

# Tutorial de CSS3 desde cero

Este documento está pensado para que pueda ser desarrollado por una persona que conoce solo HTML. El objetivo es poder aprender CSS de forma sencilla viendo un concepto teórico, luego algunos ejercicios resueltos y por último y lo más importante efectuar una serie de ejercicios.

## 1. Hojas de estilo CSS

---

CSS son las siglas de Cascade Style Sheet que traducido significa hojas de estilo en cascada.

Las hojas de estilo es una tecnología que nos permite controlar la apariencia de una página web. En un principio, los sitios web se concentraban más en su contenido que en su presentación.

HTML no pone atención en la apariencia del documento, sino en la estructura. CSS describe como los elementos dispuestos en la página son presentados al usuario. CSS es un gran avance que complementa el HTML y la Web en general.

Con CSS podemos especificar estilos como el tamaño, fuentes, color, espaciado entre textos y recuadros, así como el lugar donde disponer texto e imágenes en la página.

El lenguaje de las Hojas de Estilo está definido en la Especificaciones CSS1, CSS2 y CSS3 del World Wide Web Consortium (W3C), es un estándar aceptado por toda la industria relacionada con la Web, o por lo menos, gran parte de navegadores (es verdad que el Internet Explorer de Microsoft nos dio algunos dolores de cabeza en versiones antiguos). Para tener una visión más exhaustiva visita la página oficial de [W3C](http://www.w3.org).

Podemos asociar las reglas de estilo a las marcas HTML de tres maneras: directamente a la marca, en el head de la página o agrupar las reglas de estilo en un archivo independiente con extensión \*.css

Veremos las tres metodologías, pero pondremos énfasis en la tercera forma, que es la más adecuada para separar el contenido de la página y la forma como se debe representar la misma por medio de la hoja de estilo.

Un ejemplo de la aplicación de CSS a un documento HTML podría ser:

```
<!DOCTYPE html>
<html>
<head>
  <title>Prueba de hojas de estilo</title>
  <meta charset="UTF-8">
</head>
<body>
<p style="color:#000000;background-color:yellow;font-
family:verdana;font-size:25px;text-align:center">Esto es un
ejemplo</p>
</body>
</html>
```

Con lo que conseguiríamos este efecto:

Esto es un ejemplo

## 2. Definición de estilos a nivel de elemento HTML

Es la forma más fácil pero menos recomendada para aplicación de un estilo a un elemento HTML. Se define en la propiedad style los estilos a definir para dicho elemento.

Es común a veces definir estilos directamente en los elementos HTML cuando estamos probando diseños de elementos particulares de la página y posteriormente trasladar el estilo creado a la zona de definición de estilos.

La sintaxis para definir un estilo a un elemento HTML es la siguiente:

```
<!DOCTYPE html>
<html>
<head>
  <title>Título de la página</title>
  <meta charset="UTF-8">
</head>
<body>
<h1 style="color:#ff0000;background-color:#ffff00">
  Este mensaje es de color rojo sobre fondo amarillo.
</h1>
</body>
</html>
```

El resultado en el navegador al cargar esta página es:

**Este mensaje es de color rojo sobre fondo amarillo.**

Por defecto, todo navegador tiene un estilo definido para cada elemento HTML, lo que hacemos con la propiedad style es redefinir el estilo por defecto. En este problema definimos que el elemento h1 defina como color de texto el rojo y como color de fondo el amarillo:

```
<h1 style="color:#ff0000;background-color:#ffff00">
  Este mensaje es de color rojo sobre fondo amarillo.
</h1>
```

Veremos más adelante que hay muchas propiedades en CSS. En este primer ejemplo inicializamos las propiedades color (define el color del texto) y **background-color** (define el color de fondo del texto).

Cada vez que inicializamos una propiedad debemos separarla de la siguiente por punto y coma.

Cuando definimos estilos directamente en el elemento HTML, tenemos que tener en cuenta que el estilo se aplica únicamente a dicho elemento donde inicializamos la propiedad style, es decir, si tenemos dos secciones h1, deberemos definir la propiedad style para cada elemento:

```
<h1 style="color:#ff0000;background-color:#ffff00">
Primer título
</h1>
<h1 style="color:#ff0000;background-color:#ffff00">
Segundo título
</h1>
```

Como podemos observar, más allá que los dos estilos son exactamente iguales, estamos obligados a definirlos para cada elemento HTML.

### 3. Definición de estilos a nivel de página

---

También podemos hacer la definición de estilos para los distintos elementos HTML de la página en una sección especial de la cabecera que la encerramos entre las marcas HTML (en su interior definimos los estilos para los elementos HTML que necesitamos):

```
<style>
</style>
```

El problema del concepto anterior donde debíamos crear un estilo similar para el elemento h1 se puede resolver en forma más adecuada empleando la definición de estilos a nivel de página.

Ejemplo: Mostrar dos títulos con texto de color rojo sobre fondo amarillo.

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <style>
    h1 {color:#ff0000;background-color:#ffff00}
  </style>
</head>
<body>
<h1>Primer título</h1>
<h1>Segundo título</h1>
</body>
</html>
```

El resultado en el navegador al cargar esta página es:

**Primer título**

**Segundo título**

Podemos observar que en la cabecera de la página definimos la sección:

```
<style>
h1 {color:#ff0000;background-color:#ffff00}
</style>
```

Debe estar encerrada por el elemento style. En este ejemplo indicamos al navegador que en todos los lugares de esta página donde se utilice el elemento h1 debe aplicar como estilo de color de texto el rojo y fondo el amarillo. Podemos observar que es mucho más eficiente que definir los estilos directamente sobre los elementos HTML dentro del cuerpo de la página.

Podemos definir estilos para muchos elementos en la sección style:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <style>
    h1 {color:#ff0000}
    h2 {color:#00ff00}
    h3 {color:#0000ff}
  </style>
</head>
<body>
<h1>rojo</h1>
<h2>verde</h2>
<h3>azul</h3>
</body>
</html>
```



## 4. Propiedades relacionadas a fuentes

Contamos con una serie de propiedades relacionadas a fuentes:

- **font-family**
- **font-size**
- **font-style**
- **font-weight**
- **font-variant**

Podemos inicializar algunas o todas las propiedades relacionadas a fuentes, un ejemplo:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <style>
    h1 {
      font-family:times new roman;
      font-size:30px;
      font-style:italic;
      font-weight:bold;
    }
    h2 {
      font-family:verdana;
    }
  </style>
</head>
<body>
  <h1>Titulo de nivel 1</h1>
  <h2>Titulo de nivel 2</h2>
</body>
</html>
```



```
    font-size:20px;
  }
</style>
</head>
<body>
<h1>Titulo de nivel 1</h1>
<h2>Titulo de nivel 2</h2>
</body>
</html>
```

Como podemos observar, hemos definido dos reglas de estilos para los elementos h1 y h2, eso significa que el navegador utilizará esas reglas de fuentes para todas las partes de la página que se utilicen dichos elementos HTML.

La primera regla definida para el elemento h1 es:

```
h1 {
    font-family:times new roman;
    font-size:30px;
    font-style:italic;
    font-weight:bold;
}
```

Recordemos que para definir la regla de estilo debemos indicar el nombre del elemento HTML a la que definiremos el estilo (en este caso h1) y luego, entre llaves, todas las propiedades y sus valores separados por punto y coma.

La primera propiedad inicializada es **font-family**, algunas de las fuentes más comunes que puede acceder el navegador son:

Arial, **Arial Black**, Arial Narrow, Courier New, Georgia, **Impact**, Tahoma, Times New Roman, Verdana, etc.

En caso que la fuente no esté disponible el navegador selecciona el estilo por defecto para ese elemento HTML.

Podemos definir varias fuentes por si acaso alguna no se encuentra disponible para el navegador, esto lo hacemos separando por coma todas las fuentes (se eligen de izquierda a derecha):

```
font-family: verdana, arial, georgia;
```

La segunda propiedad inicializada es **font-size**, hay varias medidas para indicar el tamaño de la fuente (veremos más adelante otros sistemas de medida), en nuestro caso indicamos en píxeles:

```
font-size:30px;
```

La tercera propiedad es **font-style**, que puede tener los siguientes valores: normal, italic, oblique.

La última propiedad es **font-weight** (peso de la fuente), pudiendo tomar los siguientes valores:

**normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800 y 900.**

Esta propiedad indica el peso de la fuente (mientras tenga un valor mayor los caracteres serán más rellenos)

La segunda regla define sólo dos propiedades relacionadas a la fuente:

```
h2 {  
    font-family:verdana;  
    font-size:20px;  
}
```

Las propiedades que no se inicializan quedan como responsabilidad al navegador, quien elegirá los valores correspondientes.

Existe una última propiedad no utilizada en este ejemplo que es **font-variant** que puede asumir estos dos valores:

small-caps y normal.

Define si la fuente se muestra en mayúsculas tipo normal o pequeñas.

## 5. Agrupación de varios elementos HTML

La agrupación de varios elementos HTML con una misma regla de estilo nos permite ahorrar la escritura de reglas duplicadas para diferentes elementos de HTML.

La sintaxis es disponer los nombres de los elementos HTML que queremos aplicar una regla separados por comas:

```
h1,h2,h3 {  
    font-family:verdana;  
    color:#0000ff;  
}
```

Supongamos que queremos la misma fuente y color para los elementos h1, h2 y h3 luego podemos implementarlo de la siguiente manera:

```
<!DOCTYPE html>  
<html>  
<head>  
    <title>Problema</title>  
    <meta charset="UTF-8">  
  
    <style>  
h1,h2,h3 {  
    font-family:verdana;  
    color:#0000ff;  
}  
    </style>  
</head>  
<body>  
<h1>Título de nivel 1</h1>  
<h2>Título de nivel 2</h2>  
<h3>Título de nivel 3</h3>  
</body>  
</html>
```

**Título de nivel 1**

**Título de nivel 2**

**Título de nivel 3**

Es decir, separamos por coma todos los elementos a los que se aplicará la misma regla de estilo:

```
h1,h2,h3 {
  font-family:verdana;
  color:#0000ff;
}
```

## 6. Definición de varias reglas para un elemento

En algunas circunstancias, es necesario desglosar la inicialización de propiedades en distintas reglas.

Podemos definir más de una regla para un mismo elemento HTML, en este ejemplo el elemento h1 tiene dos reglas:

```
h1,h2,h3,h4,h5,h6 {
  font-family:Verdana;
}
h1 {
  font-size:40px;
}
```

Supongamos que queremos todas las cabeceras con la misma fuente pero tamaños de fuente distintos, luego podemos implementarlo de la siguiente manera:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
<style>
h1,h2,h3,h4,h5,h6 {
  font-family:Verdana;
}
h1 {
  font-size:40px;
}
h2 {
  font-size:30px;
}
h3 {
  font-size:25px;
}
h4 {
  font-size:20px;
}
h5 {
  font-size:15px;
}
h6 {
  font-size:10px;
}
</style>
</head>
```

**Título de nivel 1**

**Título de nivel 2**

**Título de nivel 3**

**Título de nivel 4**

**Título de nivel 5**

**Título de nivel 6**

```
<body>
<h1>Título de nivel 1</h1>
<h2>Título de nivel 2</h2>
<h3>Título de nivel 3</h3>
<h4>Título de nivel 4</h4>
<h5>Título de nivel 5</h5>
<h6>Título de nivel 6</h6>
</body>
</html>
```

Es decir, podemos inicializar un conjunto de elementos HTML con una serie de propiedades de estilo comunes. Luego agregamos en forma individual las propiedades no comunes a dichas marcas:

```
h1,h2,h3,h4,h5,h6 {
    font-family:Verdana;
}
h1 {
    font-size:40px;
}
```

Es decir, al elemento h1 realmente le hemos definido dos propiedades: **font-family** y **font-size**. Lo mismo para los otros elementos HTML.

## 7. Propiedades relacionadas al texto

---

Las propiedades relacionadas al texto son:

- **color**
- **text-align**
- **text-decoration**

“**color**” nos permite definir el color del texto, lo podemos indicar por medio de tres valores hexadecimales que indican la mezcla de rojo, verde y azul. Por ejemplo si queremos rojo puro debemos indicar:

```
color:#ff0000;
```

Si queremos verde puro:

```
color:#00ff00
```

Si queremos azul puro:

```
color:#0000ff
```

Luego si queremos amarillo debemos mezclar el rojo y el verde:

```
color:#ffff00
```

Hay muchos programas que nos permiten seleccionar un color y nos descomponen dicho valor en las proporciones de rojo, verde y azul.

Otra forma de indicar el color, si tenemos los valores en decimal, es por medio de la siguiente sintaxis:

```
color:rgb(255,0,0);
```



Es decir, por medio de la función `rgb(red, green, blue)`, indicamos la cantidad de rojo, verde y azul en formato decimal.

La segunda propiedad relacionada al texto es **text-align**, que puede tomar alguno de estos cuatro valores:

**left, right, center y justify**

Si especificamos:

`text-align:center;`

El texto aparecerá centrado. Si queremos justificar a derecha, emplearemos el valor `right` y si queremos a la izquierda, el valor será `left`.

La tercera propiedad relacionada al texto que podemos emplear es **text-decoration** que nos permite entre otras cosas que aparezca subrayado el texto, tachado o una línea en la parte superior, los valores posibles de esta propiedad son:

**none, underline, overline y line-through**

Como ejemplo, definiremos estilos relacionados al texto para los elementos de cabecera `h1`, `h2` y `h3`:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
<style>
h1 {
  color:#ff0000;
  text-align:left;
  text-decoration:underline;
}
h2 {
  color:#dd0000;
  text-align:center;
  text-decoration:underline;
}
h3 {
  color:#aa0000;
  text-align:right;
  text-decoration:underline;
}
</style>
</head>
<body>
<h1>Título de nivel 1.</h1>
<h2>Título de nivel 2.</h2>
<h3>Título de nivel 3.</h3>
</body>
</html>
```

Título de nivel 1.

Título de nivel 2.

Título de nivel 3.

Es decir, para los títulos de nivel 1 tenemos la regla:

```
h1 {
  color:#ff0000;
  text-align:left;
  text-decoration:underline;
}
```

Color de texto rojo intenso, el texto debe aparecer de izquierda a derecha y subrayado.

Luego para el elemento h2 tenemos:

```
h2 {
  color:#dd0000;
  text-align:center;
  text-decoration:underline;
}
```

El color sigue siendo rojo pero un poco más oscuro, el texto debe aparecer centrado y subrayado.

Y por último:

```
h3 {
  color:#aa0000;
  text-align:right;
  text-decoration:underline;
}
```

Para los títulos de nivel tres, el texto es rojo más oscuro, alineado a derecha y subrayado.

## 8. Otras propiedades relacionadas al texto

---

Ahora veremos más propiedades relacionadas al texto:

- **letter-spacing**
- **word-spacing**
- **text-indent**
- **text-transform**

Las propiedades **letter-spacing** y **word-spacing** permiten indicar el espacio que debe haber entre los caracteres y entre las palabras respectivamente.

La propiedad **text-indent**, indenta la primera línea de un texto. A partir de la segunda línea, el texto aparece sin indentación. Podemos indicar un valor negativo con lo que la indentación es hacia la izquierda.

Por último, la propiedad **text-transform** puede inicializarse con alguno de los siguientes valores:

**none**, **capitalize**, **lowercase** y **uppercase**.

Cada uno de estos valores transforman el texto como sigue:

- **capitalize**: Dispone en mayúsculas el primer carácter de cada palabra.
- **lowercase**: Convierte a minúsculas todas las letras del texto.
- **uppercase**: Convierte a mayúsculas todas las letras del texto.
- **none**: No provoca cambios en el texto.

El siguiente ejemplo define reglas para los elementos h1 y p:

```

<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
</head>
<body>
  <h1>Este es un título de nivel 1</h1>
  <p>
    Todo el texto siguiente se encuentra encerrado en el elemento de párrafo y con el objetivo de ver el
    efecto de la propiedad text-indent. La propiedad text-indent podemos inicializarla con un valor
    positivo, como es el caso actual o podemos inicializarla con un valor negativo lo que provocará que el
    texto de la primera línea del párrafo comience en una columna inferior al de todo el bloque.
  </p>
</body>
</html>

```

La cabecera de nivel uno, tiene la siguiente regla:

```

h1 {
  letter-spacing:10px;
  word-spacing:30px;
  text-transform:capitalize;
}

```

Es decir que las letras deben tener una separación de 10 píxeles, las palabras deben estar separadas por 30 píxeles y por último deben disponerse en mayúsculas la primera letra de cada palabra.

Para el elemento p tenemos la siguiente regla:

```

p {
  text-indent:20px;
}

```

Es decir, la primera línea del párrafo deberá imprimirse 20 píxeles a la derecha donde normalmente debería aparecer.

## 9. Herencia de propiedades de estilo

El conjunto de elementos HTML forma en sí un árbol que en su raíz podemos identificar el elemento body del cual se desprenden otros elementos contenidos en esta sección, como podrían ser los elementos h1, h2, h3, h4, h5, h6, p y div luego estas en su interior contienen otros elementos HTML como podrían ser em, strong, pre, etc.

Con ejemplos veamos que hay muchos estilos que se heredan (todos los vistos hasta el momento se heredan), es decir si definimos la propiedad color para el elemento h1, luego si dicho elemento incorpora un texto con el elemento "em" en su interior, la propiedad color del elemento "em" tendrá el mismo valor que la propiedad h1 (es decir el elemento em hereda las propiedades del elemento h1)

Con un ejemplo veremos el resultado de la herencia de propiedades entre los elementos HTML:

```
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
<style>
body {
  color:#0000ff;
  font-family:verdana;
}
</style>
</head>
<body>
<h1>Este es un título de nivel 1 y con el elemento 'em' la palabra:
<em>Hola</em></h1>
<p>Todo este párrafo debe ser de color azul ya que lo hereda del
elemento body.</p>
</body>
</html>
```

**Este es un título de nivel 1 y con el elemento 'em' la palabra: *Hola***

Todo este párrafo debe ser de color azul ya que lo hereda del elemento body.

En este ejemplo hemos definido la siguiente regla para el elemento body:

```
body {
  color:#0000ff;
  font-family:verdana;
}
```

Inicializamos la propiedad color con el valor de azul y la fuente de tipo verdana. Con esto todos los elementos contenidos en el body que no redefinan estas dos propiedades recibirán los valores aquí definidos. En este ejemplo la cabecera de primer nivel es decir h1 y el párrafo tienen como color el azul y la fuente es de tipo verdana.

Ahora bien en muchas situaciones podemos redefinir propiedades para elementos contenidos, veamos como podemos hacer que el texto contenido en los elementos em y p aparezcan de color distinto:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
<style>
body {
  color:#0000ff;
  font-family:verdana;
}
```

**Este es un título de nivel 1 y con el elemento 'em' la palabra: *Hola***

Todo este párrafo debe ser de color gris ya que lo redefine la marca p y no lo hereda de la marca body.

```
em {
  color:#008800;
}
p {
  color:#999999;
}
</style>
</head>
<body>
</body>
<h1>Este es un título de nivel 1 y con el elemento 'em' la palabra:
<em>Hola</em></h1>
<p>Todo este párrafo debe ser de color gris ya que lo redefine la
marca p y no lo hereda de la marca body.</p>
</html>
```

Ahora hemos definido tres reglas, la primera igual que el problema anterior, define la propiedad color en azul y la fuente de tipo verdana para el elemento body:

```
body {
  color:#0000ff;
  font-family:verdana;
}
```

La segunda regla define la propiedad color en verde para el elemento em, con esto no heredará el color azul del elemento body (que es el elemento padre):

```
em {
  color:#008800;
}
```

Algo similar hacemos con el elemento p para indicar que sea de color gris:

```
p {
  color:#999999;
}
```

Pero podemos ver que todos los elementos heredan la fuente verdana ya que ninguna lo sobrescribe.

## 10. Definición de un estilo en función del contexto

Otro recurso que provee las hojas de estilo en cascada (CSS) es la definición de un estilo para un elemento HTML siempre y cuando la misma esté contenida por otro elemento determinado. Sólo en ese caso el estilo para dicho elemento se activará.

Supongamos que queremos que cuando disponemos un texto encerrado por el elemento strong dentro de un párrafo el color del mismo sea verde. Pero si el elemento strong está contenida por el elemento h1 el color debe ser rojo, luego la sintaxis del problema es:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
<style>
p strong{
  color:#0000ff;
}
h1 strong{
  color:#ff0000;
}
</style>
</head>
<body>
<h1>Aquí tenemos un título de nivel uno, luego un bloque con
el elemento strong debe aparecer <strong>así</strong></h1>
<p>
Luego si escribimos un párrafo y encerramos un bloque con el
elemento strong
debe aparecer <strong>así</strong>
</p>
</body>
</html>
```

**Aquí tenemos un título de nivel uno, luego un bloque con el elemento strong debe aparecer así**

Luego si escribimos un párrafo y encerramos un bloque con el elemento strong debe aparecer así

Es importante analizar la sintaxis para la definición de estilos en función del contexto.

Tenemos:

```
p strong{
  color:#0000ff;
}
```

Estamos diciendo que todos los bloques de la página que contengan el elemento strong y que estén contenidas por el elemento p (párrafo) se pinten de azul. Si bien hay dos bloques en la página que están encerrados por el elemento strong, solo uno se pinta de color azul, el otro bloque se pinta de color rojo, ya que tenemos:

```
h1 strong{
  color:#ff0000;
}
```

Es decir, activar el color rojo para el contenido encerrado por el elemento strong, siempre y cuando se encuentre contenido por el elemento h1.

No confundir con la sintaxis para definir reglas de estilo a dos elementos que se hace separando por coma los elementos HTML:

```
h1,strong{
  color:#ff0000;
}
```

Es decir que el texto contenido por los elementos h1 y strong deben aparecer de color rojo.

Se pueden definir estilos en función del contexto, con mayor precisión, como por ejemplo:

```
div h1 em {
  color:#ff0000;
}
```

Con esto estamos diciendo que se debe pintar de color rojo el texto contenido por el elemento em siempre y cuando esté contenida en un elemento h1 y esta a su vez dentro de un div.

La página completa queda codificada de la siguiente forma:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
<style>
div h1 em {
  color:#ff0000;
}
</style>
</head>
<body>
<div>
<h1>Este es un titulo de nivel 1 dentro de un <em>div</em></h1>
</div>
<h1>Este es un titulo de nivel 1 fuera de un <em>div</em></h1>
</body>
</html>
```

**Este es un titulo de nivel 1 dentro de un *div***

**Este es un titulo de nivel 1 fuera de un *div***

## 11. Hojas de estilo en un archivo externo

Hasta ahora hemos visto la definición de estilos a nivel de elemento HTML y la definición de estilos a nivel de página. Dijimos que la primera forma es muy poco recomendada y la segunda es utilizada cuando queremos definir estilos que serán empleados sólo por esa página.

Ahora veremos que la metodología más empleada es la definición de una hoja de estilo en un archivo separado que deberá tener la extensión ".css".

Este archivo contendrá las reglas de estilo (igual como las hemos visto hasta ahora) pero estarán separadas del archivo HTML.

La ventaja fundamental es que con esto podemos aplicar las mismas reglas de estilo a parte o a todas las páginas del sitio web. Veremos que esto será muy provechoso cuando necesitemos hacer cambios de estilo (cambiando las reglas de estilo de este archivo estaremos cambiando la apariencia de múltiples páginas del sitio).

También tiene como ventaja que al programador le resulta más ordenado tener lo referente a HTML en un archivo y las reglas de estilo en un archivo aparte.

Otra ventaja es que cuando un navegador solicita una página, se le envía el archivo HTML y el archivo CSS, quedando guardado este último archivo en la caché de la computadora, con lo cual, en las sucesivas páginas que requieran el mismo archivo de estilos, ese mismo archivo se rescata de la caché y no requiere que el servidor web se lo reenvíe (ahorrando tiempo de transferencia)

Veamos la primera página HTML que tiene asociada una hoja de estilo en un archivo externo. El archivo HTML es (pagina.html):

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<h1>Definición de hojas de estilo en un archivo externo.</h1>
<p>
Hasta ahora hemos visto la definición de estilos a nivel de elemento
HTML
y la definición de estilos a nivel de página. Dijimos que la primera
forma es muy poco recomendable y la segunda es utilizada cuando
queremos definir estilos que serán empleados solo por esa página.
Ahora veremos que la metodología más empleada es la definición
de una hoja de estilo en un archivo separado que deberá tener la
extensión
css.
</p>
</body>
</html>
```

El archivo que tiene las reglas de estilo es (estilos.css):

```
body {
  background-color:#eafadd;
}
h1 {
  color:#0000cc;
  font-family:times new roman;
  font-size:25px;
  text-align:center;
  text-decoration:underline;
}
p {
  color:#555555;
```



```
font-family:verdana;
text-align:justify;
}
```

Como podemos observar, para indicar el archivo de estilos externo, debemos agregar en la cabecera (head) del archivo HTML la siguiente marca:

```
<link rel="StyleSheet" href="estilos.css" type="text/css">
```

La propiedad href hace referencia al archivo externo que afectará la presentación de esta página. En la propiedad type, indica al navegador cual es el formato de archivo. El atributo rel se usa para definir la relación entre el archivo enlazado y el documento HTML.

Sólo queda probar esta página funcionando:

## Definición de hojas de estilo en un archivo externo.

Hasta ahora hemos visto la definición de estilos a nivel de elemento HTML y la definición de estilos a nivel de página. Dijimos que la primera forma es muy poco recomendable y la segunda es utilizada cuando queremos definir estilos que serán empleados solo por esa página. Ahora veremos que la metodología más empleada es la definición de una hoja de estilo en un archivo separado que deberá tener la extensión css.

De ahora en adelante nos acostumbraremos a trabajar con hojas de estilo definidas en un archivo externo, que es en definitiva la forma más común de desarrollar un sitio web aplicando CSS.

## 12. Definición de estilos por medio de clases

En muchas situaciones una regla de estilo puede ser igual para un conjunto de elementos HTML, en esos casos conviene plantear una regla de estilo con un nombre genérico que posteriormente se puede aplicar a varios elementos de HTML.

Para el planteo de una regla de estilo por medio de una clase creamos un nombre de clase y le antecedemos el carácter punto:

```
.resaltado{
  color:#000000;
  background-color:#ffff00;
  font-style:italic;
}
```

Luego para asignar dicha regla a un elemento HTML definimos la propiedad **class** al elemento que necesitamos fijarle este estilo:

```
<h1 class="resaltado">Este elemento h1 aparece con la clase
resaltado</h1>
```

Podemos especificar la clase "resaltado" a todos los elementos HTML que necesitemos aplicarle dichas reglas de estilo.

Veamos un ejemplo, la pagina.html es:

```
<!DOCTYPE html>
<html>
<head>
```

```

<title>Problema</title>
<meta charset="UTF-8">
<link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<h1 class="resaltado">Titulo de nivel 1</h1>
<p>
Este párrafo muestra el resultado de aplicar la clase .resaltado en
la última
<span class="resaltado">palabra.</span>
</p>
</body>
</html>

```

La hoja de estilo externa (estilos.css) es:

```

body {
  background-
color:#eeeeee;
}
.resaltado{
  color:#000000;
  background-color:#ffff00;
  font-style:italic;
}

```

***Titulo de nivel 1***

Este párrafo muestra el resultado de aplicar la clase .resaltado en la última ***palabra.***

La sintaxis para definir una clase aplicable a cualquier elemento HTML es:

```

.resaltado{
  color:#000000;
  background-color:#ffff00;
  font-style:italic;
}

```

Es decir, la inicializamos con el carácter punto y seguidamente un nombre de clase. Dentro definimos las reglas de estilo como hemos venido trabajando normalmente.

El nombre de la clase no puede comenzar con un número.

Luego, para indicar que un elemento sea afectado por esta regla:

```

<h1 class="resaltado">Titulo de nivel 1</h1>

```

Es decir, agregamos la propiedad class y le asignamos el nombre de la clase (sin el punto).

Podemos inicializar tantos elementos HTML con esta regla como necesitemos:

```

<p>
Este parrafo muestra el resultado de aplicar la clase. resaltado en
la última <span class="resaltado">palabra.</span>
</p>

```

Aquí definimos la propiedad class al elemento span y le asignamos la misma clase aplicada anteriormente al elemento h1.

## 13. Definición de estilos por medio de id

---

La diferencia fundamental en la definición de un estilo por medio de id con respecto a las clases, es que sólo podremos aplicar dicho estilo a un solo elemento HTML dentro de la página, ya que todos los id que se definen en una página HTML deben tener nombres distintos.

La sintaxis para definir un estilo por medio de id es:

```
#cabecera {  
  font-family:Times New Roman;  
  font-size:30px;  
  text-align:center;  
  color:#0000ff;  
  background-color:#bbbbbb;  
}
```

Es decir, utilizamos el carácter numeral (#) para indicar que se trata de un estilo de tipo **id**.

Luego, sólo un elemento HTML dentro de una página puede definir un estilo de este tipo:

```
<div id="cabecera">
```

Hay que subrayar que sólo un elemento HTML puede definir la propiedad id con el valor de cabecera.

Los dos archivos completos del ejemplo entonces quedan (pagina.html):

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Problema</title>  
  <meta charset="UTF-8">  
  <link rel="StyleSheet" href="estilos.css" type="text/css">  
</head>  
<body>  
  <div id="cabecera">  
    <h1>Título de la cabecera</h1>  
  </div>  
</body>  
</html>
```

Y el archivo de estilos (estilos.css) es:



## 14. Propiedades del borde de un elemento

---

Debemos ahora hacernos la idea de que todo elemento que se crea dentro de una página HTML genera una caja. Tendremos que definir ciertas propiedades relacionadas al borde de un elemento HTML.

Imaginemos los controles que hemos creado h1, h2, h3, p, em, etc. si fijamos la propiedad **background-color** veremos que el contenido se encuentra dentro de un rectángulo.

Podemos acceder a las propiedades del borde de ese rectángulo mediante las hojas de estilo CSS; las propiedades más importantes a las que tenemos acceso son:

- **border-width**
- **border-style**
- **border-color**

Veamos un ejemplo que inicialice estas propiedades:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<p class="pregunta">¿Quién descubrió América
y en que año fue?</p>
<p class="respuesta">Colón en 1492</p>
</body>
</html>
```

La hoja de etilo es:

```
.pregunta {
  background-
color:#ffff00;
  border-width:1px;
  border-style:solid;
  border-color:#000000;
}
.respuesta {
  border-width:1px;
  border-style:dashed;
  border-color:#000000;
}
```

¿Quién descubrió América y en que año fue?

Colón en 1492

Como podemos ver, hemos definido dos clases ".pregunta" que inicializa el color de fondo en amarillo y luego define el ancho del borde en un pixel, el estilo es sólido y el color de la línea de borde es negro.

Luego recordar que para indicar que un elemento tenga este estilo debemos inicializar la propiedad class del elemento HTML respectivo:

```
<p class="pregunta">¿Quién descubrió América y en que año fue?</p> Al
segundo estilo definido lo hemos hecho con la clase ".respuesta"
.respuesta {
  border-width:1px;
  border-style:dashed;
```

```
border-color:#000000;  
}
```

En ésta hemos cambiado el estilo de borde por el valor dashed.

Disponemos de los siguientes estilos de borde:

**none, hidden, dotted, dashed, solid, double, groove, ridge, inset, y outset.**

## 15. Otras propiedades del borde de un elemento

Como vimos en el apartado anterior tenemos propiedades que nos permiten fijar el grosor, estilo y color del borde de un elemento HTML. Pero podemos ir un paso más allá, CSS nos permiten modificar independientemente cada uno de los cuatro bordes del rectángulo.

Contamos con las siguientes propiedades:

- **border-top-width**
- **border-right-width**
- **border-bottom-width**
- **border-left-width**
- **border-top-style**
- **border-right-style**
- **border-bottom-style**
- **border-left-style**
- **border-top-color**
- **border-right-color**
- **border-bottom-color**
- **border-left-color**

Un ejemplo inicializando estas propiedades:

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Problema</title>  
  <meta charset="UTF-8">  
  <link rel="StyleSheet" href="estilos.css" type="text/css">  
</head>  
<body>  
<h1 class="titulopagina">Tutorial de CSS desde cero</h1>  
</body>  
</html>
```

y el archivo de estilos:

```
.titulopagina {  
  background-color:#ffffcc;  
  text-align:center;  
  font-family:verdana;  
  font-size:40px;  
  
  border-top-width:1px;  
  border-right-width:3px;
```

**Tutorial de CSS desde  
cero**

```
border-bottom-width:3px;
border-left-width:1px;

border-top-style:dotted;
border-right-style:solid;
border-bottom-style:solid;
border-left-style:dotted;

border-top-color:#ffaa00;
border-right-color:#ff0000;
border-bottom-color:#ff0000;
border-left-color:#ffaa00;
}
```

Es decir, esta metodología de inicializar el borde de un elemento HTML, tiene utilidad si los mismos varían en grosor, estilo o color.

Veremos más adelante que hay otras formas de inicializar los bordes de los elementos que reducen el texto a escribir.

## 16. Propiedades del padding de un elemento HTML

El padding (almohadilla) suma espacio entre el contenido del elemento HTML (por ejemplo dentro del elemento párrafo el texto propiamente dicho es el contenido) y el borde (recordar propiedad border)

Podemos configurar los 4 en caso que tengan el mismo valor accediendo a la propiedad **padding** o podemos configurar en forma independiente cada lado:

- **padding-top**
- **padding-right**
- **padding-bottom**
- **padding-left**

Veamos un ejemplo, la pagina.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<pre class="codigofuente">
public class Rectangulo{
  //atributos
  int alto;
  int ancho;
  boolean relleno;
  //métodos
  public int devolverArea(){
    return alto*ancho;
```

```

    }
    public void rellenar(boolean r){
        relleno=r;
    }
    public void cambiarTamano(int an, int al){
        ancho=an;
        alto=al;
    }
    public Rectangulo() { // constructor
        alto=20;
        ancho=10;
        relleno=false;
    }
} //fin clase Rectangulo
</pre>
</body>
</html>

```

El archivo estilos.css es:

```

.codigofuente {
    font-size:12px;
    background-color:#ffffcc;
    border-width:1px;
    border-style:dotted;
    border-color:#ffaa00;

    padding:20px;
}

```

```

public class Rectangulo{
    //atributos
    int alto;
    int ancho;
    boolean relleno;
    //métodos
    public int devolverArea(){
        return alto*ancho;
    }
    public void rellenar(boolean r){
        relleno=r;
    }
    public void cambiarTamano(int an, int al){
        ancho=an;
        alto=al;
    }
    public Rectangulo() { // constructor
        alto=20;
        ancho=10;
        relleno=false;
    }
} //fin clase Rectangulo

```

Tenemos 20px de separación entre el borde y el contenido del elemento "pre".

Con el elemento HTML "pre", se respetan los espacios y retornos de carro que hayamos puesto en el texto fuente. Este estilo de texto es muy adecuado cuando queremos mostrar el código fuente de un programa.

## 17. Propiedades del margen de un elemento HTML

Otra serie de propiedades relacionadas al contorno de un elemento HTML son las propiedades relacionadas al margen:

- **margin**
- **margin-top**
- **margin-right**
- **margin-bottom**
- **margin-left**

El margen está después del borde. El margen separa un elemento HTML de otro elemento HTML dentro de la página.

Veamos un ejemplo, la página HTML muestra dos párrafos con cero pixeles de margen:

```

<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<p>Primer párrafo</p>
<p>Segundo párrafo</p>
</body>
</html>

```

La hoja de estilo:

```

p {
  background-color:#fffffaa;
  margin:0px;
  border-width:1px;
  border-style:solid;
  border-color:#ff0000;
}

```

Primer párrafo
Segundo párrafo

Hay que tener en cuenta que cuando dos elementos HTML uno debajo del otro hay especificado márgenes el resultado final del margen inferior de uno y el superior de otro no se suman. Por ejemplo si un elemento tiene margen inferior de 10 y el elemento que se encuentra debajo de este tiene margen de 20px luego el margen total entre estos dos elementos es de 20px (es el mayor de los dos márgenes y no 30px que es la suma)

Pruebe modificar el valor para la propiedad margin y vea el resultado de la página.

El modelo final de caja se puede resumir con esta imagen:



Es común que cuando el margen sea cero no se indique la unidad de medida:  
`margin:0;`



## 18. Propiedades relacionadas a listas

Las listas se utilizan para enumerar una serie de elementos, se utiliza el elemento HTML **ul** (Unordered List), y cada ítem de la lista con el elemento HTML **li** (List Item).

Las CSS nos permiten configurar las listas por medio de tres propiedades:

- **list-style-type**, puede asignársele los valores siguientes:
  - none
  - disc
  - circle
  - square
  - decimal
  - decimal-leading-zero
  - lower-roman
  - upper-roman
  - lower-alpha
  - upper-alpha
- **list-style-position**, que puede tener como valores:
  - inside
  - outside
- **list-style-image**, cuyos valores pueden ser:
  - none
  - url

Veamos un ejemplo que prueba todos los valores posibles que puede tomar la propiedad list-style-type:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<ul class="vacio">
  <li>Brasil</li>
  <li>Uruguay</li>
  <li>Argentina</li>
</ul>
<ul class="circulorelleno">
  <li>Brasil</li>
  <li>Uruguay</li>
  <li>Argentina</li>
</ul>
<ul class="circulovacio">
  <li>Brasil</li>
  <li>Uruguay</li>
  <li>Argentina</li>
</ul>
```

```

<ul class="cuadrado">
  <li>Brasil</li>
  <li>Uruguay</li>
  <li>Argentina</li>
</ul>
<ul class="decimal">
  <li>Brasil</li>
  <li>Uruguay</li>
  <li>Argentina</li>
</ul>
<ul class="romanominuscula">
  <li>Brasil</li>
  <li>Uruguay</li>
  <li>Argentina</li>
</ul>
<ul class="romanomayuscula">
  <li>Brasil</li>
  <li>Uruguay</li>
  <li>Argentina</li>
</ul>
<ul class="letrasminusculas">
  <li>Brasil</li>
  <li>Uruguay</li>
  <li>Argentina</li>
</ul>
<ul class="letrasmayusculas">
  <li>Brasil</li>
  <li>Uruguay</li>
  <li>Argentina</li>
</ul>
</body>
</html>

```

Luego la hoja de estilo es:

```

.vacio{
  list-style-type:none;
}
.circulorelleno{
  list-style-type:disc;
}
.decimal{
  list-style-type:decimal;
}
.romanominuscula{
  list-style-type:lower-roman;
}

```

```
.romanomayuscula{
  list-style-type:upper-roman;
}
.circulovacio{
  list-style-type:circle;
}
.cuadrado{
  list-style-type:square;
}
.letrasminusculas{
  list-style-type:lower-alpha;
}
.letrasmayusculas{
  list-style-type:upper-alpha;
}
```

Hemos definido un conjunto de clases para aplicarlas a las listas de HTML.

```
Brasil
Uruguay
Argentina

• Brasil
• Uruguay
• Argentina

○ Brasil
○ Uruguay
○ Argentina

■ Brasil
■ Uruguay
■ Argentina

1. Brasil
2. Uruguay
3. Argentina

i. Brasil
ii. Uruguay
iii. Argentina

I. Brasil
II. Uruguay
III. Argentina

a. Brasil
b. Uruguay
c. Argentina

A. Brasil
B. Uruguay
C. Argentina
```

## 19. Propiedades del fondo (background)

Hasta ahora hemos probado y utilizado la propiedad background-color para fijar el color de fondo del contenido de un elemento HTML (body, h1, h2, p, etc.).

Hay otras propiedades relacionadas al fondo que nos permiten, entre otras cosas, disponer un archivo de imagen. Las propiedades relacionadas al background y sus valores son:

- **background-color**
- **background-image**
- **background-repeat**
- **background-position**
- **background-attachment**

Veamos un ejemplo de disponer una imagen sobre el fondo de la página:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
</body>
</html>
```

La hoja de estilo queda:

```
body {
  background-image:url(fondo.jpg);
}
```



La única propiedad que hemos inicializado es **background-image** indicando el nombre del archivo de imagen a mostrar.

La imagen se repite en *X* e *Y* hasta llenar la página por completo, ya que por defecto **background-repeat** está inicializada con el valor **repeat**, podemos probar a modificar el estilo primero con:

```
body {  
    background-image:url(fondo.jpg);  
    background-repeat:repeat-x;  
}
```

Luego con:

```
body {  
    background-image:url(fondo.jpg);  
    background-repeat:repeat-y;  
}
```

Y por último:

```
body {  
    background-image:url(fondo.jpg);  
    background-repeat:no-repeat;  
}
```

Tener en cuenta que podemos aplicar una imagen a otros elementos de HTML (h1, h2, h3, p, etc.)

Con la última propiedad **background-position** podemos indicar la posición de la imagen según los siguientes valores:

**top left, top center, top right, center left, center center, center right, bottom left, bottom center, bottom right, x-% y-% y x-pos y-pos.**

Para que tenga sentido esta propiedad debemos inicializar la propiedad **background-repeat** con el valor **no-repeat**.

Por ejemplo:

```
body {  
    background-image:url(fondo.jpg);  
    background-repeat:no-repeat;  
    background-position:20% 50%;  
}
```

Dispone la imagen 20% avanzando desde la izquierda y 50% avanzando desde arriba.

La siguiente regla:

```
body {  
    background-image:url(fondo.jpg);  
    background-repeat:no-repeat;  
    background-position:400px 10px;  
}
```

Dispone la imagen 400 pixeles desde la derecha y 10 píxeles desde arriba.

La regla:

```
body {  
    background-image:url(fondo.jpg);  
    background-repeat:no-repeat;  
    background-position:top right;  
}
```

Dispone la imagen en la parte superior a la derecha.

## 20. Selector universal \*

---

Si queremos inicializar las propiedades de todos los elementos HTML podemos utilizar el selector universal. Utilizamos el carácter asterisco (\*) para hacer referencia a este selector.

Ejemplo:

```
* {  
    margin:0;  
    padding:0;  
    color:#ff0000;  
}
```

Con dicho selector estamos especificando que todos los elementos HTML tendrán un margin y padding de cero y los textos serán de color rojo.

Veamos un ejemplo dentro de una página:

```
<!DOCTYPE html>  
<html>  
<head>  
    <title>Problema</title>  
    <meta charset="UTF-8">  
    <link rel="StyleSheet" href="estilos.css" type="text/css">  
</head>  
<body>  
<h1>Título de nivel 1</h1>  
<h2>Título de nivel 2</h2>  
<p>Esto esta dentro de un párrafo</p>  
</body>  
</html>
```

Luego en la hoja de estilo definimos:

```
* {  
  color:#0000aa;  
  margin:0;  
  padding:0;  
}
```

# Título de nivel 1

## Título de nivel 2

Esto esta dentro de un párrafo

Esto significa que todos los elementos se imprimen de color azul con cero píxeles de margin y padding, salvo que otra regla lo cambie, Imaginemos si definimos `h1 { color:#ff0000}` significa que tiene prioridad esta regla.

En realidad, en forma tácita lo hemos estado utilizando, cuando definimos una clase sin indicar el tipo de elemento HTML donde actuará:

```
.pregunta {  
  background-color:#ffff00;  
  border-width:1px;  
  border-style:solid;  
  border-color:#000000;  
}
```

Podemos expresar la regla anterior perfectamente como:

```
*.pregunta {  
  background-color:#ffff00;  
  border-width:1px;  
  border-style:solid;  
  border-color:#000000;  
}
```

Es decir que podemos asignar esta regla a cualquier elemento HTML.

Esto nos permite comprender, cuando definimos una regla que sólo se puede utilizar en un sólo tipo de elemento HTML:

```
p.pregunta {  
  background-color:#ffff00;  
  border-width:1px;  
  border-style:solid;  
  border-color:#000000;  
}
```

Esta regla sólo se puede utilizar dentro de elementos de tipo párrafo.

## 21. Pseudoclases

---

Las pseudoclases son unas clases especiales, que se refieren a algunos estados del elemento HTML, las que se utilizan fundamentalmente son las que se aplican al elemento `<a>` (enlace).

La sintaxis varía con respecto al concepto de clase visto anteriormente ya que se tratan de pseudoclases predefinidas:

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

Es decir, separamos el nombre del elemento HTML con dos puntos.

Para el elemento HTML <a> tenemos cuatro pseudoclases fundamentales:

- **link**. Enlace sin ingresar.
- **visited**. Enlace presionado.
- **hover**. Enlace que tiene la flecha del mouse encima.
- **active**. Es la que tiene foco en ese momento (prueba a tocar la tecla tab)

Es importante hacer notar que el orden en que definimos las pseudoclases es fundamental para su funcionamiento (debe respetarse el orden: link-visited-hover-active)

Este ejemplo es muy sencillo para ver el paso en los distintos estados que puede tener un enlace:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<a href="http://www.google.com">Google</a>
<a href="http://www.yahoo.com">Yahoo</a>
<a href="http://www.bing.com">Bing</a>
</body>
</html>
```

La hoja de estilo es:

```
a:link{
  background-color:#00ff00;
  color:#ff0000;
}
a:visited{
  background-color:#000000;
  color:#ffffff;
}
a:hover{
  background-color:#ff00ff;
  color:#ffffff;
}
a:active{
  background-color:#ff0000;
  color:#ffff00;
}
```



Apenas ejecutes la página los tres enlaces deben aparecer de color rojo con fondo verde:

```
a:link{
  background-color:#00ff00;
  color:#ff0000;
}
```

Si presionamos la tecla tab podremos ver que el enlace que tiene foco aparece de color amarillo con fondo rojo:

```
a:active{
  background-color:#ff0000;
  color:#ffff00;
}
```

Si pasamos la flecha del mouse sobre algún enlace veremos que aparece de color blanco con fondo lila:

```
a:hover{
  background-color:#ff00ff;
  color:#ffffff;
}
```

Por último todos los enlaces que hayamos hecho clic deberán aparecer de color blanco con fondo negro:

```
a:visited{
  background-color:#000000;
  color:#ffffff;
}
```

## 21.1. Eliminar el subrayado a un enlace "a" por medio de las pseudoclases

Otra actividad común en gran cantidad de sitios es eliminar el subrayado a los enlaces con el objetivo que la página tenga mejor estética. A esto lo podemos hacer configurando las pseudoclases:

```
a:link {
  text-decoration:none;
}
a:visited {
  text-decoration:none;
}
```

Accedemos a las pseudoclases link y visited e inicializamos la propiedad text-decoration con el valor none.

Probamos la solución en esta página:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<a href="http://www.google.com">Google</a>
<a href="http://www.yahoo.com">Yahoo</a>
<a href="http://www.bing.com">Bing</a>
</body>
</html>
```



La hoja de estilo es:

```
a:link {
  text-decoration: none;
}
a:visited {
  text-decoration: none;
}
```

Google Yahoo Bing

Es decir, configuramos la propiedad text-decoration con el valor none, por defecto está configurada con el valor underline.

Tener en cuenta que podemos agrupar la regla de esta forma:

```
a:link, a:visited {
  text-decoration: none;
}
```

## 21.2. Creación de un menú vertical configurando las pseudoclasas.

Un recurso muy útil es disponer un menú en una página, si no requerimos uno muy elaborado podemos resolverlo utilizando sólo CSS y HTML (en otros casos se requiere además de JavaScript)

Vamos a implementar uno muy sencillo que requiere agrupar en un div una serie de párrafos que contienen un hipervínculo cada uno. Cuando la flecha del mouse se encuentra sobre el hipervínculo cambiamos el color del fondo y la letra del hipervínculo. Un ejemplo de esto es:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<div id="menu">
<p><a href="http://www.google.com">Google</a></p>
<p><a href="http://www.yahoo.com">Yahoo</a></p>
<p><a href="http://www.bing.com">Bing</a></p>
<p><a href="http://www.ask.com">ASK</a></p>
</div>
</body>
</html>
```

La hoja de estilo asociada a esta página es:

```

#menu {
  font-family: Arial;
}

#menu p {
  margin:0px;
  padding:0px;
}

#menu a {
  display: block;
  padding: 3px;
  width: 160px;
  background-color: #f7f8e8;
  border-bottom: 1px solid #eeeeee;
  text-align:center;
}

#menu a:link, #menu a:visited {
  color: #ff0000;
  text-decoration: none;
}

#menu a:hover {
  background-color: #336699;
  color: #ffffff;
}

```



Podemos decir que definimos un estilo por medio de un Id llamado “menu”. La regla para este Id:

```

#menu {
  font-family: Arial;
}

```

La segunda regla aparece el concepto de selectores descendientes (ocurre cuando un elemento HTML se encuentra contenido en otro) en este caso se seleccionan todos los párrafos que se encuentren dentro del div que define el id #menu (con esto logramos que si hay otros párrafos en el documento HTML no se le apliquen esta regla):

```

#menu p {
  margin:0px;
  padding:0px;
}

```

Estamos indicando que todos los párrafos contenidos en #menu deben tener cero en margin y padding.

Luego las anclas "a" dentro de #menu definen las siguientes propiedades (si hay otros enlaces dentro de la página no se le aplica este estilo ya que solo se aplican a las "a" que descienden de #menu):

```

#menu a {
  display: block;
}

```

```
padding: 3px;
width: 160px;
background-color: #f7f8e8;
border-bottom: 1px solid #eeeeee;
text-align:center;
}
```

El valor **block** para la propiedad **display** permite que el ancla ocupe todo el espacio del párrafo, indistintamente del largo del hipervínculo.

Otra propiedad nueva es **width**, esta fija el tamaño máximo que puede tener el hipervínculo antes de provocar un salto de línea.

Por último inicializamos las pseudoclases:

```
#menu a:link, #menu a:visited {
  color: #ff0000;
  text-decoration: none;
}
```

```
#menu a:hover {
  background-color: #336699;
  color: #ffffff;
}
```

Estamos definiendo el mismo color de texto para los vínculos seleccionados como aquellos que no han sido seleccionados:

```
#menu a:link, #menu a:visited {
  color: #ff0000;
```

Por último cambiamos el color de fondo de la opción cuando se tiene la flecha del mouse encima:

```
#menu a:hover {
  background-color: #336699;
```

### 21.3. Creación de un menú horizontal con una lista

Otro estilo de menú muy común es donde las opciones se encuentran una al lado de otra.

Utilizaremos una lista no ordenada con una serie de ítems que contienen los enlaces.

Como elementos de una lista se muestran uno debajo de otro debemos inicializar la propiedad **float** con el valor **left** para que los ítems se ubiquen uno al lado de otro.

Veamos el código para la implementación de un menú horizontal:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
```

```
<ul id="menuhorizontal">
<li><a href="http://www.google.com">Google</a></li>
<li><a href="http://www.yahoo.com">Yahoo</a></li>
<li><a href="http://www.bing.com">Bing</a></li>
<li><a href="http://www.ask.com">Ask</a></li>
</ul>
</body>
</html>
```

El archivo de estilos es:

```
#menuhorizontal {
  margin:0;
  padding:0;
  list-style-type:none;
}
#menuhorizontal a {
  width:100px;
  text-decoration:none;
  text-align:center;
  color:#ff0000;
  background-color:#f7f8e8;
  padding:3px 5px;
  border-right:1px solid blue;
  display:block;
}

#menuhorizontal li {
  float:left;
}

#menuhorizontal a:hover {
  background-color:#336699;
}
```



Inicializamos el margin y padding con cero y el estilo de la lista con none para que no aparezcan los círculos de cada ítem:

```
#menuhorizontal {
  margin:0;
  padding:0;
  list-style-type:none;
}
```

En los enlaces del menú inicializamos la propiedad display con el valor block para que el enlace tenga efecto en todo el rectángulo:

```
#menuhorizontal a {
  width:100px;
  text-decoration:none;
  text-align:center;
  color:#ff0000;
  background-color:#f7f8e8;
  padding:3px 5px;
  border-right:1px solid blue;
```

```
display:block;
}
```

Para que los enlaces aparezcan uno al lado de otro inicializamos la propiedad float con el valor left:

```
#menuhorizontal li {
  float:left;
}
```

Finalmente la pseudoclase para indicar el color de fondo del enlace cuando la flecha del mouse pasa por encima es:

```
#menuhorizontal a:hover {
  background-color:#336699;
}
```

## 22. Propiedades de la dimensión de un objeto en la página

Disponemos de dos propiedades fundamentales que nos permiten fijar el ancho y el alto de un elemento HTML:

- **Width**
- **height**

Cuando no se fija el ancho de un elemento HTML la propiedad toma el valor por defecto que es "**auto**" por lo que el navegador tiene que calcular el ancho teniendo en cuenta el contenido del elemento y el espacio disponible.

Por el momento hemos visto solo la medida en píxeles y porcentajes (luego veremos que podemos utilizar otras unidades de medida para fijar el ancho y el alto de un elemento)

Otras cuatro propiedades relacionadas con el ancho y el alto de un elemento HTML son:

- **min-width**. Fija el ancho mínimo que puede tomar el elemento HTML (si reducimos el ancho del navegador a un valor menor a este veremos que aparece la barra de scroll en forma horizontal, es decir el elemento no puede tomar un valor menor a este)
- **max-width**. Fija el ancho máximo que puede tomar el elemento HTML (es útil en los casos de aquellos elementos HTML que muestran texto y no queremos que tome una dimensión muy grande ya que esto hace complejo la lectura)
- **min-height**. Fija el alto mínimo.
- **max-height**. Fija el alto máximo.

Veamos un ejemplo:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<div id="cabecera">
```

```
Blog del Programador
</div>
</body>
</html>
```

Solamente hemos definido un div donde mostramos la cabecera de una página.

La hoja de estilo definida:

```
#cabecera {
  width:100%;
  height:100px;
  background-color:#ffee00;
  color:#0000aa;
  text-align:center;
  font-family:Times New Roman;
  font-size:50px;
  font-weight:bold;
}
```

**Blog del Programador**

La propiedad **width** la inicializamos con el valor 100%, lo que significa que ocupará todo el ancho de la página (podemos inicializarlo en píxeles si lo necesitamos). Luego a la propiedad **height** la inicializamos en 100 píxeles.

El resto de propiedades son las ya vistas en conceptos anteriores.

Es decir que las propiedades width y height nos permiten dar una dimensión al elemento HTML ya sea con valores absolutos indicados en píxeles o relativos indicados por porcentajes.

## 23. Unidades de medida (px, rem, em, cm, mm etc.)

Hasta ahora siempre que hemos especificado tamaños de letra, margin, padding, border etc. lo hemos hecho a través de píxeles o porcentajes. Esto es debido a que la forma más sencilla de imaginar un tamaño es el tamaño de un pixel del monitor. Veremos que hay varias unidades de medida para indicar tamaños y que algunas son más indicadas para algunas situaciones que otras.

Disponemos de las siguientes unidades de medida:

- **px.** Píxeles.
- **rem.** Tamaño de la fuente respecto al elemento raíz del documento.
- **em.** Altura de la fuente por defecto.
- **ex.** Altura de la letra x minúscula.
- **In.** Pulgadas.
- **cm.** Centímetros.
- **mm.** Milímetros.
- **pt.** Puntos, 1 punto es lo mismo que 1/72 pulgadas.
- **pc.** Picas, 1 pc es lo mismo que 12 puntos.
- **%.** Porcentaje.

## Unidades con longitudes relativas

De todas las medidas enunciadas en un principio las unidades con longitudes relativas más utilizadas son, rem, em y %.

Las unidades de longitud relativa escalan mejor en móviles, tablets y monitores.

Por ejemplo el framework de CSS Bootstrap 4 utiliza en forma masiva la unidad de medida "rem", si abrimos su código fuente veremos que se inicializan más de 600 propiedades utilizando esta medida.

Veamos como se calculan las medidas de cada elemento cuando utilizamos la unidad "rem":

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
  <h1>Título nivel 1</h1>
  <h2>Título nivel 2</h2>
  <h3>Título nivel 3</h3>
  <h4>Título nivel 4</h4>
  <h5>Título nivel 5</h5>
  <h6>Título nivel 6</h6>
</body>
</html>
```

El archivo de estilos es:

```
html {
  font-size: 10px;
}
h1 {
  font-size: 2.5rem;
}

h2 {
  font-size: 2rem;
}

h3 {
  font-size: 1.75rem;
}

h4 {
  font-size: 1.5rem;
}

h5 {
  font-size: 1.25rem;
}
```

**Título nivel 1**

**Título nivel 2**

**Título nivel 3**

**Título nivel 4**

**Título nivel 5**

**Título nivel 6**

```
h6 {  
  font-size: 1.05rem;  
}
```

Para entender las medidas de los elementos h1, h2, h3 etc. debemos hacer la siguiente operación, como el elemento "html" tiene una medida de 10px luego el elemento "h1" se calcula multiplicando 10px por 2.5 (es decir que los elementos h1 tienen una medida de 25px)

La medida definida en el elemento raíz que es "html" afecta a todos los elementos que utilizan la medida "rem".

Probemos a modificar la medida font-size del elemento "html" por el valor 20px y veremos que se ha afectado a todos los elementos que utilizan "rem".

Podemos, después borrar la regla que definimos para el elemento "html":

```
html {  
  font-size: 20px;  
}
```

En dicho caso el navegador define un valor por defecto para la propiedad font-size.

## 24. Formas para indicar el color

---

Hasta ahora hemos visto que para asignar el color utilizamos tres valores hexadecimales (rojo, verde, azul) precedido por el carácter #:

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

Ahora veremos otras sintaxis para indicar el color:

```
background-color:rgb(255,0,0);
```

Utilizando la función rgb pasando como parámetro la cantidad de rojo, verde y azul en formato decimal (un valor entre 0 y 255).

También con la función rgb podemos indicar un porcentaje entre 0% y 100% para cada componente:

```
background-color:rgb(100%,0%,0%)
```

Por último en algunas situaciones podemos utilizar una sintaxis reducida para ciertos valores:

```
background-color:#ffaaff;
```

Lo podemos indicar con esta otra sintaxis resumida:

```
background-color:#faf;
```

Es decir, si cada valor hexadecimal está formado por el mismo carácter o número podemos utilizar esta sintaxis simplificada indicando un carácter o número solamente. Es decir, a este color no lo podemos representar con la sintaxis indicada:

```
background-color:#ffaafa
```

Ya que los últimos dos caracteres, fa, son distintos.



Confeccionemos una página y creemos distintas clases utilizando distintos formatos de asignación de color:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<p class="fondo1">Primer párrafo</p>
<p class="fondo2">Segundo párrafo</p>
<p class="fondo3">Tercer párrafo</p>
</body>
</html>
```

El archivo de estilos.css

```
.fondo1 {
  background-color:rgb(255,0,0);
}
.fondo2 {
  background-color:rgb(100%,50%,50%);
}
.fondo3 {
  background-color:#fab;
}
```

Primer párrafo

Segundo párrafo

Tercer párrafo

## 25. Definir un cursor para un elemento HTML

Disponemos de una propiedad llamada cursor que tiene por objetivo definir el cursor a mostrar cuando la flecha del mouse se encuentra sobre el elemento HTML.

Los valores que podemos asignarle a esta propiedad cursor son:

- **Crosshair**
- **Default**
- **Pointer**
- **Move**
- **e-resize**
- **ne-resize**
- **nw-resize**
- **n-resize**
- **se-resize**
- **sw-resize**
- **s-resize**
- **w-resize**
- **text**
- **wait**
- **help**
- **auto**

Veamos un ejemplo para configurar la propiedad cursor para el elemento de tipo ancla (a):

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<p>Este texto tiene por mostrar las anclas con un cursor distinto al que
está definido por defecto:</p>
<a href="http://www.google.com">google.com</a><br>
<a href="http://www.yahoo.com">yahoo.com</a><br>
<a href="http://www.bing.com">bing.com</a>
</body>
</html>
```

La hoja de estilo es:

```
a {
  cursor:help;
}
```

## 26. Aplicación de hojas de estilo a un formulario

Un formulario es el elemento esencial para el envío de datos al servidor por parte del visitante del sitio.

Veamos un ejemplo donde implementamos un formulario y le aplicamos una serie de reglas de estilo a los diferentes elementos HTML que intervienen:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<div id="contenedorform">
<form method="post" action="#">
  <label>Ingresa nombre:</label>
  <input type="text" name="nombre" size="30">
  <br>
  <label>Ingresa mail:</label>
  <input type="text" name="mail" size="45">
  <br>
  <label>Comentarios:</label>
  <textarea name="comentarios" cols="30" rows="5"></textarea>
  <br>
```

```

    <input class="botonsubmit" type="submit" value="confirmar">
  </form>
</div>
</body>
</html>

```

La hoja de estilo que se aplica es:

```

#contenedorform {
  width:500px;
  margin-left:20px;
  margin-top:10px;
  background-color:#ffe;
  border:1px solid #CCC;
  padding:10px 0 10px 0;
}

```

```

#contenedorform form label {
  width:120px;
  float:left;
  font-family:verdana;
  font-size:14px;
}
.botonsubmit {
  color:#f00;
  background-color:#bbb;
  border: 1px solid #fff;
}

```

Podemos observar que definimos un div contenedor y dentro de este el formulario. Para que los textos aparezcan a la izquierda, definimos una serie de label que las flotamos a izquierda, por lo que los controles del formulario aparecerán a derecha todos encolumnados.

