2.3. Hojas de estilo en cascada

Índice

Objetivos

Los objetivos de esta unidad son los siguientes:

- Aplicar estilos CSS a los documentos HTML.
- Conocer las principales reglas CSS.
- Conocer las propiedades de los elementos en CSS.
- Adaptar los elementos de la página a diferentes dispositivos.

Conceptos

¿Qué es una hoja de estilos?

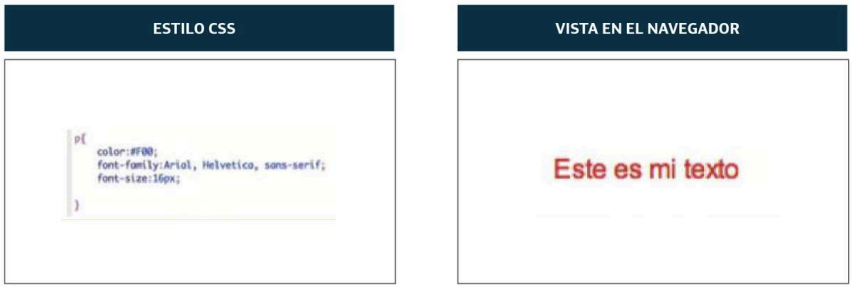
El lenguaje de marcas HTML surgió como una forma de definir un documento, pero nunca para darle formato.

Es en la versión HTML 3.2 cuando comienzan a usarse etiquetas como *<font>* y otras que generan documentos HTML poco estructurados y con demasiadas etiquetas de formato.

Estos documentos eran de alguna forma caóticos y poco claros. No se podía distinguir de forma sencilla la semántica ni la estructura del documento.

La revisión de HTML 4 aportó la creación del etiquetado de estilos para definir, por un lado el formato, y por otro el contenido de un documento, estableciendo que:

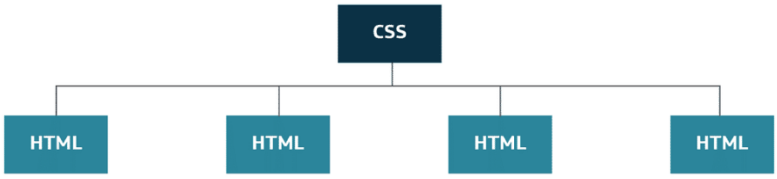
- La página web (documento HTML) solo debe contener información.
- El formato se debe definir en las llamadas hojas de estilo (CSS).
- Las hojas de estilo son estructuras de marcas que definen el formato de los elementos. Se pueden incluir dentro del archivo HTML o fuera de él **enlazando el archivo** .css en la página.
- CSS significa Cascading Style Sheets.
- CSS son los estilos que se van a aplicar en documentos de marcas como HTML.
- CSS ahorra mucho trabajo, ya que los estilos se definen en un solo lugar.



Ejemplo de aplicación de estilos a un párrafo.

Con las hojas de estilo no solo podremos hacer que nuestros documentos se vean del modo que deseemos, además definiremos la estructura de nuestra página, menús, columnas, etc.

Además, hoy en día, debemos hacer que nuestras páginas se adapten a múltiples dispositivos, y CSS será un gran protagonista en esta tarea.



Tipos de CSS

En este módulo trataremos CSS de forma estática, aplicando el estilo a los objetos de nuestro documento, pero es importante conocer que podemos dotar a nuestras hojas de estilos de dinamismo con tecnologías como SASS o LESS.

CSS Estático

Las hojas de estilo estáticas son las que usaremos en este módulo. Los elementos son configurados según unas reglas no cambiantes (estáticas), sin la posibilidad de condicionar su estado.

Características de un CSS estático:

- Cada elemento del estilo tiene sus atributos estáticos, por lo que no pueden modificarse en tiempo de carga o de ejecución.
- Los atributos no pueden ser calculados en tiempo de carga o ejecución.
- Si varios elementos tienen atributos iguales, se definen para cada uno de ellos.
- Para cambiar los estilos se accede al fichero CSS directamente y se cambia el valor del atributo.

Aunque CSS estático, podremos utilizar algunas propiedades como *:hover*, *:focus*, etc., que aunque generan algo de dinamismo al documento, se sigue considerando estático.

CSS Dinámico

Para facilitar la creación de los estilos CSS existen una serie de herramientas que permiten manejar los estilos mediante un pre-procesamiento o en tiempo de ejecución. Con esto se consigue mejorar la estructura y eficiencia de la hoja de estilos.

Para crear este tipo de CSS es necesario usar:

- **Herramientas como Stylus, LESS o SASS**. Estas herramientas se constituyen como un lenguaje de programación que realiza un pre-procesamiento de las hojas de estilos, de manera que antes de ser cargadas en la página se han calculado y definido.

- **Lenguaje JavaScript**. Con JavaScript podemos interactuar con el DOM de la página modificando los estilos al producirse un evento.

¿Cómo aplicamos los estilos a la página?

En este apartado veremos los distintos métodos para asociar estilos CSS a las páginas HTML.

Para aplicar estilos a nuestras páginas podemos hacerlo de tres formas:

- En la propia página con una etiqueta *<style>*.
- En una etiqueta por medio del atributo *style*.
- En una hoja de estilos independiente. Este es el método recomendado y el que utilizaremos con más frecuencia durante el curso.

Aplicar estilos dentro de la página con una etiqueta <style>

```
<head>
 <meta charset="UTF-8">
  <title>Mi primera web</title>
  <style type="text/css">
      p {
          color: blue;
          font-family: 'Comic sans MS',Verdana;
          font-size: 18px;
      }
 </style>
</head>
```

La etiqueta HTML *<style>* permite encerrar especificaciones de estilo en formato CSS. Este sistema no es recomendable a nivel de productividad, pues impide la reutilización del código y dificulta las tareas de mantenimiento. Además es penalizado por el posicionamiento SEO.

Se suele usar para piezas de diseño como boletines informativos (*newsletter*) o bocetos.

Estilos en línea usando atributo *style* de las etiquetas HTML

```
<p style='background-color: aqua; color: blue'>Esto es un párrafo</p>
```

Este sistema **tampoco es recomendable** porque sobrecarga y complica el contenido de las etiquetas HTML del documento innecesariamente.

Estilos definidos en una hoja de estilo independiente

Este **sistema es el más recomendable** y consiste en la creación de un **fichero independiente con extensión CSS**, que será denominado "**archivo de hoja de estilo**". Siguiendo con la idea de mantener cierto orden en la estructura de la web, las hojas de estilo deberían guardarse en una carpeta específica.

Para enlazar el archivo HTML con el de la hoja de estilos *.css*, usaremos la etiqueta ***link***, indicando **la relación** que tiene con este archivo (***rel="stylesheet"***), el **tipo** de documento (***type="text/css"***) y la **ruta** en la que se encuentra (***href="estilos/estilo.css***").

```
<head>
<meta charset="UTF-8">
<title>Mi primera web</title>
<link rel="stylesheet" type="text/css" href="estilos/estilo.css">
</head>
```

Para comenzar a crear nuestra hoja de estilo generaremos un fichero con el editor que se haya escogido y escribiremos el conjunto de reglas CSS que queramos aplicar al documento.

```
p {
font-family: Arial, Helvetica, sans-serif;
font-size: 17px;
text-align: justify;
text-indent: 40px;
margin: 10px 10px 10px 10px;
color:#red;
}
.verde {
color:green;
}
```

Sintaxis CSS

La sintaxis de un estilo es muy simple e intuitiva. Su estructura consta de una serie de reglas que describen la forma en la que se visualiza cada uno de los elementos.

El Selector
Hace referencia a uno o varios elementos HTML; en el ejemplo afectará a las cabeceras de nivel 1.

Propiedad
Con la propiedad se identifica el elemento que se va a modificar. Por ejemplo el color, la alineación, el margen, etc.

La Declaración
Sirve para indicar cómo se va a ver el selector; es un conjunto de propiedades a las que se les asigna un valor.

Valor
En el valor se define cómo es la propiedad. En este ejemplo estamos "alineando el Título 1 a la derecha".

