



TEMA 4. CSS Básico

Lenguajes de Marcas
CFGS DAW 1

Autor: Pascual Ligeró

Revisado por:

Diana Bautista Rayo - d.bautistarayo@edu.gva.es

Óscar Villar - or.villarfernandez@edu.gva.es

2022/2023

Licencia



**CC BY-NC-SA 3.0 ES Reconocimiento – NoComercial –
CompartirIgual (by-nc-sa)**

No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

NOTA: Esta es una obra derivada de la original realizada por Pascual Ligeró.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:



Importante



Atención



Interesante

ÍNDICE DE CONTENIDO

1.INTRODUCCIÓN	4
1.1. ¿Qué es CSS3?	4
1.2. Funcionamiento básico de CSS3	4
2. INCLUIR CSS EN UN DOCUMENTO HTML5	6
2.1. Incluir CSS mediante estilos en línea	6
2.2. Incluir CSS en el mismo documento HTML	6
2.3. Uso de hojas de estilos externas	7
3. DEFINICIONES CSS	8
3.1. Partes de un término CSS	8
3.2. Comentarios	8
4. SELECTORES BÁSICOS	8
4.1. Selector universal	9
4.2. Selector de etiqueta	9
4.3. Selector de clase	10
4.4. Selectores de ID	12
4.5. Agrupación de reglas	13
4.6. Herencia	14
4.7. Colisiones de estilos	15
5. MODELO DE CAJA	16
5.1 Fundamentos	16
5.2. Espaciado interno (padding)	17
5.3. Márgenes	18
5.4. Bordes	19
5.5. Redondeado de bordes	19
5.6. Abreviados o shorthands	19
5.7. Sombreado de caja	20
6. PROPIEDADES CSS DE TEXTO	20
6.1. Propiedades 'FONT'	21
6.2. Propiedades de espaciado	21
6.3. Propiedades 'TEXT'	22
7. COLORES CSS	22
8. UNIDADES DE MEDIDA	24
9. INTRODUCCIÓN AL POSICIONAMIENTO: FLOAT	25
10. BIBLIOGRAFÍA	28

UD04. CSS Básico

1.INTRODUCCIÓN

1.1. ¿Qué es CSS3?

CSS3 es un **lenguaje de hojas de estilos** creado para controlar el **aspecto o presentación de los documentos** definidos en HTML5. Sin duda es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.

Separar la definición de los contenidos y la definición de su aspecto presenta numerosas ventajas, ya que obliga a crear documentos HTML5 bien definidos y con significado completo (también llamados "*documentos semánticos*").

Además, mejora la **accesibilidad** del documento, **reduce la complejidad** de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes.

Al crear una página web, se utiliza en primer lugar el lenguaje HTML5 para *marcar* los contenidos, es decir, para designar la función de cada elemento dentro de la página: párrafo, titular, texto destacado, tabla, lista de elementos, etc.

Una vez creados los contenidos, se utiliza el lenguaje CSS3 para definir el aspecto de cada elemento: color, tamaño y tipo de letra del texto, separación horizontal y vertical entre elementos, posición de cada elemento dentro de la página, etc.

1.2. Funcionamiento básico de CSS3

Antes de que se generalizase el uso de CSS3, los diseñadores de páginas web declaraban conjuntos de atributos asociados a las etiquetas de los elementos declarados para modificar su aspecto. También se ha hecho uso de etiquetas (hoy día obsoletas HTML4 o inferiores) tales como la etiqueta <center> para conseguir centrados de elementos.

Otras cuestiones que la filosofía de CSS3 promueve es justamente el utilizar herramientas para el posicionado de elementos (layout) como: float, position, flexbox o grid. Antiguamente se recurría de forma abusiva a float o se construía el layout de una página haciendo una distribución de elementos mediante la estructura tabla. Hoy float y table se siguen utilizando. Float es una herramienta básica para un rápido posicionamiento de un elemento concreto a un área determinada. Por otro lado, la etiqueta table nos sirve para confeccionar estructuras tabulares principalmente con información textual.

El siguiente ejemplo muestra una página HTML con estilos definidos sin utilizar CSS y utilizando atributos (***ojo** el uso de atributos es correcto, pero aquellos referidos a estilo generalmente los sacaremos como propiedades/valor CSS*):

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejemplo de estilos sin CSS</title>
</head>

<body>
  <h1><font color="red" face="Arial" size="5">El Titular principal</font></h1>
  <p><font color="gray" face="Verdana" size="2">Un párrafo.</font></p>
</body>
</html>
```

El ejemplo anterior utiliza la etiqueta `` (obsoleta en potencia, pero apta para componer el siguiente ejemplo pedagógico) con sus atributos `color`, `face` y `size` para definir el color, el tipo y el tamaño de letra de cada elemento de la página.

El problema de utilizar este método para definir el aspecto de los elementos se puede ver claramente con el siguiente ejemplo: si la página tuviera 50 elementos diferentes, habría que insertar 50 etiquetas ``.

Si el sitio web entero se compone de 10.000 páginas diferentes, habría que definir 500.000 etiquetas ``. Como cada etiqueta `` tiene tres atributos, habría que definir 1.5 millones de atributos.

Como el diseño de los sitios web está en constante evolución, es habitual modificar cada cierto tiempo el aspecto de las páginas del sitio. Siguiendo con el ejemplo anterior, cambiar el aspecto del sitio requeriría modificar 500.000 etiquetas y 1.5 millones de atributos.

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, los estilos externos en hojas de reglas CSS.

En esta primera unidad introductoria de CSS veremos entre otras muchas cosas, cómo aprovechar las distintas oportunidades que nos brinda este lenguaje de marcado para poder definir conjuntos de reglas mediante la definición de estilos en línea (en parte hereda los inconvenientes del ejemplo presentado con la etiqueta ``), los estilos internos y los externos en hojas de estilo CSS referenciadas por los documentos HTML que los implementan.

Estos dos últimos enfoques conjuntamente con las características propias de CSS: herencia, cascada y especificidad permiten definir estilos mediante un enfoque modular y totalmente reutilizable.

2. INCLUIR CSS EN UN DOCUMENTO HTML5

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.

2.1. Incluir CSS mediante estilos en línea

Este método, el uso de estilos CSS en línea en documentos HTML **es sin duda el menos mantenible y el menos utilizado**, ya que presenta los mismos problemas que el ejemplo didáctico mostrado al principio de este documento con las etiquetas ``:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejemplo de estilos CSS en línea</title>
</head>

<body>
  <p style="color: purple; ">Un párrafo de texto.</p>
</body>
</html>
```

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 o para fines pedagógicos donde se introduce al alumnado en CSS.

2.2. 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 (sección `<head>`). Ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejemplo de estilos CSS en el propio documento</title>
  <style>
    p { color: black; font-family: Verdana; }
  </style>
</head>

<body>
  <p>Un párrafo de texto.</p>
</body>
</html>
```

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.

2.3. Uso de hojas de estilos externas

En este caso, todos **los estilos CSS se incluyen en un archivo de extensión .css** que las páginas HTML enlazan mediante la etiqueta `<link>`.

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 (si solo queremos declarar una única regla, claro):

```
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`.

