiente. Puede ser un post en un foro, un articulo de periódico, una entrada de un blog, un comentario, etc.
- **section** → Representa a una sección genérica de una pagina. Agrupa contenidos que tienen una relación temática entre si.
- **Footer** → Representa un pie de página para el elemento que lo contiene. Generalmente contiene información acerca de quién lo escribió, enlaces a documentos relacionados, datos de derechos de autor o similares.



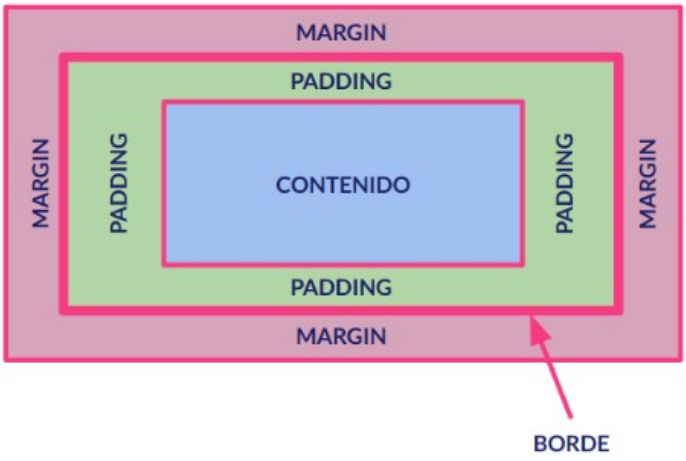
¿Has encontrado algún tipo de **error** en los dos ejemplos anteriores?

Ver Ejemplo → Etiquetas-Semanticas

7. MAQUETACIÓN TRADICIONAL

7.1 Posicionamiento tradicional

Pese a que todos los elementos de mi página HTML son cajas lo cierto es que no todas las cajas se comportan igual cuando las añadimos.



El comportamiento viene determinado por la propiedad CSS **display**.
Cada etiqueta tiene un valor por defecto para esta propiedad pero, para conseguir lo que el diseño que queremos, podremos modificarlas si lo estimamos necesario. Los valores que puede tomar son muchos pero los más usados son:

inline **inline-block** **block** **none**

Fuente : *https://developer.mozilla.org/es/docs/Web/CSS/display*

inline

Los elementos **inline** no rompen el flujo de la línea y se van colocando uno detrás de otro mientras caben. Aceptan margin y padding pero solo se tienen en cuenta los valores horizontales. Ignoran width y height.
Ejemplos: ,<img>,<a>, <b>, etc…

inline-block

Los elementos **inline-block** funcionan exactamente como los anteriores pero podremos asignarles width y height.

block

Los **elementos en bloque** rompen el flujo de la línea y provocan "un salto de línea" tanto anterior como posterior. Por defecto, si no lo especificamos, ocuparán toda la anchura de la etiqueta que los contiene, la etiqueta contenedora.
Ejemplos: <h1>,<p>,<section>, ,<li>,<nav>, etc, …

none

Elementos con valor none, son elementos que desaparecen de la página. No dejan un espacio vacío aunque siguen en el código HTML.
Aclaración La propiedad visibility:hidden sí que se deja ese hueco aunque no se muestre.

Ver Ejemplo → Propiedad-Display

7.2 Layout

Dependiendo de cómo llevemos a cabo nuestra maquetación podemos tener distintos tipos de layouts:

Fixed Elastic Fluid

Fixed - px

Este tipo de layouts establecen un tamaño fijo en pixels para la anchura de los distintos elementos.

La principal **ventaja** al usar este tipo de layouts es que tengo contral total, los distintos elementos van a tener siempre el tamaño que yo quiero.

Sin embargo esta ventaja no compensa lass principales **desventajas**:

- *En pantallas pequeñas puede aparecer un scroll horizontal y esto es un gran error en diseño web, ya que las páginas con scroll horizontal son poco usables.*
- *En pantallas muy anchas puede que tenga mucho espacio en blanco a los lados del contenedor principal.*

Elastic - em

Este tipo de layouts establece la anchura de los elementos en em que es el tamaño de letra por defecto (suele ser 16px).

La principal **ventaja** es que al escalar el texto haciendo zoom in o zoom out los elementos cuyas dimensiones se hayan establecido en em escalarán correctamente.