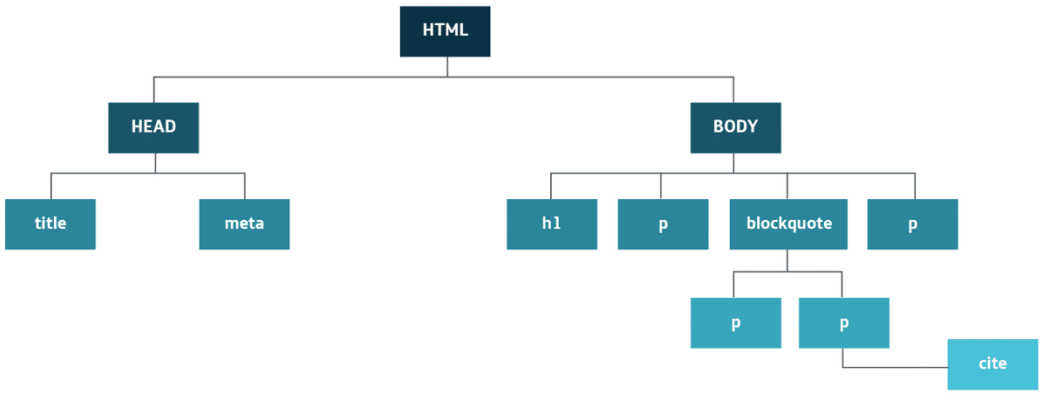
Cada propiedad acepta un tipo de valor, pudiendo ser un número, el valor hexadecimal de un color, etc.

La herencia

La herencia o *inheritance* es una de las características principales de los estilos.

Para facilitar la comprensión del concepto de herencia debemos entender que un documento HTML se puede representar como un árbol genealógico.

```
h1 {
  font: 1em/1.5 Verdana, Helvetica, sans-serif;
}
p {
  font: 1em/1.5 Verdana, Helvetica, sans-serif;
}
Blockquote {
  font: 1em/1.5 Verdana, Helvetica, sans-serif;
}
Cite {
  font: 1em/1.5 Verdana, Helvetica, sans-serif;
}
```



Algunas de las propiedades de las hojas de estilo se propagan por los descendientes de los elementos. Por ejemplo, en un documento sencillo de estilos podríamos definir las características del documento así:

```
body {
  font: 1em/1.5 Verdana, Helvetica, sans-serif;
}
```

En este ejemplo establecemos a nivel de *body* que la fuente tiene un tamaño y una tipografía específica, afectando a todos los elementos de texto.

Aun así, **no todas las propiedades se heredan**. Una de las formas de saber qué propiedades son heredables y cuáles no es leer la especificación de W3C, que nos proporciona la información necesaria para saber su "*inherited*" con un "yes" o con un "no".

Herencias no deseadas

Algunas veces este tipo de comportamiento hace que heredemos propiedades no deseadas, ya que con la cascada unas anulan a otras. Para solucionar este problema, existe el atributo *!important* que fuerza este cambio si el explorador no aplica la nueva regla.

```
ul{
 margin: 10px !important;
}
```

Aunque este atributo es muy útil y nos soluciona los problemas de herencias, su utilización no es muy recomendable.

La cascada

El mecanismo más importante de CSS es la cascada (*Cascading Style Sheets*).

La cascada nos sirve para saber qué ocurre si dos reglas diferentes asignan la misma propiedad al mismo elemento.

Se encarga de controlar el resultado final cuando surgen conflictos sobre qué estilos se aplican y en qué orden de preferencia en un elemento.

Para saber cómo se aplican las reglas de la cascada debemos tener en cuenta tres aspectos importantes:

Especificidad

Se produce cuando dos declaraciones tienen la misma importancia, bien porque ambas cuenten con *!important*, bien porque ambas carezcan de él.

La especificidad de los elementos se realiza con un cálculo, que se puede representar como cuatro valores separados por comas. Para ello se tienen en cuenta los siguientes aspectos:

1. El número de estilos en línea aplicados al elemento a través del atributo *style* en el marcado.
2. El número de selectores de la regla de estilo que son un identificador (*id*).
3. El número de selectores que son una clase (*class*), un selector de atributo o una pseudoclase.
4. El número de selectores que son un elemento o un pseudoelemento.

Importancia

Es posible que una de las declaraciones que utilicemos en nuestros documentos sea lo bastante importante para que se aplique, independientemente de que otra regla pueda afectarla.

Para eso utilizamos *!important*, que indica la importancia de la declaración que queremos mantener pase lo que pase.

Debemos incluirla antes del punto y coma final para que surja efecto.

Orden

El orden básicamente hace que, si un elemento tiene dos reglas con el mismo peso, tenga preferencia la última que ha sido especificada.

Unidades en CSS

Para establecer la medida de espacios, tamaños o márgenes, en CSS tenemos diferentes unidades de medida.

Todas las medidas se indican como un **valor numérico entero o decimal seguido de una unidad de medida.**

```
p {
   font-size:18px;
}
```

Tipos de unidades

Existen dos tipos de unidades, absolutas y relativas.

Unidades absolutas

- ***in***, pulgadas ("*inches*"). Una pulgada son 2.54 centímetros.
- ***cm***, centímetros.
- ***mm***, milímetros.
- ***pt***, puntos. Un punto equivale a unos 0.35 milímetros.
- ***px***, píxel. Relativa respecto de la resolución de la pantalla del dispositivo en el que se visualiza la página HTML.
- ***pc***, picas. Una pica equivale a 12 puntos, es decir, a unos 4.23 milímetros.

Unidades relativas

- ***em***, relativa respecto del tamaño de letra del elemento.
- ***%*** , el tanto por cien relativo al tamaño de la ventana donde se está visualizando el documento.

Reglas en CSS

Principales propiedades

Las propiedades nos ayudan a especificar qué se va a modificar. Hay un gran número de propiedades, vamos a ir viendo las más importantes.

Referentes al texto

font-family

Tipo de fuente.
Valores: Arial; Helvetiva; sans-serif; serif; Times New Roman; Times; Verdana; Georgia; Geneva; Courier; Corier New ...

```
selector {
   font-family: 'arial';
}
```

Las tipografías utilizadas deben ser las incluidas en los sistemas operativos, pues usará el instalado en el cliente. En CSS3 existe la posibilidad de usar tipos de letra personalizados, aunque nos penalizará en el peso de la página.

font-size

Tamaño de la fuente.
Valores:

- Unidades.
- Porcentaje.

```
selector {
  font-size:19px;
}
```

font-style

Estilo de la fuente.
Valores:

- *italic*;
- *normal*;

font

Permite definir en una única propiedad el tamaño de la letra, el espacio entre renglones y la familia tipográfica. Se integra en una línea siguiendo:

propiedad: Tamaño Texto / Interlineado Fuente;

```
p {
font: 1em/1.5em Verdana, Helvetica, sans-serif;
}
```

font-variant

Permite modificar cómo se visualiza el texto.
Valores:

- *normal*;
- *small-caps*; (versalitas)

font-weight

Aplica el formato de "negrita" al texto.
Valores:

- valores absolutos: 100, 200, 300 ... 900
- *bold*;
- *normal*;

```
selector {
 font-weight:bold;
}
```

text-decoration

Permite definir diferentes elementos para aplicar al texto, como subrayado o tachado.
Valores:

- *blinks*; (parpadeante)
- *line-through*; (tachado)
- *underline*; (subrayado)
- *overline*; (sobrerayado)

text-transform

Transformación del texto.
Valores:

- *capitalice*; (convierte la primera letra de cada palabra a mayúsculas).
- *uppercase*; (todo en mayúsculas)
- *lowercase*; (todo en minúsculas)

color

Color del elemento.
Valores: nombre del color, valor del color.

En este ejemplo puedes ver aplicadas estas propiedades.

Referentes a la alineación

letter-spacing

Espacio entre letras.
Valores: unidades.

word-spacing

Espacio entre palabras.
Valores: unidades.

line-height

Espacio entre renglones.
Valores: unidades, porcentaje.

text-align

Alineación horizontal del texto.
Valores:

- *left*;
- *right*;
- *center*;
- *justify*;

text-indent

Sangrado.
Valores: unidades, porcentaje.

En este ejemplo puedes ver aplicadas estas propiedades.