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

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejemplo de estilos CSS en un archivo externo</title>
  <link rel="stylesheet" href="/css/estilos.css"/>
</head>

<body>
  <p>Un párrafo de texto.</p>
</body>
</html>
```

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 tres 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`.
- **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.

De todas las formas de incluir CSS en las páginas HTML, esta es sin duda, la más utilizada

3. DEFINICIONES CSS

3.1. Partes de un término CSS

CSS define una serie de términos que permiten describir cada una de las partes que componen los estilos CSS. El siguiente esquema muestra las partes que forman un estilo CSS muy básico:

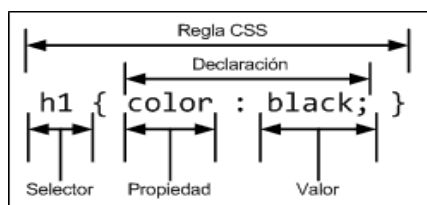


Figura 1.1 Componentes de un estilo CSS básico Los diferentes términos se definen a continuación:

- **Regla:** cada uno de los estilos que componen una hoja de estilos CSS. Cada regla está compuesta de una parte de "*selectores*", un símbolo de "*llave de apertura*" (`{`), otra parte denominada "*declaración*" y por último, un símbolo de "*llave de cierre*" (`}`).
- **Selector:** indica el elemento o elementos HTML a los que se aplica la regla CSS.
- **Declaración:** especifica los estilos que se aplican a los elementos. Está compuesta por una o más propiedades CSS.
- **Propiedad:** característica que se modifica en el elemento seleccionado, como por ejemplo su tamaño de letra, su color de fondo, etc.
- **Valor:** establece el nuevo valor de la característica modificada en el elemento.

Un archivo CSS puede contener un número ilimitado de reglas CSS, cada regla se puede aplicar a varios selectores diferentes y cada declaración puede incluir múltiples pares propiedad/valor.

3.2. Comentarios

CSS permite incluir comentarios entre sus reglas y estilos. Los comentarios son contenidos de texto que el diseñador incluye en el archivo CSS para su propia información y utilidad.

Los navegadores ignoran por completo cualquier comentario de los archivos CSS, por lo que es común utilizarlos para estructurar de forma clara los archivos CSS complejos.

El comienzo de un comentario CSS se indica mediante los caracteres `/*` y el final del comentario se indica mediante `*/`, tal y como se muestra en el siguiente ejemplo:

Aunque los navegadores ignoran los comentarios, su contenido se envía junto con el resto de estilos, por lo que no se debe incluir en ellos ninguna información sensible o confidencial.

La sintaxis de los comentarios CSS es muy diferente a la de los comentarios HTML, por lo que no deben confundirse:

```
/* Este es un comentario
   CSS de varias líneas */
```

```
<!-- Este es un
comentario HTML de varias líneas -->
```

4. SELECTORES BÁSICOS

Para crear diseños web profesionales, es **imprescindible conocer y dominar los selectores de CSS**. Una regla de CSS está formada por una parte llamada "**selector**" y otra parte llamada "**declaración**".

La declaración indica "*qué hay que hacer*" y el selector indica "*a quién hay que hacérselo*". Por lo tanto, los selectores son imprescindibles para aplicar de forma correcta los estilos CSS en una página.

A un mismo elemento HTML se le pueden aplicar varias reglas CSS y cada regla CSS puede aplicarse a un número ilimitado de elementos. En otras palabras, una misma regla puede aplicarse sobre varios selectores y un mismo selector se puede utilizar en varias reglas.

La mayoría de páginas de los sitios web se pueden diseñar utilizando los cinco selectores básicos.

4.1. 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). El selector universal se indica mediante un asterisco (*):

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

A pesar de su sencillez, el universal no se utiliza muy frecuentemente, 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 o servir para resetear estilos por defecto.

4.2. Selector de 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:

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

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

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 (,). Véase el siguiente ejemplo:

```
h1, h2, h3 {  
  color: blue;  
  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: blue;  
  font-family: Arial, Helvetica, sans-serif;  
}  
  
h1 { font-size: 2em; }  
h2 { font-size: 1.5em; }  
h3 { font-size: 1.2em; }
```

4.3. 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?.

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`.

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 red;
  background: violet;
}

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

```
/* Todos los elementos "p" de la página y todos los elementos con
   atributo class="avisos" de la página */
p, .avisos { ... }
```

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`"*.

4.4. 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.

4.5. Agrupación de reglas

Cuando se crean archivos CSS complejos con decenas o cientos de reglas, puede ocurrir que ciertos 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;}
```

ATENCIÓN: No se debe abusar de esta forma de declarar estilos CSS (**y menos este apiñamiento último**). Puntualmente, los profesores de la asignatura os mostraremos ejemplos que sí hagan uso de reglas CSS declaradas en una sola línea por motivos de falta de espacio en el documento donde son presentados los fragmentos de código.

4.6. Herencia

Una de las características principales de CSS es la herencia de los estilos definidos para los elementos. **Cuando se establece el valor de una propiedad CSS en un elemento, sus elementos descendientes heredan de forma automática el valor de esa propiedad.** Si se considera el siguiente ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejemplo de herencia de estilos</title>
  <style>
    body { color: blue; }
  </style>
</head>

<body>
  <h1>Titular de la página</h1>
  <p>Un párrafo de texto no muy largo.</p>
</body>
</html>
```

En el ejemplo anterior, el selector `body` solamente establece el color de la letra para el elemento `<body>`. No obstante, **la propiedad `color` es una de las que se heredan de forma automática**, por lo que todos los elementos descendientes de `<body>` muestran ese mismo color de letra. Por tanto, establecer el color de la letra en el elemento `<body>` de la página implica cambiar el color de letra de todos los elementos de la página.

Aunque la herencia de estilos se aplica automáticamente, **se puede anular su efecto estableciendo de forma explícita otro valor para la propiedad que se hereda**, como se muestra en el siguiente ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
```

```
<title>Ejemplo de herencia de estilos</title>
<style>
  body { font-family: Arial; color: black; }
  h1 { font-family: Verdana; }
  p { color: red; }
</style>
</head>

<body>
  <h1>Titular de la página</h1>
  <p>Un párrafo de texto no muy largo.</p>
</body>
</html>
```

En el ejemplo anterior, se establece en primer lugar el color y tipo de letra del elemento `<body>`, por lo que todos los elementos de la página se mostrarían con ese mismo color y tipo de letra. No obstante, las otras reglas CSS modifican alguno de los estilos heredados.

De esta forma, los elementos `<h1>` de la página se muestran con el tipo de letra `Verdana` establecido por el selector `h1` y se muestran de color negro que es el valor heredado del elemento `<body>`. Igualmente, los elementos `<p>` de la página se muestran del color rojo establecido por el selector `p` y con un tipo de letra `Arial` heredado del elemento `<body>`.

La mayoría de propiedades CSS aplican la herencia de estilos de forma automática. Además, para aquellas propiedades que no se heredan automáticamente, CSS incluye un mecanismo para forzar a que se hereden sus valores, tal y como se verá más adelante.

Por último, aunque la herencia automática de estilos puede parecer complicada, **simplifica en gran medida la creación de hojas de estilos complejas.** Como se ha visto en los ejemplos anteriores, si se quiere establecer por ejemplo la tipografía base de la página, simplemente se debe establecer en el elemento `<body>` de la página y el resto de elementos la heredarán de forma automática.