Sin embargo, hay una **desventaja** importante:

- *Si escalo elementos que ocupen el mismo espacio éstos pueden solaparse y la única forma de solucionar este problema sería comprobar el tamaño de las fuentes en todo tipo de dispositivos y todo tipo de tamaños.*

Fluid - %

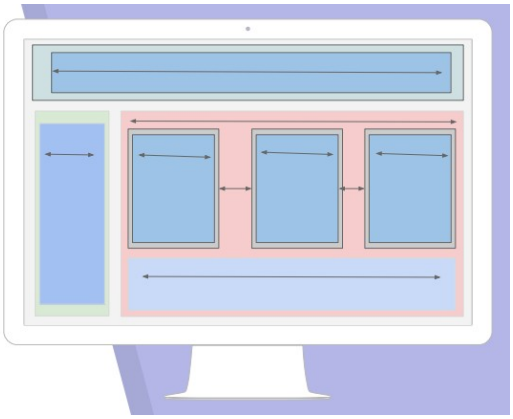
Este tipo de layouts establecen el ancho de los distintos contenedores en % (con respecto al contenedor/etiqueta padre).

Si decidimos usar este tipo de layout la principal **ventaja** es que los elementos mantienen sus proporciones independientemente del tamaño de la pantalla.

Sin embargo, debemos afrontar unas importantes **desventajas**:

- *En pantallas pequeñas las columnas puede ser muy estrechas.*
- *En columnas estrechas los textos largos provocan celdas muy alargadas.*
- *Si tengo imágenes o vídeos con un tamaño fijo tendré problemas :(*

7.3 content-box vs border-box



Por defecto los navegadores utilizan la propiedad **content-box**, apareciendo el inconveniente de las dimensiones de los elementos:

- **La altura del elemento** → altura del contenido + el padding + el borde.
- **La anchura del elemento** → anchura del contenido + el padding + el borde.

En esas circunstancias, y con un diseño complejo, cuando queramos cuadrar todo perfectamente vamos a tener que echar cuentas de todo, sumar todo para todas las cajas, añadir los márgenes etc…

La mejor solución para eso es establecer la propiedad CSS **border-box** y de esta manera no tendremos que echar cuentas con los bordes y los *paddings*. El tamaño que demos al elemento será la suma de todo.
Si queremos que siempre sea así añadiremos las siguientes líneas a mi CSS:

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

	*, *:before, *:after {
		-webkit-box-sizing: inherit;
		-moz-box-sizing: inherit;
		 box-sizing: inherit;
}
```

- Modelo **content-box** [*Modelo por cajas tradicional*]
  width + padding + border
- Modelo **border-box**:
  El width incluye el paddign y el border

***Aclaración: De esta forma al aumentar el padding o border, el elemento no crece, se mantiene el tamaño especificado en width*

7.4 Alinear HORIZONTAL

Si queremos centrar elementos en línea añadiremos la propiedad ***text-align:center*** al contenedor padre.

Si queremos centrar un elemento en bloque dentro de su etiqueta contenedora añadiremos la propiedad CSS ***margin: X auto*** al elemento que queremos centrar (*X es una distancia expresada en cualquier unidad*). El elemento debe tener anchura.

Si tenemos varios elementos en bloque dentro de un contenedor debemos

- Añadir la propiedad ***text-align:center*** al contenedor padre.
- Añadir la propiedad ***display:inline-block*** a los elementos a centrar.

7.5 Alinear VERTICAL

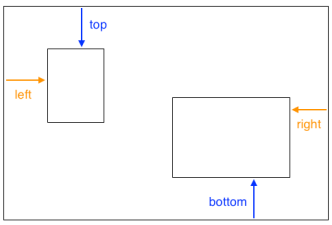
Se puede conseguir con el mismo ***padding*** arriba y abajo.

Ver Ejemplo → Alinear-Elementos

8. POSICIONAMIENTO

Conforme vayamos aprendiendo querremos hacer diseños más complejos y mover los elementos con respecto a dónde les correspondería según el flujo normal atendiendo a su propiedad display.

Para este posicionamiento más exacto y concreto CSS nos proporciona la propiedad position cuyo valores van íntimamente asociados a las propiedades CSS **top**,**bottom**, **left**, **right** y **z-index**.

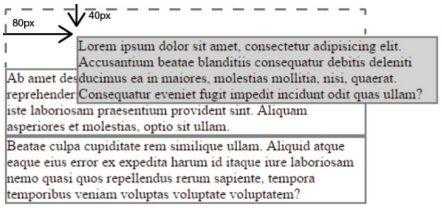


Las 4 primera de estas propiedades indicar desplazamientos conforme a un punto de referencia en concreto y la propiedad z-index nos va a permitir trabajar con capas.

Los **distintos valores que puede tomar la propiedad position** son:

- **static**:
  es el valor por defecto. El elemento sigue el flujo que le corresponde. Aunque use top,bottom,left,right o z-index al elemento **no se aplica ningún desplazamiento.**

- **relative**
  Como static, pero permite aplicar las propiedades: *left, top, right, botton*. A partir de la posición que le corresponde, se le aplica las propiedades top, left y right. Este tipo de posicionamiento no modifica el posicionamiento de las demás cajas a su alrededor y puede provocar solapamientos.



- **absolute**
  La referencia de coordenadas dependerá del contenedor padre(*el primer padre que no sea estático*). Sí influye al resto, porque se desplazan para ocupar su hueco liberado.

- **fixed**:
  Se le aplica top,bottom,right o z-index en relación con documento. No atiende al scroll una vez generada por lo que su posición siempre permanece fija.
  Es un caso particular de posicionamiento absoluto en el que siempre se toma como referencia la ventana del navegador(coordenada 0,0). Su posición en la pantalla siempre es la misma independientemente de los otros elementos y de si el usuario sube o baja la ventana del navegador**. S**u hueco es ocupado por otro elemento, y el flota por toda la pagina.

- **sticky**: Se comporta como relative hasta llegar a una posición de scroll y a partir de entonces fixed.

8.1 z-index

Al posicionar los elementos con las propiedades position puede suceder que nos encontremos que haya elementos que lleguen a solaparse por tener que ocupar la misma área.

Con z-index podemos establecer capas decidiendo el orden de solapamiento de estos elementos. Se mostrará arriba del todo aquel elemento de los que se solapen con el mayor valor de z-index.

Ver Ejemplo → Posicionar-Elementos

9. COLUMNAS

Usando CSS podemos dividir el contenido que queremos mostrar en varias columnas tal y como podemos ver en un periódico.
Para ello utilizaremos las siguiente propiedades CSS:

- **column-count** → para especificar el número de columnas que tendrá el elemento contenedor.
- **column-width** → si queremos fijar en ancho de las columnas del contenedor. El navegador calculará cuántas columnas al menos caben con ese ancho.
- **column-gap** → permite establecer la distancia que separa las distintas columnas.
- **colum-rule** → funciona de manera muy similar a borde y me permite especificar el estilo, color y anchura de la línea que separa las columnas.
- **column-span** → con valores all o none para indicar si el elemento en cuestión, que estará dentro del contenedor donde hemos especificado que habrá columnas, sigue el flujo en columnas o no.
- **column-fill** → para establecer cómo se rellenan las columnas. El contenedor debe tener altura. Los valore son auto o balance (todas las columnas la misma altura)
- **break-inside** → con valor void si queremos que el elemento no quede roto de una columna a otra.

Ver → II_9_Columnas

10. ELEMENTOS FLOTANTES

La caja se desplaza todo lo posible a la derecha o a la izquierda de la línea en la que se encuentra y deja de formar parte del flujo normal de la página. El resto de cajas ocupan su lugar.

float : *none* | *left* | *right*

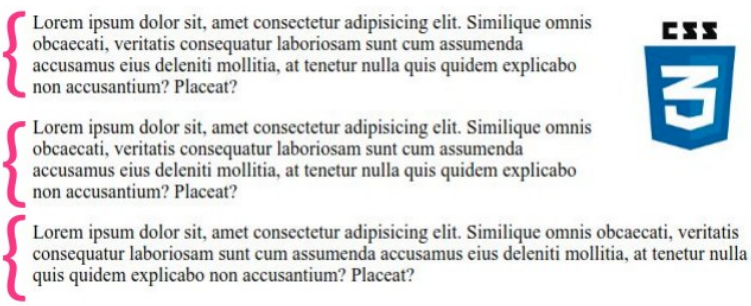
- *El elemento omite totalmente su posición, se alinea lo más posible al lado indicado dentro de su misma línea.*
- *Su uso original y para el que se creó esta propiedades, es la colocación de texto alrededor de una imagen.*