Referente a contenedores

border

Para el borde de los elementos, en la misma regla podemos indicar el grosor, el color y el tipo.

```
div {
  border: 1px #000 dashed;
}
```

También podemos dar aspecto a cada uno de los cuatro bordes por separado. Para cada borde se puede establecer su anchura, su color y su estilo.

```
div {
 border-top-width: 10px;
 border-right-width: 5px;
 border-bottom-width: 20px;
 border-left-width: 6px;
}
```

Para definir el grosor.

```
div {
 border-top-color: #CC0000;
 border-right-color: blue;
 border-bottom-color: #00FF00;
 border-left-color: #CCC;
}
```

Para definir el color.

```
div {
 border-top-style: dashed;
 border-right-style: double;
 border-bottom-style: dotted;
 border-left-style: solid;
}
```

Para definir el estilo del borde

width y height

Ancho y alto de la caja.
Valores:

- unidades.
- porcentaje.

background-color

Color de fondo.
Valores: nombre del color, valor del color.

background-image

Poner imagen de fondo.
Valores: *url* ('URL').

```
div {
 background-image: url ('../directorio/foto.gif');
}
```

En este ejemplo puedes ver aplicadas estas propiedades.

Referente a listas

list-style-type

Para el tipo de símbolo (listas sin orden) o tipo de numeración (listas ordenadas) que precede a cada elemento de una lista.
Valores:

- *circle*;
- *disc*;
- *square*;
- *decimal*;
- *lower-alpha*;
- *lower-roman*;
- *upper-alpha*;
- *upper-roman*;
- *none*;

list-style-image

Para utilizar una imagen como símbolo de elementos de una lista.
Valores:

- *url* (localización de la imagen);
- *none*;

list-style-position

Determina la forma de sangrado de las listas anidadas.
Valores:

- *inside*;
- *outside*;

En este ejemplo puedes ver aplicadas estas propiedades.

Observa en el ejemplo que estamos aplicando las propiedades sobre el selector "*ul*" y "*ol*", pudiendo aplicar también estilos a los elementos "*li*" de cada tipo de lista.

Márgenes y relleno. Atributos margin y padding

El **margen** es el espacio que hay entre el borde del elemento y los elementos que tiene alrededor, es decir, es un espacio que está en el exterior del elemento al que se aplica.

El **relleno** o ***padding*** es el espacio que hay entre el elemento y su contenido, es decir, es un espacio que está en el interior del elemento.

Es muy importante saber que el *padding* se suma al ancho y alto del elemento.

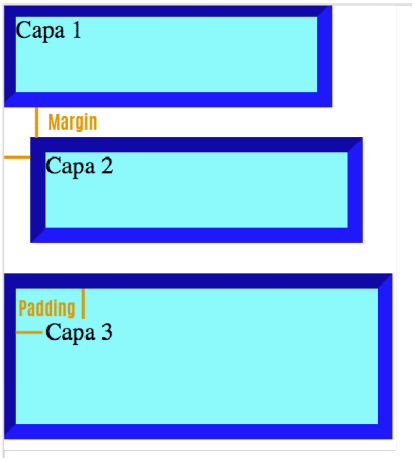
Puedes ponerlo en práctica con una sencilla página con tres capas.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="ISO-8859-1">
    <title>Margin y Padding</title>
    <link href="estilos/estilo.css" rel="Stylesheet">
</head>

<body>
    <div id="div1">Capa 1</div>
    <div id="div2">Capa 2</div>
    <div id="div3">Capa 3</div>
</body>

</html>
```

El CSS asociado:

```
@CHARSET "ISO-8859-1";

#div1, #div2, #div3 {
      width: 200px;
      height: 50px;
      border: inset 10px blue;
      background-color: cyan;
}
#div2 {
      margin: 20px;
}

#div3 {
      padding: 20px;
}
body {
      margin: 0px;
}
```

El resultado es este:



La segunda capa tiene márgenes y la tercera capa tiene relleno.

Especificación de tamaños mínimos

Hemos visto que cuando se establece el alto y ancho de un elemento en porcentaje, este se adapta al tamaño del dispositivo e incluso se redimensiona ajustándose al tamaño de una ventana del navegador. En ocasiones nos interesa que una capa tenga un tamaño mínimo pase lo que pase.

```
#todo {
      min-width: 1000px;
}
```

Para probarlo vamos a crear una página web muy simple con una única capa que se redimensiona y siempre se queda en el centro de la página:

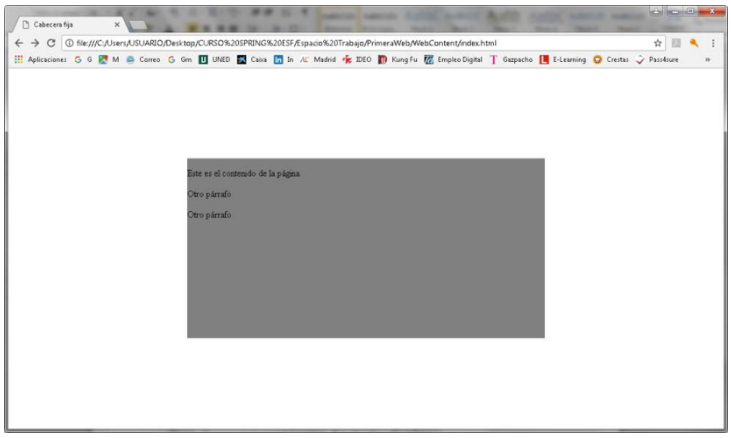
```
<!DOCTYPE html>
<html>

<head>
    <meta charset="ISO-8859-1">
    <title>Cabecera fija</title>
    <link href="estilos/estilo.css" rel="Stylesheet">
</head>

<body>
    <div id="contenido">
        <p>Este es el contenido de la página</p>
        <p>Otro párrafo</p>
        <p>Otro párrafo</p>
    </div>
</body>

</html>
```

Y el CSS asociado:

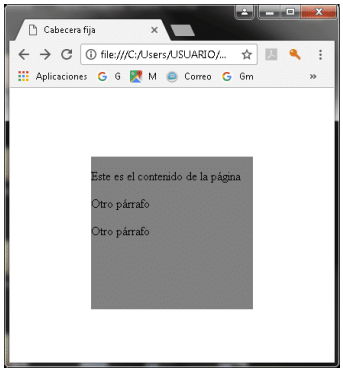
```
#contenido {
	position: absolute;
	width: 50%;
	min-width: 200px;
	height: 50%;
	min-height: 200px;
	top: 25%;
	left: 25%;
	background-color: gray;
}
```

La capa ocupará un 50% de alto y un 50% de ancho, el otro 50% se reparte como margen, ya que dejamos un 25% arriba, un 25% abajo, y el 25% sobrante quedará a la derecha y abajo. Al redimensionar la ventana del navegador podrás apreciar cómo va cambiando la capa para adaptarse. Sin embargo, el tope está en los 200px de alto y ancho, en ningún caso la capa medirá menos que eso.

Este es el resultado:



Y cuando hacemos la ventana más pequeña:



Por mucho que reduzcas el tamaño de la ventana, la capa nunca será menor de 200px de alto y de ancho.

Posicionamiento de contenedores

Capas y posicionamiento

Para gestionar la posición de los elementos dentro de su contenedor vamos a trabajar con las siguientes propiedades CSS.

- ***position***: determina el tipo de posicionamiento de los elementos en su contenedor. El contenedor de un elemento puede ser el cuerpo de la página (etiqueta *<body>*) o bien otro elemento (por ejemplo un * *).
- ***height***: altura del elemento.
- ***width***: anchura del elemento.
- ***top***: posición del elemento desde arriba.
- ***left***: posición del elemento desde la izquierda.

Para comprender mejor el funcionamiento de estas propiedades vamos a ver algunos ejemplos.

Partimos de una página HTML muy sencilla con tres capas identificadas por un *id*.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="ISO-8859-1">
    <title>Los contenedores</title>
    <link href="estilos/estilo.css" rel="Stylesheet" type="text/css">
</head>

<body>
    <div id="primero">Este es el primer DIV</div>
    <div id="segundo">Este es el segundo DIV</div>
    <div id="tercero">Este es el tercer DIV</div>
</body>

</html>
```

Y configuramos la hoja de estilos de la siguiente manera:

```
@CHARSET "ISO-8859-1";

#primero {
      position: relative;
      background-color: aqua;
      width: 300px;
      height: 100px;
}
```

```
#segundo {
	position: relative;
	background-color: gray;
	width: 400px;
	height: 100px;
}

#tercero {
	position: relative;
	background-color: aqua;
	width: 500px;
	height: 100px;
	top: 100px; /* Con respecto al flujo del documento HTML */
	left: 100px; /* Con respecto al flujo del documento HTML */
}
```