4.7. Colisiones de estilos

En las hojas de estilos complejas, es habitual que varias reglas CSS se apliquen a un mismo elemento HTML. El problema de estas reglas múltiples es que se pueden dar colisiones como la del siguiente ejemplo:

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

<p>...</p>
```

¿De qué color se muestra el párrafo anterior? CSS tiene un mecanismo de **resolución de colisiones** muy complejo y que tiene en cuenta el **tipo de hoja de estilo** que se trate (de navegador, de usuario o de diseñador), **la importancia de cada regla y lo específico que sea el selector.**

El método seguido por CSS para resolver las colisiones de estilos se muestra a continuación:

1. Determinar todas las declaraciones que se aplican al elemento para el medio CSS seleccionado.
2. Ordenar las declaraciones según su origen (CSS de navegador, de usuario o de diseñador)

y su prioridad (palabra clave `!important`).

3. Ordenar las declaraciones según lo específico que sea el selector. Cuanto más genérico es un selector, menos importancia tienen sus declaraciones.
4. Si después de aplicar las normas anteriores existen dos o más reglas con la misma prioridad, se aplica la que se indicó en último lugar.

ATENCIÓN: No debe hacerse un uso descuidado de la palabra clave `!important`, la cual otorga una mayor prioridad para una regla CSS concreta. El abuso de `!important` puede llevar a nuevas colisiones de reglas CSS y por tanto a un código poco legible y por ende no mantenible.

Hasta que no se expliquen más adelante los conceptos de tipo de hoja de estilo y la prioridad, el mecanismo simplificado que se puede aplicar es el siguiente:

1. Cuanto más específico sea un selector, más importancia tiene su regla asociada.
2. A igual *especificidad*, se considera la última regla indicada.

Como en el ejemplo anterior los dos selectores son idénticos, las dos reglas tienen la misma prioridad y prevalece la que se indicó en último lugar, por lo que el párrafo se muestra de color azul.

En el siguiente ejemplo, la regla CSS que prevalece se decide por lo específico que es cada selector:

```
p { color: red; }
* { color: blue; }
#especial{ color: green;}
<p id="especial">...</p>
```

Al elemento `<p>` se le aplican las tres declaraciones. Como su origen y su importancia es la misma, decide la especificidad del selector.

El selector `*` es el menos específico, ya que se refiere a *"todos los elementos de la página"*. El selector `p` es poco específico porque se refiere a *"todos los párrafos de la página"*.

Por último, el selector `#especial` sólo hace referencia a *"elemento de la página cuyo atributo id sea igual a especial"*.

Como el selector `#especial` es el más específico, su declaración es la que se tiene en cuenta y por tanto el párrafo se muestra de color verde.

5. MODELO DE CAJA

5.1 Fundamentos

ATENCIÓN: es importante repasar las características de los elementos de bloque, línea y inline-block para un mayor aprovechamiento de esta sección.

Podemos afirmar con rotundidad que todos los elementos declarados en nuestra página web obedecen a una estructura rectangular con forma de caja. El estudio de sus dimensiones se conoce como modelo de caja y este implica el espaciado interno y externo de un elemento, su borde y contenido.

Podemos inspeccionar un elemento con las herramientas del desarrollador y ver estas cuatro propiedades referidas a dimensionamiento mediante un gráfico como el que se aquí se adjunta.

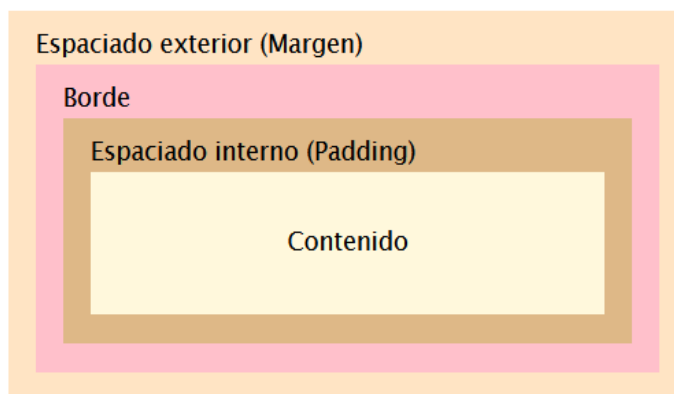


Fig 1.2 Gráfico del modelo de caja.

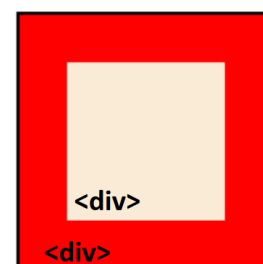
Veamos un caso de estudio del modelo de caja para un elemento división o `<div>`:

Podemos generar un `<div>` (o cualquier otro elemento) y estudiar sus propiedades CSS referidas al modelo de caja mediante las herramientas de desarrollador incluidas en el navegador.

En el caso de los `<div>` y por ser un elemento de bloque, deberemos añadir internamente un contenido o declararle unas dimensiones alto/ancha (de lo contrario este no producirá nunca un resultado visual).

Optamos por declarar un alto y ancho y a continuación, declaramos también un borde. Si nos fijamos, este borde recién definido (por pequeño que sea su trazo) provoca un incremento en la anchura y altura total del elemento, pudiendo producir un empuje de elementos.

5.2. Espaciado interno (padding)



Nuevamente, si declaramos un espaciado interno (o padding) este pudiera provocar un mayor empuje de elementos en su periferia. El espaciado interno funciona dejando una separación entre el contenido de un elemento y el posible borde declarado. Un caso especial pudiera ser aquel en el que no se hayan declarado unas dimensiones, pero si un padding interno. En este caso se generará un espaciado interno en función del espaciado definido y estas serían las dimensiones del elemento. Generalmente **declaramos padding a un contenedor para conseguir una separación adicional entre el límite interno del borde de un elemento y su contenido.**

Fig 1.3 Definición de padding interno

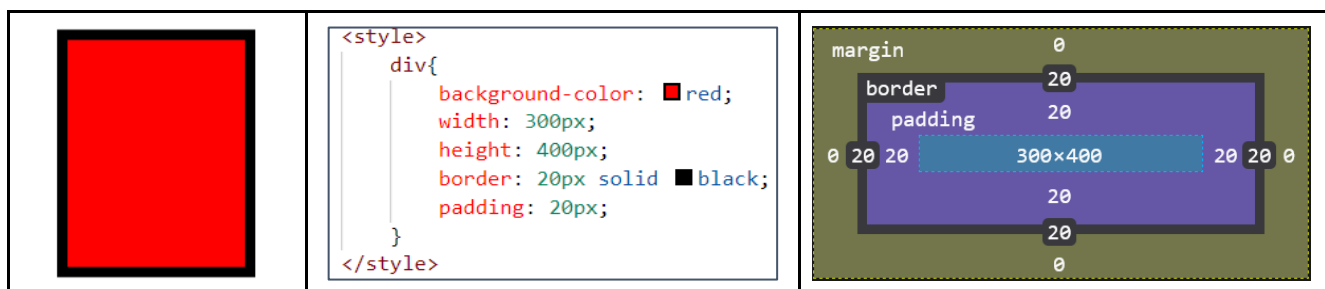


Fig 1.4 Definición de alto/ancho, borde y padding de un <div> y aumento de tamaño.

Dado que la declaración de dimensiones, espaciados internos y bordes es muy común en CSS, se dispone de la opción de **fusionar estas tres medidas**, de tal manera que se respete aquellos valores definidos en el alto/ancho de un elemento. Una vez aplicada la propiedad que presentamos a continuación, **el navegador recalculará automáticamente** el tamaño de estas tres medidas y las ajustará de tal manera que el tamaño final total se corresponda con las dimensiones inicialmente definidas para width y height. La propiedad generalmente es aplicada en el selector universal (para todas las cajas) y es:

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

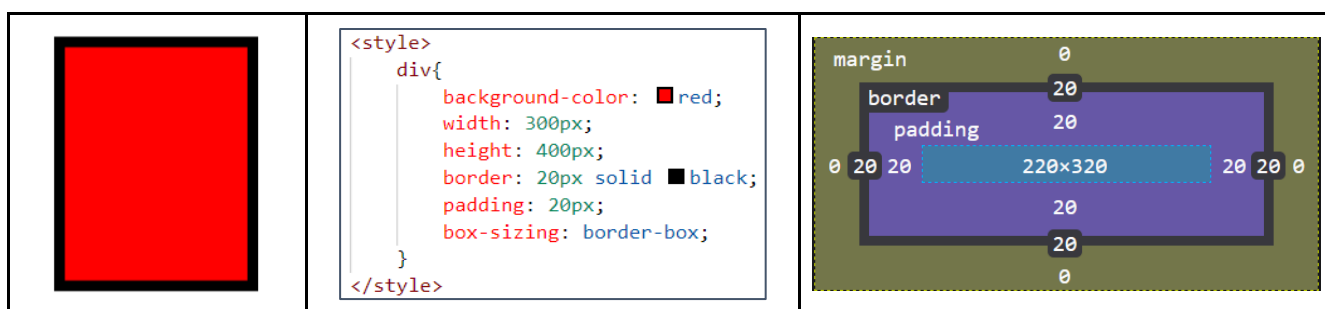


Fig 1.5 Definición de alto/ancho y borde de un <div> junto a la propiedad box-sizing.

5.3. Márgenes

El espaciado exterior (o margin) es declarado como **propiedad de un elemento** para provocar la separación con sus elementos adyacentes o con respecto a elementos que ejercen la función de contenedor.

La propiedad margin es un abreviado o shorthand de: margin-top, margin-right, margin-bottom, margin-left.

También sucede lo mismo con padding o border y sus respectivas concreciones para poder declarar propiedades CSS sólo a uno de los 4 lados de la caja.

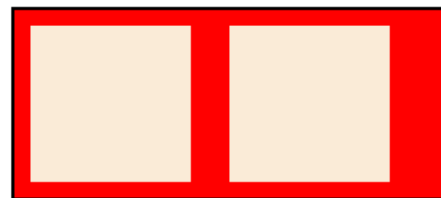


Fig 1.6 Definición de margen derecho en el elemento izquierdo.

Estas cuatro propiedades de las que se compone el modelo de caja admiten una gran variedad de unidades de medida entre las que destacaremos las absolutas (píxeles) y las relativas (em, rem, %, vw y vh), las cuales serán objeto de estudio en el apartado correspondiente de esta unidad.

5.4. Bordes

La definición de un borde en CSS conlleva especificación de su grosor, el tipo de línea (lisa, punteada, rayada, doblemente rayada, etc) y un color. Un ejemplo de declaración de border sería:

```
border: 3px dashed steelblue;
```

El borde comienza a dibujarse desde el momento que finaliza el espaciado interno (padding). Cabe recordar el posible empuje que puede producir su declaración en un elemento y la solución presentada anteriormente con la propiedad **box-sizing: border-box**.

Un borde que no produce empuje alguno y que pudiera servir como guía visual en nuestros desarrollos HTML5/CSS3 es outline. Este toma la estructura y forma de un borde, pero no participa en el modelo de caja. Puede utilizarse de igual modo para efectuar resaltados puntuales de elementos en función de la interacción del usuario con nuestra página.

```
outline: 1px solid red;
```

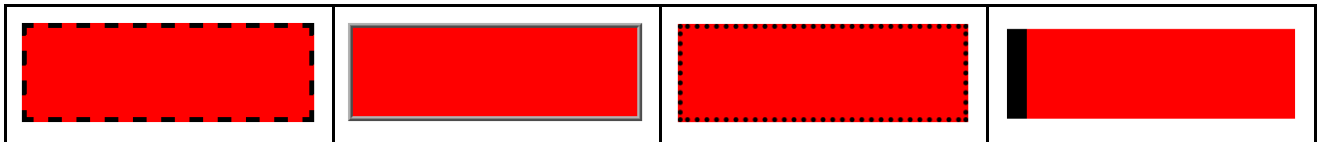


Fig 1.7 Diferentes declaraciones de bordes: dashed, ridge, dotted y un solo borde izquierdo sólido.

5.5. Redondeado de bordes

Podemos conseguir un redondeado en cada uno de los cuatro bordes de una caja, o bien, utilizar el abreviado border-radius. El fundamento de esta propiedad es justamente definir el radio de la circunferencia que dibujaría un borde en particular. Dicho radio se puede especificar, haciendo uso de cualquiera de las unidades de medida disponibles.

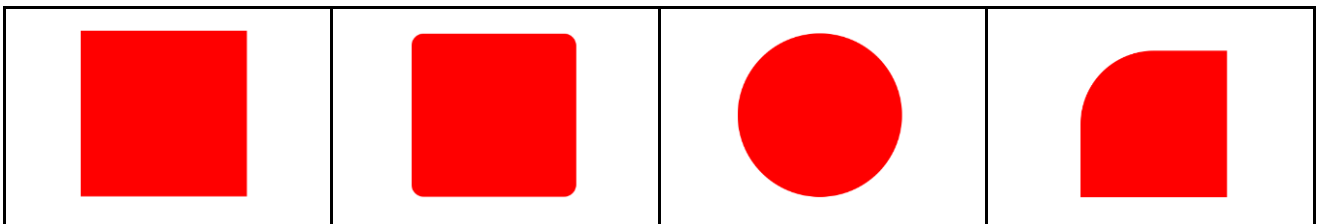


Fig 1.8 Diferentes bordes redondeados aplicados: sin declarar, 4px, al 50% y uso de border-top-left-radius.

5.6. Abreviados o shorthands

Ya han sido presentados los abreviados o “shorthands”: padding, margin, border y border-radius.

Los abreviados tienen la capacidad de poder definir los cuatro lados (o bordes) de la caja de un elemento mediante la definición de 1, 2, 3 o los 4 valores que corresponden a cada uno de los bordes/lados de un elemento. De forma complementaria, podemos definir una única propiedad para un solo lado mediante la especificación que acompañará la propiedad junto a **top**, **bottom**, **left** o **right**.

Si regresamos a la recién presentada propiedad `border-radius` podemos:

- Declarar un único borde:
 - `border-top-left-radius: 4px;` ó
 - `border-radius: 4px 0 0 0;`
- Declarar los cuatro bordes en base a dos medidas (esquina superior /inferior izquierda corresponderán a la primera medida declarada):
 - `border-radius: 4px 2px;`
- Declarar los cuatro bordes en base a tres medidas (esquina superior /inferior izquierda corresponderán a la primera y segunda medidas declaradas y la superior e inferior derecha a la segunda medida):
 - `border-radius: 4px 2px 3px;`
- Declarar las cuatro medidas mediante el abreviado o shorthand:
 - `border-radius: 4px 2px 3px 1px;`
 - `border-radius: 4px;`

5.7. Sombreado de caja

Los sombreados en CSS (en su versión más simple) suelen ser “copias desplazadas” de los elementos. Para lograr una sombra elegante deberemos no utilizar anchos de sombreado excesivamente ostentosos y difuminados (propiedad opcional) sutiles. Igualmente deberemos replantearnos utilizar colores muy intensos.

Podemos definir un sombreado de caja mediante la propiedad CSS `box-shadow`.

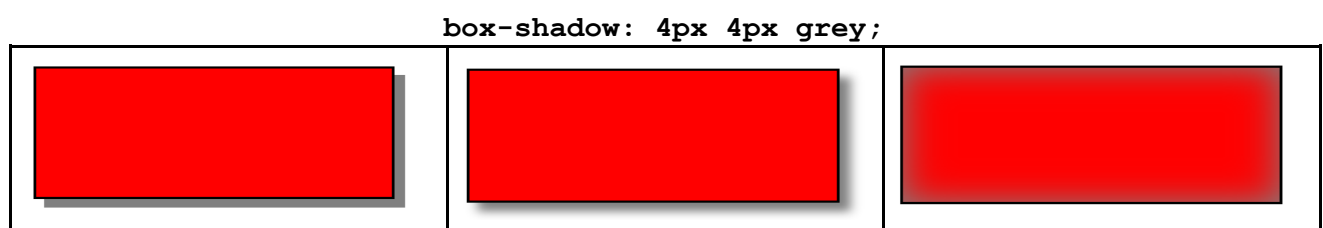


Fig 1.8 Diferentes sombras de caja aplicadas: simple, con difuminado y sombreado interno.

El sombreado puede definirse interno mediante una propiedad opcional llamada **`inset`**. El efecto de la última figura presentada en el gráfico anterior puede lograrse con la regla CSS:

`box-shadow: 0 0 28px grey inset;`

Como anécdota, el sombreado de caja puede explotarse de forma excepcional logrando múltiples capas y pudiendo conseguir el efecto multi-borde. Como efecto que es, nunca mediante sombras declaradas empujaremos accidentalmente el resto de elementos declarados en nuestra página.

`box-shadow:`
`0 0 0 2px ■ #000,`
`0 0 0 3px ■ #999,`
`0 0 0 18px ■ #000;`

6. PROPIEDADES CSS DE TEXTO

6.1. Propiedades 'FONT'

Podemos definir la tipografía textual mediante la propiedad CSS **font-family**. Esta nos permite declarar la fuente que deberá aplicarse a un conjunto de elementos (o elemento único) en el visor del navegador de nuestro usuario. Es recomendable usar un sistema "fallback" consistente en declarar varias fuentes por si la primera (o única) opción declarada fuese incompatible con el navegador de quien visiona nuestra web.