```
div.primero {
    border: 3px solid red;
    width: 100px;
    float: left;
}
div.segundo {
    border: 3px solid blue;
    width: 100px;
    float: left;
}
div.tercero {
    border: 3px solid green;
    width: 100px;
    float: right;
}
```

Aclaración: *primero y segundo con flotabilidad a la izquierda, son colocados a la izquierda como elementos de línea, uno a continuación de otro. Tercero con flotabilidad a la derecha, es colocado en la misma línea al otro extremo(el extremo derecho).*

La propiedad float (left o right) está pensada para especificar cómo se dispone un texto alrededor de una imagen...

si sigue habiendo “hueco vertical” en el lugar contrario al que he flotado la imagen los elementos se siguen añadiendo ahí hasta que sobrepasan en la “vertical” al elemento flotante.



10.1 Clear

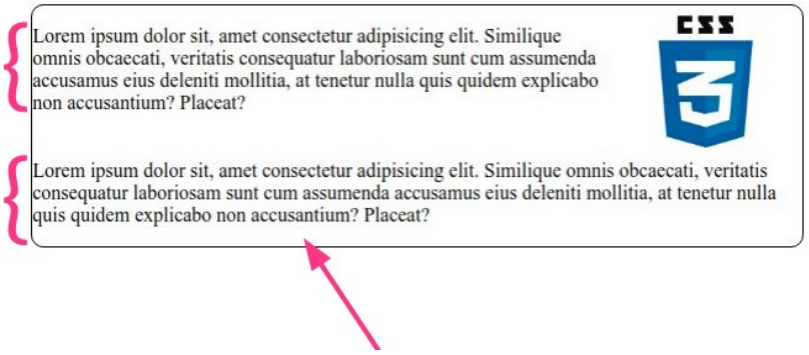
Eliminar la flotabilidad

```
clear : none | left | right | both
```

```
p.noflota {
        clear: left;
}
```

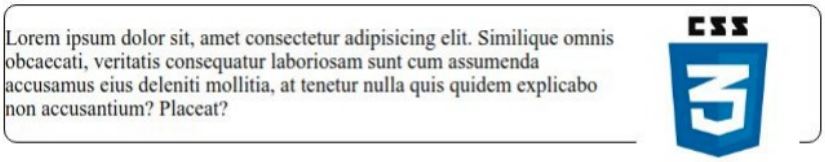
Aclaración: *Va a eliminar la característica de flotabilidad a la izquierda del elemento correspondiente.*

Si quieres "forzar" que un elemento deje de flotar debo añadir la la propiedad clear:right (o left o both) y ese elemento ya se añadirá tras el fín en vertical del elemento flotante…



10.2 CLEARFIX HACK

Problema que ocurre cuando la imagen(o elemento flotante) sale del contenedor:



La soluciones son dos (**ambas son propiedades CSS al elemento contenedor**)

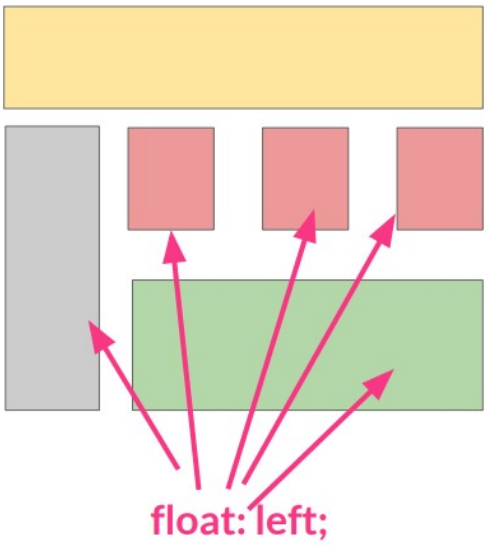
```
selector {
        overflow-y: auto;
        height: Altura_suficiente; /*Una de las dos*/
}
```

10.3 Creando layout

Posteriormente los desarrolladores se dieron cuenta de que los elementos flotantes se podían usar para maquetar.

Lo conseguiremos:

- *Flotando los elementos según necesitemos (div, nav, header…)*
- *Dándoles las dimensiones adecuadas.*



Ver → II_10_elementos_flotantes

HACER EJERCICIO → *II_11_Ejercicio_Elementos_Flotantes*