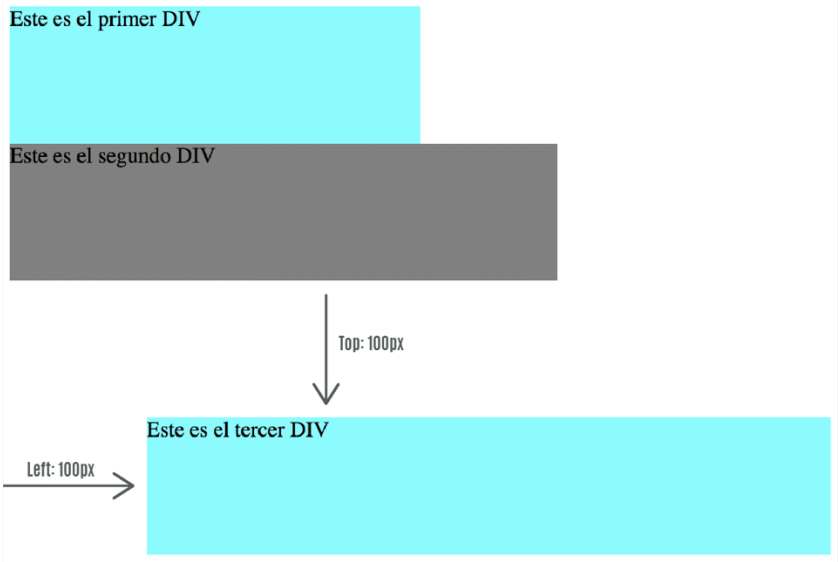
Este es el resultado:
Ejemplo HTML con tres capas identificadas por un id

El ejemplo anterior utiliza capas situadas con posicionamiento relativo. Las propiedades *top* y *left* van con respecto al flujo natural que corresponde al elemento.

Position: relative

Una capa con posicionamiento relativo se sitúa según el flujo secuencial de elementos dentro del contenedor. Observa que las capas se han situado una detrás de otra.

IMPORTANTE: cuando el posicionamiento es relativo las propiedades *top* y *left* no van en relación a la esquina superior izquierda del documento, sino a partir de la posición natural que ocupa el elemento dentro del flujo secuencial HTML.



Ancho en píxel o porcentaje

Observa que en el ejemplo los atributos *height* y *width* de las capas (* *) están establecidos en píxeles, lo que significa que tienen un tamaño fijo independientemente del tamaño del dispositivo donde se muestre la página. También es posible establecer los valores de alto y ancho en porcentaje para que los elementos se adapten al tamaño del dispositivo, así nos vamos acercando a lo que se denomina un diseño *responsive*.

Prueba a cambiar los anchos de las capas en 40%, 60% y 80% respectivamente para lograr que cambien en función del tamaño de la ventana donde se visualizan. Luego puedes abrir la página con un navegador e ir redimensionando la ventana para ver el efecto.

Position: absolute

Una capa con posicionamiento absoluto se sitúa dentro de su contenedor por coordenadas fijas (*top*, *left*) a partir de la esquina superior izquierda. Dentro de una capa con posicionamiento relativo puede haber otra con posicionamiento absoluto o viceversa.

Para ponerlo en práctica vamos a crear otra capa (etiqueta * *) dentro de la capa cuyo *id* es *segundo*.

```
<div id="segundo">Este es el segundo DIV
      <div id="segundoB">Este es un DIV hijo</div>
</div>
```

Y añadiremos el siguiente código al fichero CSS.

```
#segundoB {
      position: absolute;
      background-color: fuchsia;
      width: 60%;
      height: 50px;
      top: 50px;
      left: 50px;
}
```

El contenedor de la capa *segundoB* es la capa *segundo* y las coordenadas y ancho establecidos van con respecto a la capa *segundo*.

A continuación tienes el ejemplo completo de HTML y CSS con el resultado: Ejemplo position: absolute
En el ejemplo, el contenedor de la capa fucsia es la capa gris y sus coordenadas van en relación a su esquina superior izquierda.

Position: static

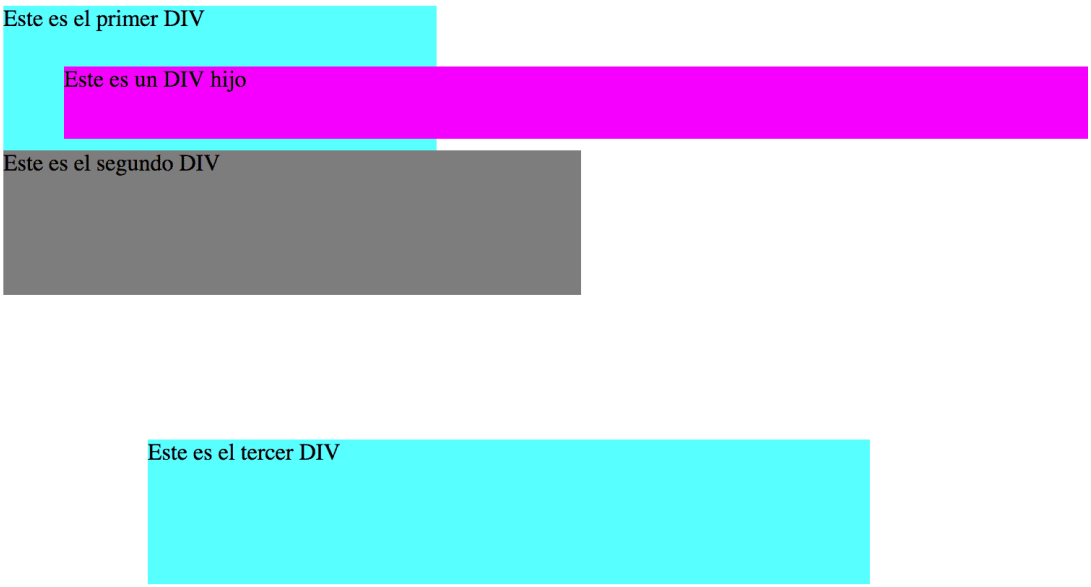
Una capa con posicionamiento estático se coloca igual que una capa relativa, pero sus elementos hijos no serán posicionados según su esquina superior izquierda, sino según la esquina del contenedor de orden superior que no sea estático.

Para probarlo podemos hacer *static* el contenedor *segundo* para comprobar lo que ocurre con su contenedor hijo *segundoB*.

```
#segundo {
      position: static;
      background-color: gray;
      width: 400px;
      height: 100px;
}
```

Las capas hijas de la capa *segundo* no se sitúan dentro de esta, podríamos decir que se escapan de su capa madre.

Este es el resultado:



Ahora las coordenadas de la capa fucsia van con respecto a la esquina superior izquierda del documento.

Este es el posicionamiento predeterminado. Puedes comprobar que si eliminas el atributo *position* funcionará igual.

Position: fixed

Funciona como el posicionamiento absoluto (*absolute*) con la diferencia de que la capa siempre estará visible aunque se utilice la barra de desplazamiento.

Vamos a ponerlo en práctica con este sencillo documento HTML que tiene una cabecera y un área de contenido.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="ISO-8859-1">
    <title>Cabecera fija</title>
    <link href="estilos/estilo.css" rel="Stylesheet">
</head>

<body>
    <div id="cabecera">
        Esta es la cabecera de la página y quiero que esté siempre visible.
    </div>
    <div id="contenido">
        <p>Este es el contenido de la página</p>
        <p>Otro párrafo</p>
        <p>Otro párrafo</p>
    </div>
</body>

</html>
```

Queremos lograr que la capa cabecera siempre esté visible.

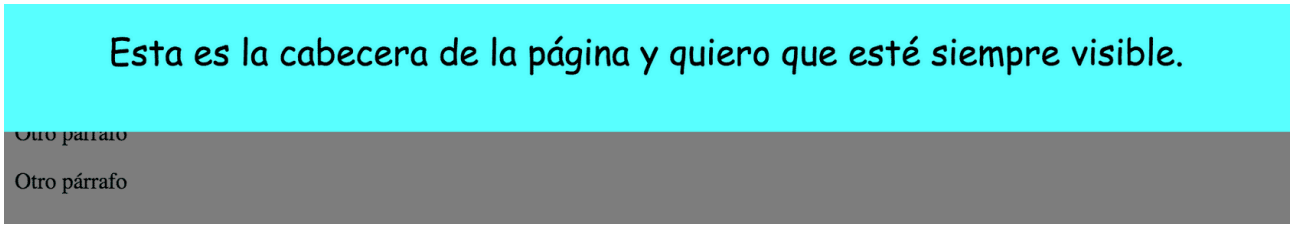
Y este es el CSS asociado a la página anterior:

```
@CHARSET "ISO-8859-1";