```
p { font-family: "Times New Roman", Times, serif; }
```

Sin duda ampliamos las posibilidades con el tipo de fuente a utilizar si recurrimos a la importación de fuentes externas.

En este aspecto tenemos dos opciones posibles: referenciar la fuente mediante el enlazado con la etiqueta **<link>** o mediante la anotación **@import** en el propio espacio de declaración de reglas CSS.

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

```
<style>
@import url('https://fonts.googleapis.com/css2?family=Syne+Mono&display=swap');
</style>
```

```
font-family: 'Syne Mono', monospace;
```

De una manera u otra, finalmente recurrimos a esta fuente con la propiedad **font-family**.

La siguiente propiedad a presentar es **font-weight**, la cual permite alterar el grosor de la tipografía empleada. Disponemos de los valores textuales: **lighter**, **light**, **normal**, **bold**, **bolder** y de los valores numéricos contenidos en el rango de las centenas: **100**, **200**, **400** (valor por defecto), etc.

Conseguimos el efecto itálica mediante la propiedad **font-style** y el valor **italic**. Su uso generalmente se restringe a ese valor.

Aneecdótico es sin duda el uso de **font-variant** y el valor **small-caps**, el cual permite definir un texto totalmente en mayúsculas, el cual resalta las mayúsculas originarias con un mayor tamaño.

HOLA ALUMNOS DE LENGUAJES DE MARCAS!

6.2. Propiedades de espaciado

Hablemos ahora de la alineación vertical. Podemos lograr definir superíndices y subíndices mediante las etiquetas HTML **<sup>** y **<sub>**. No obstante, disponemos de una propiedad CSS que permite cómodamente reemplazar las dichas etiquetas referidas a estilos. La propiedad en este caso es: **vertical-align**. La alineación vertical del texto mediante esta regla se aplica en base al texto es su posición original. A continuación se presenta un ejemplo de declaración de esta regla CSS.

```
<p>Hola alumnos de.. <span id="superindice">Lenguajes de Marcas!</span></p>
<p>Hola alumnos de.. <span id="subindice">Lenguajes de Marcas!</span></p>
#superindice { vertical-align: super; }
#subindice { vertical-align: sub; }
```

Hola alumnos de.. Lenguajes de Marcas!

Hola alumnos de.. Lenguajes de Marcas!

Fig 1.9 Definición de superíndices y subíndices mediante CSS.

El interlineado puede lograrse sin mucho esfuerzo mediante **line-height** acompañado de un valor en cualquiera de las unidades de medida asociadas a CSS. Dicha propiedad tiene sentido para un párrafo, una lista o inclusive otro conjunto de elementos declarados uno a continuación del otro en líneas distintas.

Finalmente presentar la posibilidad de espaciar palabras y letras. Por un lado, **word-spacing** permite concatenar visualmente las palabras otorgando un mayor o menor espaciado horizontal. De igual modo, **letter-spacing** permite lograr el mismo efecto a nivel de letras. Estas propiedades pueden ser interesantes, más aún si las trabajamos debidamente para lograr una mayor legibilidad en nuestros textos.

Como curiosidad, ambas propiedades: **word-spacing** y **letter-spacing** permiten declarar espaciados mediante valores negativos (lo cual originará un apiñamiento excesivo).

6.3. Propiedades 'TEXT'

Es importante recordar que lograremos el posicionamiento horizontal deseado del texto mediante la propiedad CSS **text-align**. Con gran frecuencia precisamos centrar el texto y nuevamente esto lo logramos mediante esta propiedad y el valor center. También es posible lograr un justificado a izquierdas o derechas o un justificado total (de extremo izquierdo al extremo derecho). W3C recomienda utilizar para el justificado total añadir una segunda propiedad-valor CSS: **text-justify: inter-word** para poder lograr un efecto más natural en el justificado.

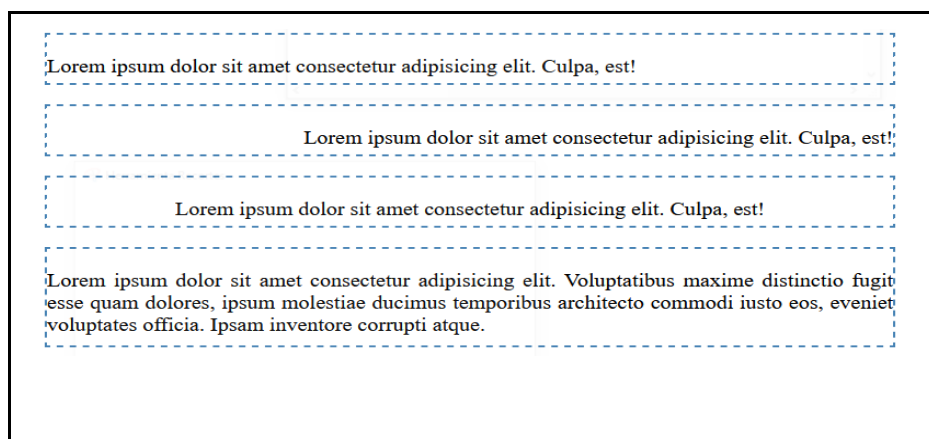


Fig 2.0 Diferentes justificados de texto mediante los valores: left, right, center, justify.

Es posible alterar las propiedades de los enlaces y quitar su subrayado mediante **text-decoration**, así como también es posible añadirlo a elementos textuales que no lo tenían, realizar tachados, entre otras opciones. Los valores posibles en este caso son: **overline**, **line-through**, **underline**, **none** o inclusive aplicar varios de estos valores combinados!.

¿Quién no ha variado las propiedades de pigmentación de la fuente a estas alturas del curso? La propiedad **color** permite definir mediante los formatos disponibles de definición de color en CSS un tipo de coloración específico. Estos formatos se estudian con detenimiento en un apartado posterior de esta unidad.

text-transform nos permite formatear el texto en minúsculas, mayúsculas o el efecto capitalize (todas las palabras en letras minúsculas salvo la primera letra de cada). Los valores para **text-transform** son: **uppercase**, **lowercase**, **normal** y **capitalize**.

Finalmente **text-shadow** nos permite definir sombreado a elementos textuales. El modo de operar es análogo al sombreado descrito para el modelo de caja. Por lo que se recomienda repasar su forma de declaración.