#cabecera {
      position: fixed;
      width: 100%;
      height: 75px;
      top: 0px;
      left: 0px;
      background-color: aqua;
      z-index: 2;
      text-align: center;
      font-family: "Comic sans MS";
      font-size: 25px;
      padding-top: 25px;
       /* Estos 15px se suman a la altura de del contenedor */
}
#contenido {
      position: absolute;
      width: 98%;
      top: 100px;
      left: 0px;
      height: 1300px;
      background-color: gray;
      z-index: 1;
      padding-left: 2%;
}
```

La capa *cabecera* tiene un posicionamiento fijo, lo que significa que siempre estará visible aunque el contenido sea muy grande.

A continuación puedes ver el código completo y el resultado:
Ejemplo HTML con cabecera y área de contenido

Y si utilizamos la barra de desplazamiento para ir hacia abajo, el resultado es que la capa de contenido se va escondiendo debajo de la capa de la cabecera, quedan superpuestas y el contenido en segundo plano.



Habrás observado que aparece un atributo CSS nuevo.

El atributo *z-index* se aplica cuando hay superposición de elementos para indicar el orden de superposición. Para nuestro ejemplo la cabecera está en primer plano y el contenido en segundo plano.

Position: inherit

El posicionamiento del elemento al que se aplica será el posicionamiento de su contenedor, es decir, el elemento padre.

En el siguiente vídeo te mostramos un ejemplo de aplicación del posicionamiento flotante.

Tipo de posicionamiento. Atributo display

El atributo *display* determina cómo se disponen los elementos a lo largo de la página.

display: block

Los elementos se disponen en bloque uno debajo de otro y ocupando todo el ancho, a no ser que se especifique un valor para la propiedad *width*. Solo tiene efecto si el posicionamiento es relativo o estático. Las etiquetas *<p>* y * * tienen este tipo de posicionamiento por defecto, este es el motivo de que siempre ocupen todo el ancho, aunque su contenido sea una simple palabra.

```
<body>
<p id="parrafo">Hola mundo</p>
</body>
```

Los párrafos tienen por defecto posicionamiento en bloque.

```
#parrafo {
      background-color: aqua;
}
```

El resultado es el siguiente:
Ejemplo display: block

display: inline

Los elementos se disponen en línea uno a continuación de otro, el tamaño se ajusta al contenido, luego no tienen ningún efecto las propiedades *width* y *height*.

```
#parrafo {
      background-color: aqua;
      display: inline;
}
```

Resultado:
Ejemplo display: inline

Párrafo dispuesto *inline* (en línea). No obedece a las especificaciones de alto y ancho.

Algunas etiquetas tienen un tipo de visualización *inline* por defecto, este es el caso de las etiquetas *<a>* (enlace) o **.

display: inline-block

Los elementos se disponen en línea y en bloque, uno a continuación de otro, el tamaño es el establecido por los atributos *height* y *width*.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="ISO-8859-1">
    <title>Ejemplo display: inline-block</title>
    <link href="estilos/estilo.css" rel="Stylesheet">
</head>

<body>
    <div id="div1">Capa 1</div>
    <div id="div2">Capa 2</div>
    <div id="div3">Capa 3</div>
```

```
    <div id="div4">Capa 4</div>
</body>

</html>
```

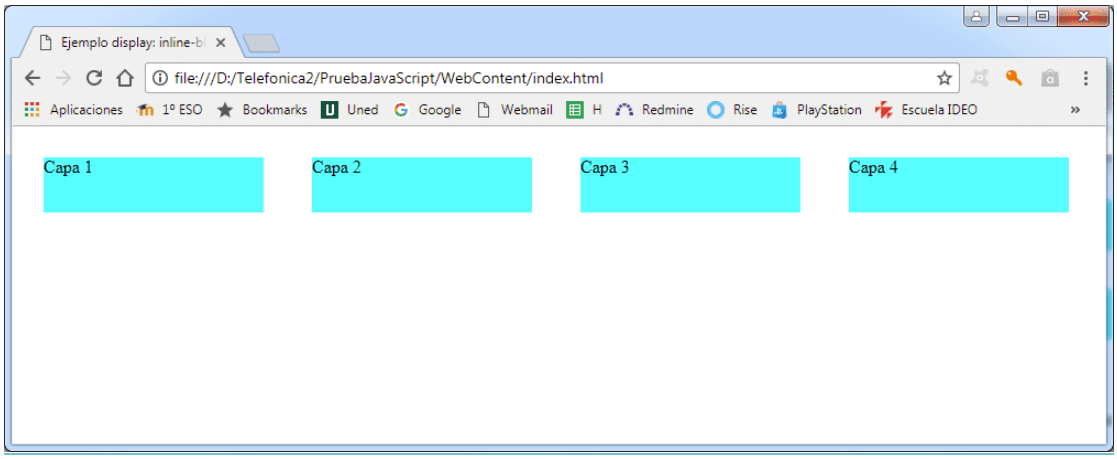
Página con cuatro capas que vamos a disponer en línea y en bloque.

```
#div1, #div2, #div3, #div4 {
      width: 200px;
      height: 50px;
      background-color: cyan;
      display: inline-block;
      margin: 20px;
}
```

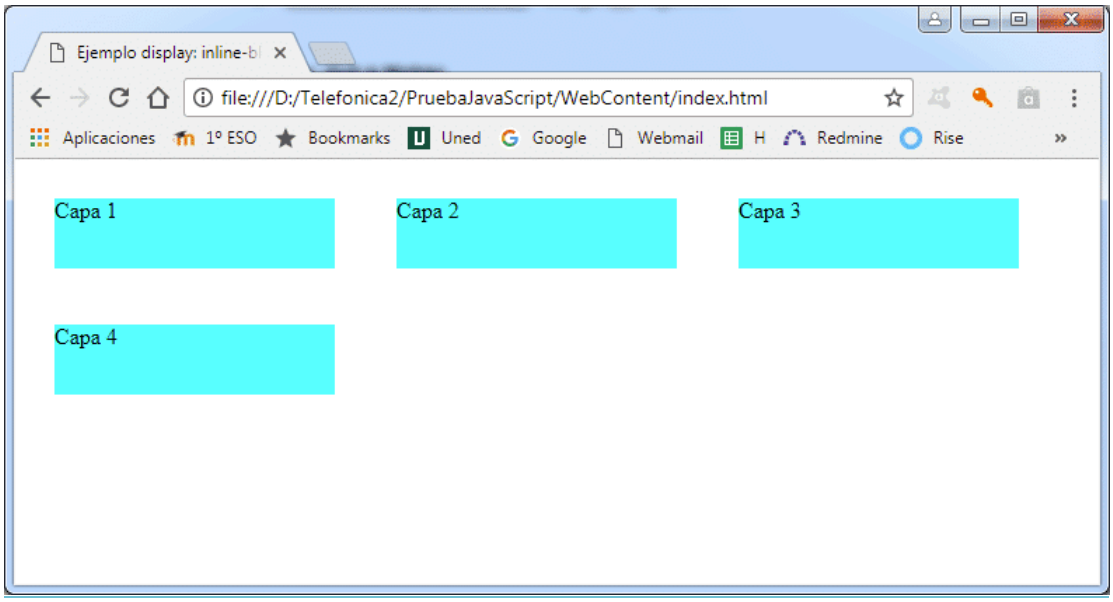
Las capas se dispondrán en línea, una tras otra ocupando el espacio disponible, pero también en bloques de 200px por 50px respetando el tamaño especificado. El resultado es una página que se adapta al tamaño del dispositivo.
El resultado son cuatro capas que se van distribuyendo una tras otra respetando el tamaño especificado. Una fila contendrá el número de capas que quepan según el ancho del contenedor.

Ejemplo display: inline-block

Dependiendo del tamaño de la ventana del navegador, cada capa se colocará como se ve en las imágenes. Pulsa sobre ellas para ampliarlas.