```
text-shadow: 2px 2px 5px blue;
```

7. COLORES CSS

CSS3 cuenta con más de **140 distintos nombres de colores** para poder adjuntar como valor cuando alteramos alguna de las múltiples propiedades asociadas. Algunos ejemplos son: `tomato`, `green`, `ivory`..

No es factible recordar todos estos nombres, por lo que VSCode y aquel editor que cuente con plugins de soporte para CSS debería ayudarnos al respecto (siempre nos quedará W3Schools ;)).

Dejando de lado la declaración de colores mediante un valor textual, tenemos varias opciones.

RGB (red, green , blue) se trata de una composición de color en base a los tres colores primarios rojo, verde y azul. Para cada uno de los componentes que participan, es posible declarar un valor decimal en el rango [0-255]. Esto es así y como podemos fácilmente adivinar, estamos ante un abanico posible de colores de hasta $(2^{32})^3$.

Existe una variante de RGB llamada RGBA, el cual permite definir un cuarto valor llamado alpha, el cual refiere a la opacidad del color declarado, el cual debe declararse en el rango [0-1] (siendo 1 totalmente opaco). El principio para RGBA no varía respecto a la declaración de los colores primarios.

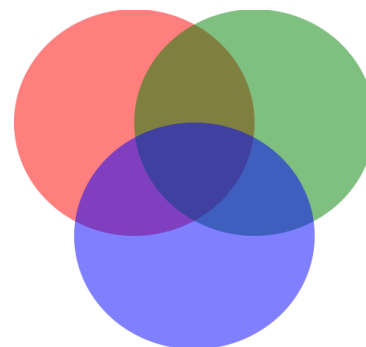


Fig 2.1 Colores primarios.

```
border: 1px dashed rgb(255,0,0);
```

```
border: 1px dashed rgb(255,0,0,.5);
```

Fig 2.2 Bordes rojos declarados en formato RGB y RGBA.

El **formato hexadecimal** es posible para declarar un color RGB. Procedemos a su declaración anteponiendo un carácter almohadilla o hash frente a 3 parejas de caracteres hexadecimales, cada pareja refiriéndose a un valor primario red, green o blue. No hay diferencia respecto al rango de representación posible. Únicamente varía la forma de declarar la información. Como curiosidad, es posible omitir para una pareja de caracteres asociada a un primario, uno de los dos caracteres si este se repite.

```
background-color: #ff0000;
```

```
background-color: #f00;
```

Fig 2.2 Equivalencia de colores de fondo declarados en formato hexadecimal.

Nos queda presentar **HSL y HSLA**. Estos son acrónimos que provienen de Hue (Tonalidad), Saturation (saturación) y Lightness (Luminosidad). HSLA utiliza un cuarto valor alpha en concepto de opacidad.

Hue se define mediante grados que corresponden a un círculo cromático, el cual su valor 0º corresponde con el rojo más vivo. El resto de valores son porcentuales. Saturation va desde un gris apagado a un máximo de vivacidad para la tonalidad elegida. Lightness va desde una oscuridad total a un brillo máximo. El valor alpha actúa como su homónimo en el formato RGBA.