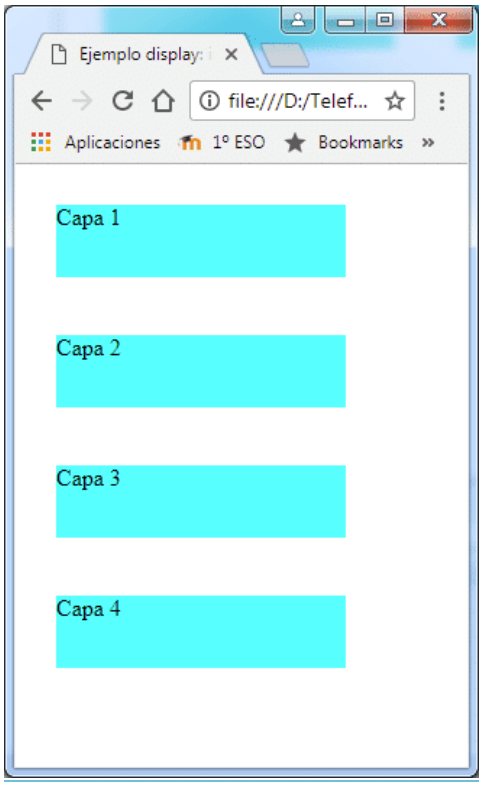


Cuatro capas en una fila.



Tres capas por fila.

Dos capas por fila.



Una capa por fila.

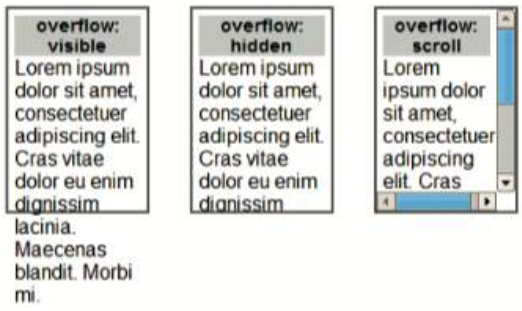
display: list-item

En bloque y en línea pero con viñetas.

Propiedad overflow

Normalmente las cajas HTML se ajustarán al contenido que tengan, pero en algunas ocasiones el contenido de un elemento no cabe dentro, dado que hemos definido un alto y ancho.

Con *overflow* podemos indicar el comportamiento que queremos que tenga el contenido.

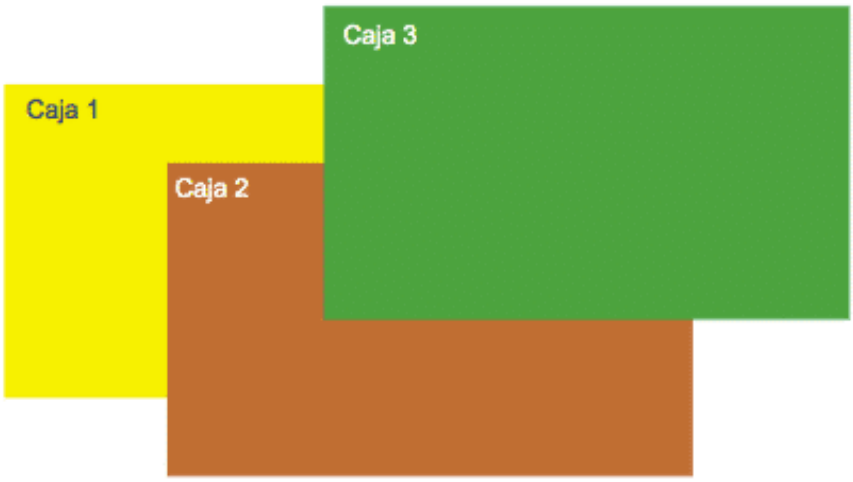



```
.caja1{
overflow:visible;
}
.caja1{
overflow:hidden;
}
.caja1{
overflow:scroll;
}
```

Z-index

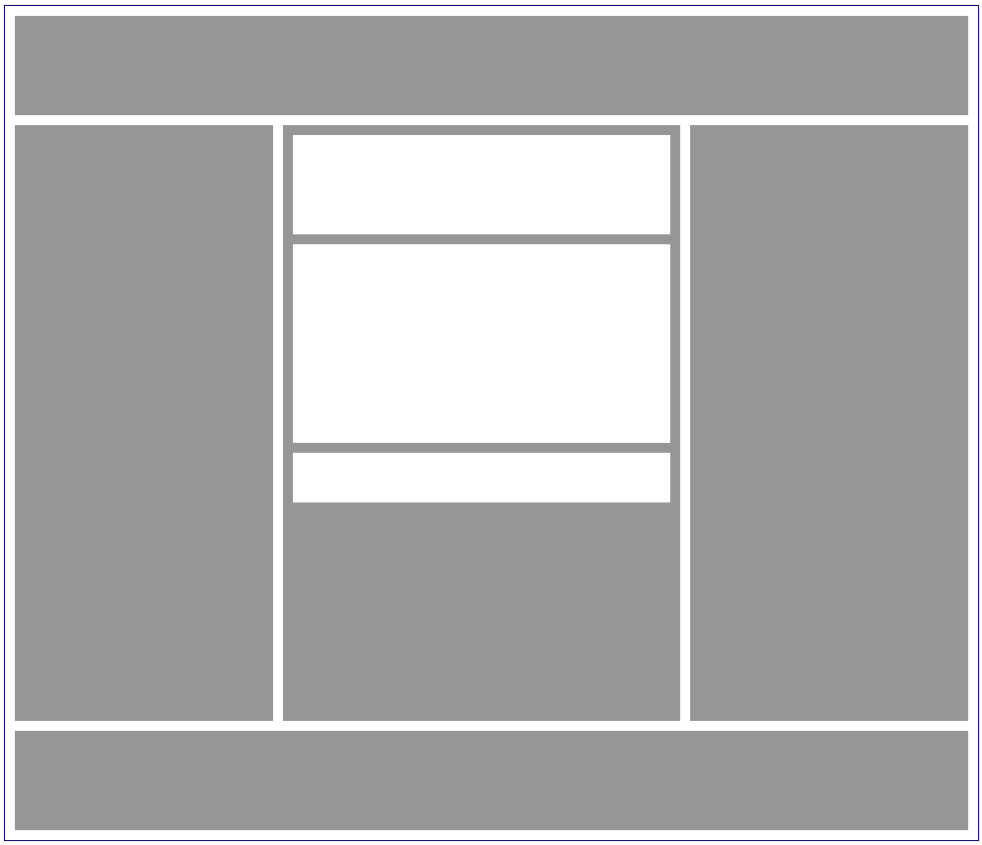
Además de la posición horizontal y vertical, podemos controlar el eje Z. Con la propiedad *z-index* podemos determinar el orden de las capas.

```
div {
position: absolute;
}
#caja1 {
z-index: 5;
top: 1em;
left: 8em;
}
#caja2 {
z-index: 15;
top: 5em;
left: 5em;
}
#caja3 {
z-index: 25;
top: 2em;
left: 2em;
}
```



Ejemplo: estructura página HTML5

Veamos ahora un ejemplo práctico de cómo aplicaríamos las reglas CCS para crear la típica estructura de una página HTML5.



Estructura HTML 5

Otros efectos

Bordes, sombras y degradados

Los nuevos atributos de estilo de CSS3 te permiten crear bordes con sombra y rellenos degradados. Como son atributos muy nuevos, es posible que no te funcionen con todos los navegadores y si funcionan puede haber variaciones entre unos y otros.

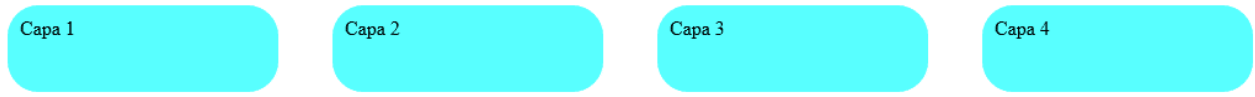
Si empleas una versión muy antigua de navegador con seguridad no funcionará, pero no te dará problemas, simplemente el elemento se visualizará de la manera habitual, sin sombras, bordes redondeados, degradados, etc.

Bordes redondeados

Añade estos atributos a las capas del ejemplo anterior:

```
border-radius: 25px;
padding: 10px;
```

Resultado:

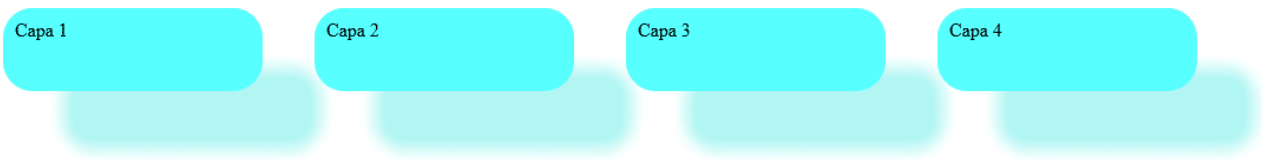


Sombras

```
box-shadow: 50px 50px 25px #A9F5F2;
```

Define una sombra en el elemento, los valores asignados son: posición de la sombra desde la derecha, posición de la sombra desde abajo, nivel de transparencia y color de la sombra.