```
color: hsl(0,100,50);    color: hsla(0,100,50,.5);
```

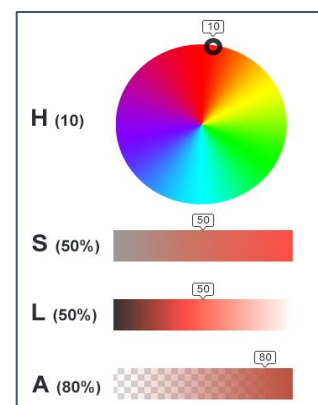


Fig 2.3 HSL / HSLA

8. UNIDADES DE MEDIDA

ATENCIÓN: es importante repasar las características de los elementos de bloque, línea y inline-block para un mayor aprovechamiento de esta sección.

Podemos clasificar las unidades de medida que pone CSS3 a disposición del desarrollador web en: **unidades de medida absoluta y relativa**. Las medidas absolutas nos permitirán definir la estructura fija de un determinado elemento o propiedad del mismo. Las relativas, en cambio permiten un crecimiento dinámico en función de otros elementos referencia o eventos fruto de la navegación del usuario. La ausencia de valor en una medida (concretamente el valor 0) no requiere de la declaración de unidad de medida alguna que lo acompañe.

La unidad de medida absoluta más recurrida sin duda es el **píxel**. Esta unidad es relativa al visor del dispositivo. Concretamente un píxel se corresponde a un punto o píxel de la pantalla física. Podemos averiguar que cantidad de pixels tiene una pantalla de un dispositivo concreto y configurar las herramientas de desarrollo del navegador para que muestre nuestro actual desarrollo desde la perspectiva de de la pantalla de dicho dispositivo. Podemos pues tomar como referencia el tamaño en píxeles de un dispositivo concreto o desarrollar para múltiples dispositivos a la vez.

El píxel es una medida a la que se suele recurrir al comienzo de nuestros desarrollos y ciertamente no funciona nada mal. Simplemente debemos recordar que un elemento acomodado estáticamente mediante una definición de dimensiones basada en píxeles, no permitirá un crecimiento dinámico de ciertas áreas que nos interese agrandar si algún evento sucede (agrandamiento de pantalla o inclusión de nuevo contenido en nuestra página).

El píxel es igualmente la unidad clave cuando queremos escalar una imagen. Toda imagen lleva consigo un tamaño implícito: su tamaño original en píxeles. Lograremos escalados perfectos (sin deformidad) si únicamente alteramos una de las dos propiedades `width` o `height` y dejamos que la otra dimensión la recalcule el navegador.

Del lado de las medidas relativas tenemos como a una gran recurrida: el **porcentaje**. El porcentaje sirve para definir las propiedades de dimensionado de los elementos tomando como referencia el alto o ancho del elemento padre que los contiene. Así de simple, pero potente.

Si tomamos la parte visible de nuestro navegador sin contar los bordes propios de la ventana de navegación ni el scroll, tenemos definido en estos momentos el área del visor o **viewport**. Las unidades **viewportwidth** y **viewportheight** (`vw` y `vh` respectivamente) equivalen a un 1% de ese ancho total del visor y un 1% del alto total. Con esta premisa podemos definir propiedades de tamaño de texto y dimensiones para elementos, los cuales agrandaran su tamaño en función de cuán grande sea en un momento determinado el visor de la página.

Nos quedan por presentar dos medidas relativas que contribuyen al igual que el porcentaje y las medidas viewport a lograr diseños web escalables.

REM es una unidad de medida que **toma como referencia el tamaño de texto definido en el elemento raíz (<html>)**. Si en `<html>` tenemos definido un tamaño de fuente de `16px`: `1rem` en mi desarrollo web equivaldrá a estos 16 píxeles, `0.5rem` a `8px` y así sucesivamente. Puede ser buena idea aplicar múltiplos de 4, 8 y 16 si decidimos trabajar con el valor `16px` en la raíz para lograr una estética más armoniosa.

EM es la última unidad relativa a presentar en esta unidad. Ésta **toma como referencia el tamaño de fuente del elemento donde se declara una medida y se emplea dicha unidad**. Esto tiene sentido cuando analizamos con calma el mecanismo de herencia que posee CSS y pensamos que un elemento determinado puede heredar o redefinir su tamaño de fuente. Esto permitirá disponer de una medida que podremos utilizar para

crear áreas o elementos de mayor o tamaño con un escalado en función de otros elementos que se toman como referencia.

9. INTRODUCCIÓN AL POSICIONAMIENTO: FLOAT

ATENCIÓN: es importante repasar las características de los elementos de bloque, línea y inline-block para un mayor aprovechamiento de esta sección.

Si os preguntan decid que sí. Que todavía se usa y que está previsto que se siga usando por muchos años más.

Float es una propiedad CSS de posicionamiento básica. Permite (con relativo poco esfuerzo) posicionar un elemento a la izquierda o derecha de una determinada área. Para poder usarla se recomienda tener bien asentadas las características propias de los elementos en bloque, línea y la híbrida: inline-block.

Float a priori puede parecer muy fácil de usar pero en la práctica esconde una complejidad a tener en cuenta. Esto es así, dado que esta herramienta fue diseñada hace mucho tiempo y por tanto no cuenta con una estructura tan versátil como las últimas contribuciones en materia de posicionamiento.

Un elemento flotado a izquierdas (o derechas) pierde su posición original (se dice que escapa del flujo HTML). Podemos imaginar una capa en el eje Z donde residen los elementos flotados. Aquellos elementos declarados en el flujo HTML a continuación del elemento recientemente flotado ocuparan su espacio.

Estudiemos ahora el siguiente caso. Si declaramos dos elementos que flotan a derechas, un elemento que no tienen la propiedad de flotabilidad (y por tanto ocupa el espacio de aquellos que sí flotan) se produce un salto de línea en ocasiones inesperado ¿Por qué?.

Posiblemente podríamos estar ante un caso en el que conviene “reordenar” las sentencias de elementos declaradas en el flujo HTML de tal manera que aquellas que requieren flotar sean las primeras.

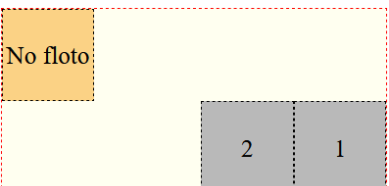
	<pre><div class="externo"> <div class="naranja centered">No floto</div> <div class="interno centered">1</div> <div class="interno centered">2</div> </div></pre>
---	--

Fig 2.4 Elementos flotados a derechas y elemento izquierdo sin flotar/flotado (si reordenar el HTML).

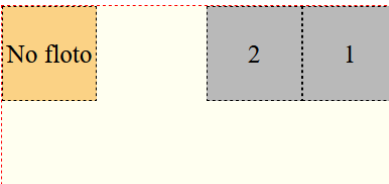
	<pre><div class="externo"> <div class="interno centered">1</div> <div class="interno centered">2</div> <div class="naranja centered">No floto</div> </div></pre>
---	--

Fig 2.5 Reordenación del HTML. Primero flotarían los divs 1 y 2 con clase “interno”.

Nótese que al hacer flotar a derechas dos elementos con un contenido numérico que pretende mostrar un cierto orden al usuario de nuestra web, este es mostrado ordenado al revés. Debemos pues detenernos y pensar en qué orden han flotado los elementos. El primer <div> flotó a la derecha en primer lugar, se posicionó y a continuación se produjo la flotación del segundo elemento.

Podríamos resolver este inconveniente eliminando la propiedad de flotabilidad a derechas para estos dos contenedores internos y creando un nuevo div que “abraze” ambos elementos el cual ahora sí, implemente la regla de flotar a derechas.

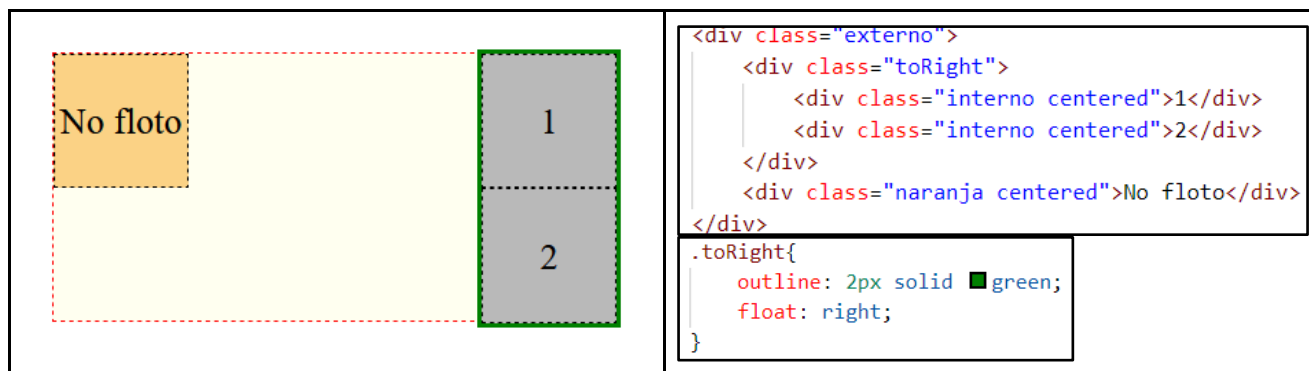


Fig 2.6 Creación de un <div> que flota a derechas junto con sus elementos internos.

Si nuestros elementos son de bloque (como es el caso) faltará que ahora los elementos internos entre sí definan una flotación a izquierdas, la cual respetará su orden en el flujo HTML.

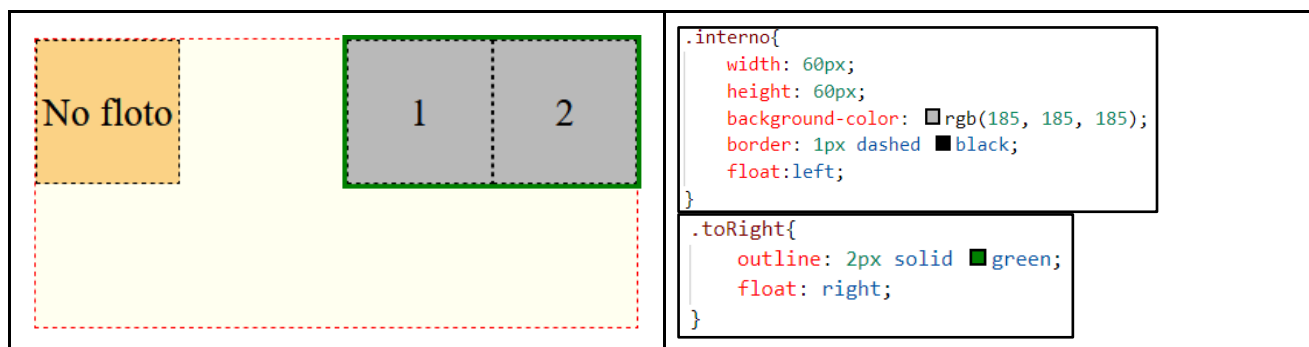


Fig 2.7 Creación de un <div> que flota a derechas junto con sus elementos internos.

Podemos forzar a que un elemento flote a la derecha de un área y que otro flote a la izquierda, tal como se muestra en la Figura 2.3, aplicando propiedades float a todos los elementos!. Quizás esta solución sea más rápida de implementar que la anterior propuesta.

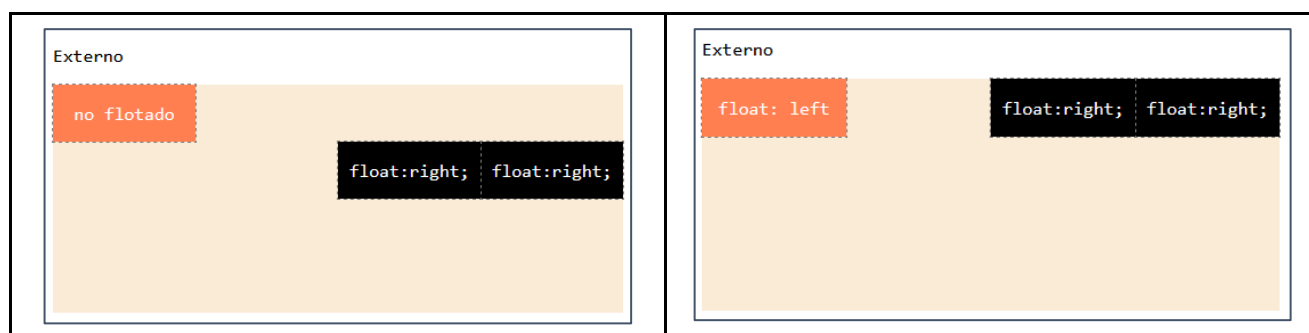


Fig 2.8 Elementos flotados a derechas y elemento izquierdo sin flotar/flotado (contenedor: alto y ancho definidos).

Respecto a los elementos padre o contenedores, si estos únicamente albergan elementos flotados, puede que pierdan la noción del espacio de sus elementos internos “flotantes” y que mengüen su tamaño sin

aparente razón. Si poseen más elementos sin flotar, los no flotados fijarán el tamaño del contenedor. Podemos forzar que un contenedor “abrace” al total de hijos flotados y no flotados si este define la propiedad/valor CSS: **overflow-hidden**.

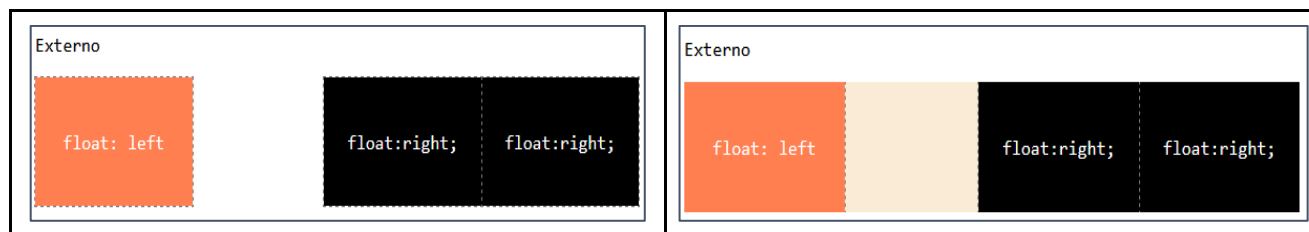



Fig 2.9 Todos los elementos flotados y un contenedor que precisa **overflow:hidden** (contenedor: sin alto y ancho).

Hacer flotar un conjunto de elementos a la izquierda, seguramente causará el efecto deseado tanto en cuanto esperamos que se agrupen con el orden que hemos declarado en las sentencias HTML. Sin embargo, hacer flotar un conjunto de elementos todos a la derecha, puede provocar una ordenación a la inversa de los mismos. Quizás en un caso así, sea buena idea encerrar dichos elementos en un contenedor y hacer flotar dicho contenedor a la derecha..

La propiedad **clear** permite crear un empuje entre un elemento flotado y sus siguientes y así poder seguir posicionando elementos en estado de “flotación”. Generalmente la propiedad clear se utiliza después de haber flotado un elemento. El valor de la propiedad clear ha de coincidir con el valor de la propiedad float del elemento inmediatamente flotado.

Veamos un caso de estudio de un <div> que contiene a su vez cinco contenedores <div> (elementos de bloque).

Ningún elemento “flotado”.	Elemento 1 flotado a la derecha. El resto ocupa su espacio.	Elemento 2 “limpia” el elemento 1 “flotado” a la derecha.
Elementos 2 y 3 “flotan” a izquierdas (4º y 5º ocupan su espacio)	Elemento 4º “limpia” el flotado a izquierdas de 2 y 3.	“Flotamos” el 4º elemento a la derecha

	<p>Para este caso es suficiente el no declarar un espaciado al contenedor, porque cuando este tiene el conjunto de elementos “limpios” de sus respectivos flotados, este toma consciencia del tamaño total de sus elementos.</p> <p>Aun así se recomienda declarar en el contenedor padre la propiedad/valor: overflow-hidden, por sí un olvido accidental descuidase algún elemento en un estado especial.</p>
<p>El 5º elemento “limpia” el “flotado” a derechas del 4º.</p>	

No debéis temer a `float`. No muerde. **Simplemente, no hagáis grandísimos esfuerzos por intentar posicionar un número elevado de elementos en vuestra página en desarrollo con esta herramienta.** Esto está totalmente desaconsejado. Sin embargo, esta herramienta la debéis conocer y saber aplicar en el momento adecuado. Hay mucho código heredado (y actual) que aún recurre a `float` para el posicionamiento de elementos.

10. BIBLIOGRAFÍA

- [1] <http://librosweb.es/libro/css/>
- [2] <https://www.w3schools.com/>