Resultado:



Prueba a ir modificando los cuatro valores y comprueba los cambios.

Degradados

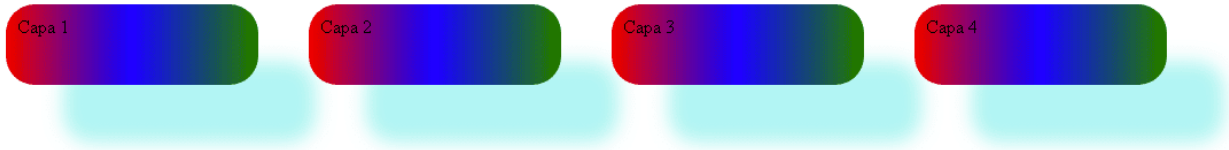
```
background: linear-gradient(left, red , blue, green);
background: -webkit-linear-gradient(left, red , blue, green);
```

El fondo de las capas a las que se aplican está compuesto por un degradado de varios colores.

El primer parámetro es el punto de partida, estamos indicando que el degradado comience por la izquierda. El resto de argumentos son los puntos de color.

Este atributo de momento funciona con pocos navegadores, concretamente funciona con Mozilla Firefox. Observa que hemos puesto dos versiones (*linear-gradient* y *–webkit-liner-gradient*), la segunda versión es para compatibilizar con navegadores Google Chrome.

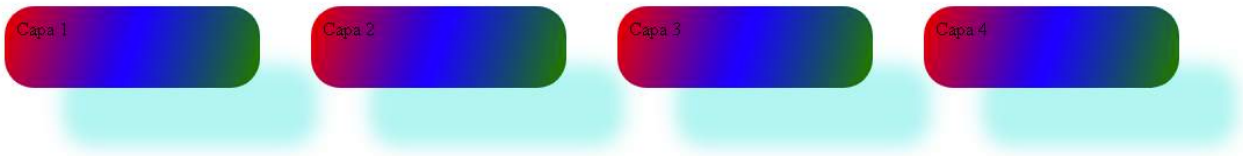
Este es el resultado:



Degradado lineal.

También puede hacerse en diagonal

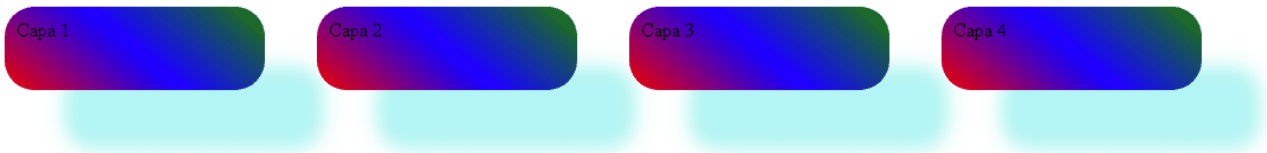
```
background: linear-gradient(left top, red , blue, green);
background: -webkit-linear-gradient(left top, red , blue, green);
```



En diagonal especificando ángulo

```
background: linear-gradient(50deg, red , blue, green);
background: -webkit-linear-gradient(50deg, red , blue, green);
```

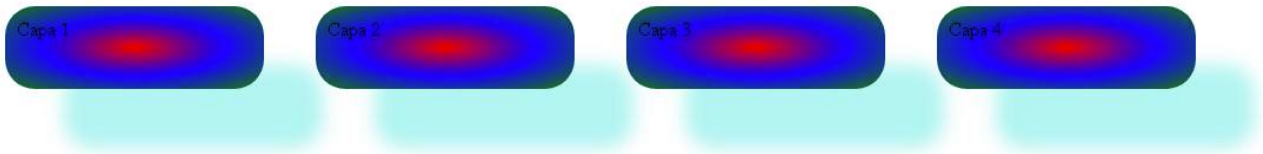
Degradado diagonal 50 grados.



Degradado radial

Formando una elipse.

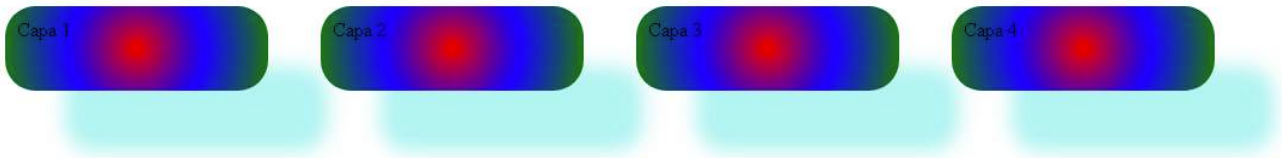
```
background: radial-gradient(red , blue, green);
background: -webkit-radial-gradient(red , blue, green);
```



Degradado en forma de elipse.

Círculo en lugar de elipse

```
background: radial-gradient(circle, red, blue, green);
background: -webkit-radial-gradient(circle, red, blue, green);
```



Degradado formando círculos.

Pseudoclases y pseudoelementos

Definición

Pseudoclases

Nos permiten dar algo de dinamismo a los elementos por medio de un evento. Se escriben detrás del selector, separadas por dos puntos.

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

Puedes ampliar la información sobre las pseudoclases en este enlace.
https://www.w3schools.com/css/css_pseudo_classes.asp

Pseudoelementos

Es la posibilidad de aplicar normas de estilo ajenos a la estructura clásica de etiquetas, poniendo nuevas normas de aplicación (al primero, al último, etc.). Se escriben detrás del selector, separadas por dos puntos dobles (: :).

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

Puedes ampliar la información sobre los pseudoelementos en este enlace.
https://www.w3schools.com/css/css_pseudo_elements.asp

Pseudoclases

Aplicado a los enlaces

- ***:hover*** - Cuando el puntero del ratón se posiciona encima del *link*.
- ***:focus*** - Selección del enlace usando el tabulador del teclado.
- ***:visited*** - Selecciona los enlaces que ya fueron visitados.
- ***:active*** - El momento en el que hacemos clic.

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

```
a:active {
color:green;
}

a:hover, a:focus {
color:blue;
}
a:visited {
color:yellow;
}
```

Aplicado a un *input* de formulario

- ***:focus*** - Cuando el cursor está dentro del campo.
- ***:target*** - Cuando vamos a una url con un *id* al final: http://pagina.es#Objetivos.
- ***:enabled*** - Selecciona los *input* que no tienen el atributo ***disabled="true"***, es decir los que podemos editar.
- ***:disabled*** - Lo contrario que el anterior, los que no podemos editar.
- *:**checked*** - Para las *checkbox* y *radio* que están seleccionadas.

```
input:focus {
background-color: rgb (156,153,133);
}
input:disabled {
background-color: rgb (196,109,59);
}
input:checked {
cursor: crosshair;
}
input:indeterminate {
cursor: e-resize;
}
```

Para el árbol de etiquetas

- ***:root*** - Selecciona al elemento raíz del DOM. La etiqueta *<html>*.
- ***:first-child*** - Selecciona al primer hijo dentro de un elemento.
- ***:last-child*** - Selecciona el último hijo.
- ***:nth-child(N)*** - Selecciona elementos de valor de *N*:
    - (n) Todos los hijos.
    - (2n+1) Todos los hijos en posición impar.
    - (2n) Todos los hijos en posición par.
    - (nX) Todos los hijos a partir de X.
    - (-nX) Todos los hijos hasta X.
    - (X) El hijo en la posición X.

```
/*Selecciona a todos los p que sean primer hijo*/
p: first-child {
background-color: red;
}
```

```
/*Selecciona a cualquier elemento que sea primer hijo*/
: first-child {
background-color: yellow;
}
```

Para el contenido

- ***:empty*** - Selecciona a todos los elementos que no tengan hijos o texto.
- ***:not(X)*** - Selecciona a todos los elementos que no posean una característica X.

Para el texto

- ***:first-letter*** - Selecciona la primera letra del texto.
- ***:first-line*** - Selecciona la primera línea del texto.

Puedes ver un pequeño ejemplo de alguna de estas pseudoclases.

Pseudoclases

Pseudoelementos

- ***::before*** - Para insertar un contenido (texto) antes de una etiqueta.
- ***::after*** - Para insertar un contenido (texto) después de una etiqueta.

```
p::before {
content: "lo que queremos insertar antes";
}
p::after{
content: "lo que queremos insertar despues";
}
```

La propiedad "*content*" permite poner un texto predeterminado. En este caso se inserta el texto antes o después del elemento "*p*".

Pseudoelementos

Diseño responsive

Media queries

Las *media queries* se utilizan en la tecnología CSS para condicionar la aplicación de un grupo de atributos de diseño al cumplimiento de una condición, que tendrá que ser relativa al tipo de dispositivo, orientación, tipo de visualización, etc.

Vamos a ponerlo en práctica con este documento HTML:

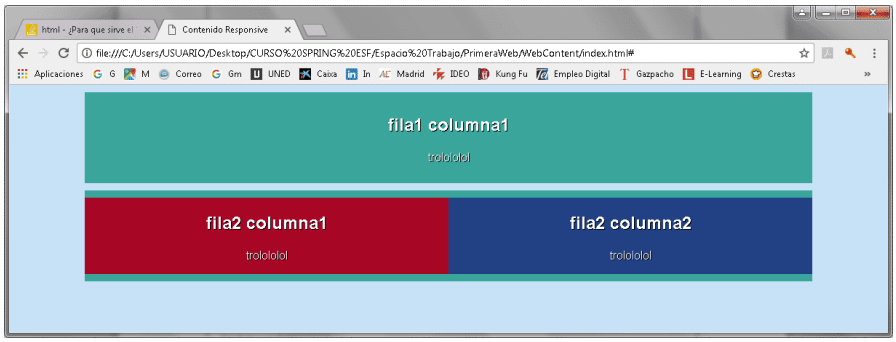
```
<!doctype html>
<html lang="es">
<head>
   <meta charset="utf-8"/>
   <meta name="viewport" content="width=device-width"/>
   <title>Contenido Responsive</title>
   <link rel="stylesheet" href="estilos/estilo.css"/>
</head>
<body>
   <section>
      <h2>fila1 columna1</h2>
      <p>trololololol</p>
   </section>
   <section>
      <article class="fila1">
         <h2>fila2 columna1</h2>
         <p>trololololol</p>
      </article>
      <article class="fila2">
         <h2>fila2 columna2</h2>
         <p>trololololol</p>
      </article>
   </section>
</body>
</html>
```

La etiqueta *<meta name="viewport" content="width=device-width"/>* representa el área visible del navegador. Es una etiqueta HTML5 que sirve para optimizar los sitios para móviles, ayudando a definir el ancho, alto y escala del área usada por el navegador para mostrar contenido.

Código CSS asociado con la página anterior:

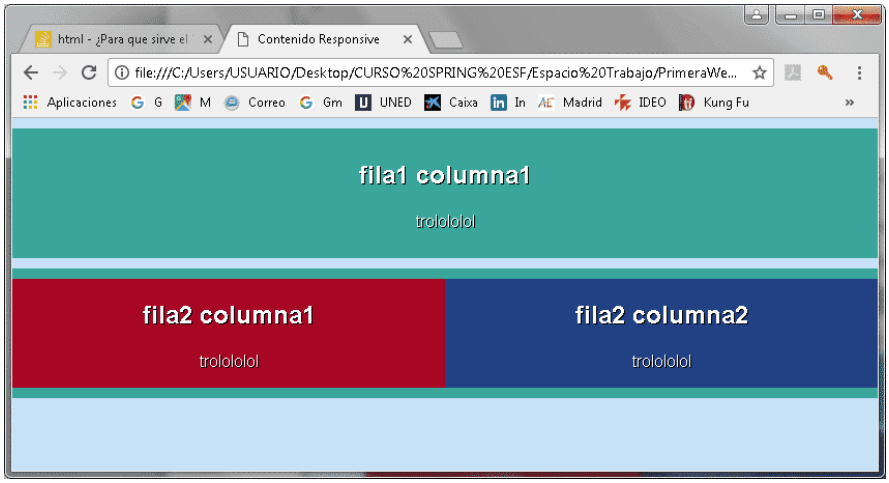
```
@CHARSET "ISO-8859-1";
article {
   float:left;
   width:50%;
}

body {
   background:#C3E5F9;
   color:white;
   font-size:16px;
   font-family:Arial;
   text-shadow:1px 1px 0 black;
   margin:0;
}

section{
   background:#12A89D;
   margin:10px auto;
   overflow:hidden;
   padding:10px 0;
   text-align:center;
   width: 1000px;
}

.fila1{
   background:#B30F2A;
}

.fila2{
   background:#1C4583;
}

@media screen and (max-width: 1000px){
   section{
      width:100%;
   }
}

@media screen and (max-width: 700px){
   article{
      width:100%;
   }
}
```

Hemos conseguido un diseño *responsive* que tiene tres comportamientos diferentes. Observa a continuación cómo se vería el resultado dependiendo del tamaño de la pantalla donde se visualiza.



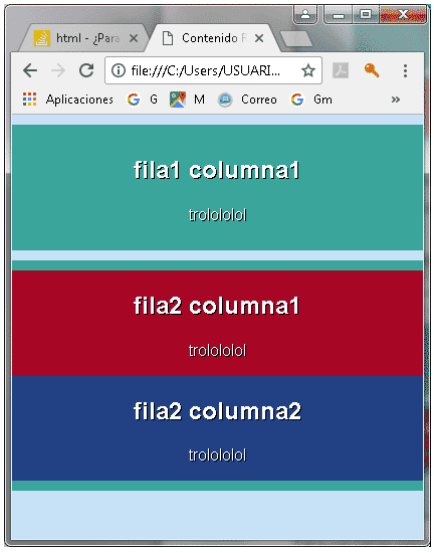
Diseño para PC

Para PC con ancho mayor o igual a 1000px deja unos márgenes a izquierda y derecha y los artículos se muestran en la misma fila.



Diseño para tabletas

Para pantallas entre 700 y 1000 pixels (normalmente tabletas) las capas ocupan un ancho del 100% de la pantalla para aprovechar mejor el tamaño.



Diseño para móvil

Para pantallas con ancho menor de 700px (normalmente móviles) los artículos se visualizan uno debajo de otro.

Las media query

Las *media query* están formadas por un ***media type*** y una o varias **condiciones**.

```
@media screen and (max-width: 700px) { }
```

En este caso el *media type* es *screen* y la condición de que el tamaño máximo sea 700px.

El *media* ***type*** puede ser ***all*** (todos), ***screen*** (pantalla), ***speech*** (lectores de pantalla) o ***print*** (impresión).

Podemos no especificar el *media type*, en cuyo caso será *all*.

```
@media (max-width: 700px) { }
```

Para todos los medios (*print* y *screen*) y tamaño del dispositivo menor o igual a 700px.

Podemos especificar varias condiciones:

```
@media (min-width: 700px) and (orientation: landscape) { }
```

Tamaño máximo sea 700px y la orientación horizontal.

```
@media (min-width: 700px) and (orientation: portrait) { }
```

Tamaño máximo sea 700px y la orientación vertical.

Otro ejemplo

Veamos un ejemplo CSS que cambia el tamaño de la fuente y el color de fondo en función del tamaño del dispositivo y es para pantalla o para impresión.

```
@CHARSET "ISO-8859-1";

body {
      background-color: aqua;
      font-size: 25px;
}

@media screen and (max-width: 700px) {
      body {
            background-color: gray;
            font-size: 20px;
      }
}
```

```
@media screen and (max-width: 300px) {
      body {
            background-color: gray;
            font-size: 15px;
      }
}
@media print {
      body {
            font-size: 50px;
            font-family: "Comic sans MS";
      }
}
```

Ejemplo de aplicación

Veamos un ejemplo de adaptación de los elementos de una página con la estructura de HTML5. Podemos comprobar que dependiendo del tamaño del dispositivo los elementos cambian para adaptarse, ajustando su ancho o desapareciendo, como el caso de la adaptación a móvil.

Adaptación a móviles

Despedida

Resumen

Has terminado la lección, veamos los puntos más importantes que hemos tratado.
En esta unidad hemos visto cómo dotar de estilos nuestros documentos HTML, pudiendo realizar diseños atractivos y que aumente la usabilidad de nuestras páginas.

Es importante que entiendas que no podemos poner un ejemplo de aplicación de cada una de las reglas CSS, por lo que lo debes practicar y probar las combinaciones para conseguir el diseño deseado.

También debes tener en cuenta que hoy en día son muchos y muy variados los dispositivos encargados de visualizar el contenido, por lo que debes adaptar tus documentos HTML a múltiples tipos de pantalla, o lo que es lo mismo crear diseños *responsive*.