## 27. Definiendo reglas de estilo a una tabla

Otro elemento muy común en un sitio web son las tablas. A este elemento HTML podemos aplicarle varias de las propiedades vistas tanto a la tabla en sí, como sus filas y celdas.

El elemento table podemos definir la propiedad border-collapse que puede tomar dos valores:

- **Collapse**
- **separate**

Si le asignamos el valor collapse una las líneas de los border de las celdas (con esto queda en muchas situaciones mejor estéticamente).

Veamos con un ejemplo como podemos afectar una tabla HTML con CSS.

```

<!DOCTYPE html>
<html>
<head>

```

```

<title>Problema</title>
<meta charset="UTF-8">
<link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<table>
  <caption>
cantidad de lluvia caida en mm.
  </caption>
  <thead>
    <tr>
      <th>Provincia</th><th>Enero</th><th>Febrero</th><th>Marzo</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>Córdoba</th>
      <td>210</td><td>170</td><td>120</td>
    </tr>
    <tr>
      <th>Málaga</th>
      <td>250</td><td>190</td><td>140</td>
    </tr>
    <tr>
      <th>Sevilla</th>
      <td>175</td><td>140</td><td>120</td>
    </tr>
  </tbody>
</table>
</body>
</html>

```

La hoja de estilo definida a esta tabla es:

```

caption
{
  font-family:arial;
  font-size:15px;
  text-align: center;
  margin: 0px;
  font-weight: bold;
  padding:10px;
}

table
{
  border-collapse: collapse;
}

th
{
  border-right: 1px solid #fff;
}

```

```
border-bottom: 1px solid #fff;
padding: 0.5em;
background-color:#6495ed;;
}
```

```
thead th
{
background-color:
#6495ed;
color: #fff;
}
```

```
tbody th
{
font-family:arial;
font-weight: normal;
background-color: #6495ed;
color:#ff0;
}
```

```
td {
border: 1px solid #000;
padding: .5em;
background-color:#ed8f63;
width:100px;
text-align:center;
```

cantidad de lluvia caída en mm.

Provincia	Enero	Febrero	Marzo
Córdoba	210	170	120
Málaga	250	190	140
Sevilla	175	140	120

El elemento caption dentro de una tabla es el título que debe aparecer arriba.

Como vimos la propiedad border-collapse puede tomar dos valores: collapse o separate. Separate deja las celdas con unos pixeles de separación, no así collapse.

El resto es la definición de una serie de reglas para los elementos th, th dentro del elemento tbody, th dentro del elemento thead y por último td.

## 28. Posicionamiento

### 28.1. Posicionamiento relativo, position: relative

La propiedad position determina el punto de referencia donde se debe localizar cada elemento HTML. Por defecto esta propiedad se inicializa con el valor **static**. Con el valor por defecto static, cada elemento HTML se localiza de izquierda a derecha y de arriba hacia abajo.

El segundo valor posible para esta propiedad es relative. En caso de fijar la propiedad position con el valor relative, podemos modificar la posición por defecto del elemento HTML modificando los valores left y top (con valores positivos o inclusive negativos)

El posicionamiento relativo mueve un elemento (div, h1, p etc.) en relación a donde se dispondría en el flujo normal. Es decir, deja espacio libre.

Veamos un ejemplo con tres div, de los cuales el segundo lo desplazamos 20 pixeles a nivel de columna y 5 pixeles a nivel de fila:

```

<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<div id="caja1">
<p>Esta es la primer caja.</p>
<p>No se desplaza.</p>
</div>
<div id="caja2">
<p>Esta es la segunda caja.</p>
<p>Se desplaza 15 píxeles a la derecha y 5 hacia abajo de su
posición
por defecto.</p>
</div>
<div id="caja3">
<p>Esta es la tercer caja.</p>
<p>No se desplaza.</p>
</div>
</body>
</html>

```

La hoja de estilo asociada es:

```

#caja1,#caja2,#caja3 {
  background-color:#f99;
  font-family:verdana;
  font-size:1.3rem;
}
#caja2 {
  position:relative;
  left:15px;
  top:5px;
}

```

Esta es la primer caja.

No se desplaza.

Esta es la segunda caja.

Se desplaza 15 píxeles a la derecha y 5 hacia abajo de su posición por defecto.

Esta es la tercer caja.

No se desplaza.

Repasemos un poquito, recordemos que cuando un conjunto de elementos tiene los mismos valores para una serie de propiedades los podemos agrupar separándolos por coma, esto sucede para los tres Id #caja1, #caja2 y #caja3 que tienen los mismos valores para las propiedades background-color, font-family y font-size:

```

#caja1,#caja2,#caja3 {
  background-color:#f99;
  font-family:verdana;
  font-size:1.3rem;
}

```

Luego como debemos inicializar la propiedad position sólo para el Id #caja2 lo hacemos en forma separada:

```
#caja2 {  
  position:relative;  
  left:15px;  
  top:5px;  
}
```

Aquí es donde inicializamos la propiedad position con el valor relative y desplazamos el elemento 15 píxeles a la derecha y 5 píxeles hacia abajo. Tengamos en cuenta que si asigno un valor muy grande a la propiedad top se superpone este elemento con el contenido del tercer div.

## 28.2. Posicionamiento absoluto, position: absolute

El posicionamiento absoluto dispone un elemento HTML completamente fuera del flujo de la página. El valor que debemos asignar a la propiedad position es absolute. Hay que tener en cuenta que no se reserva espacio en el flujo del documento como pasaba con el posicionamiento relativo (recordemos que con este posicionamiento podemos desplazar el elemento a cualquier parte de la página, pero el espacio por defecto para dicho elemento queda vacío).

El posicionamiento es siempre con respecto a la página.

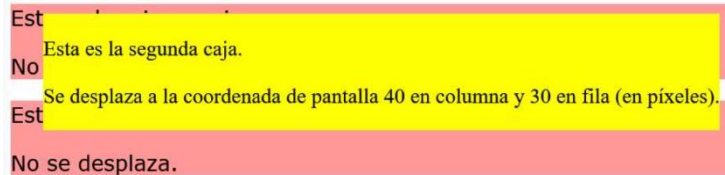
Normalmente cuando utilizamos el posicionamiento absoluto inicializamos las propiedades top y left.

Veamos un ejemplo para ver el funcionamiento del posicionamiento absoluto:

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Problema</title>  
  <meta charset="UTF-8">  
  <link rel="StyleSheet" href="estilos.css" type="text/css">  
</head>  
<body>  
<div id="caja1">  
  <p>Esta es la primera caja.</p>  
  <p>No se desplaza.</p>  
</div>  
<div id="caja2">  
  <p>Esta es la segunda caja.</p>  
  <p>Se desplaza a la coordenada de pantalla 40 en columna y 30 en  
fila (en  
  píxeles).</p>  
</div>  
<div id="caja3">  
  <p>Esta es la tercera caja.</p>  
  <p>No se desplaza.</p>  
</div>  
</body>  
</html>
```

La hoja de estilo definida:

```
#caja1,#caja3 {  
  background-color:#f99;  
  font-family:verdana;  
  font-size:1.3rem;  
}  
#caja2 {  
  background-color:#ff0;  
  font-size:1.3rem;  
  position:absolute;  
  left:40px;  
  top:30px;  
}
```



Est Esta es la primera caja.  
No Esta es la segunda caja.  
Est Se deslaza a la coordenada de pantalla 40 en columna y 30 en fila (en píxeles).  
No No se deslaza.

Como vemos inicializamos la propiedad position con el valor absolute y fijamos como coordenada para la caja la columna 40 y la fila 30 (en píxeles).

### 28.3. Posicionamiento absoluto y propiedad z-index

Hemos visto que cuando disponemos uno o más elementos con la propiedad position con valor absolute los mismos salen del flujo del resto de elementos de la página.

Ahora veamos que pasa si disponemos una serie de elementos con posición absoluta y se superponen. En esta situación debemos inicializar la propiedad **z-index** con un valor entero, el mismo indica cual tiene prioridad para visualizarse. El valor más grande indica cual tiene prioridad para visualizarse. Es decir que el navegador imprime los elementos con posición absoluta del valor más bajo al más alto.

Por ejemplo, si queremos mostrar tres div de color rojo, verde y azul con posiciones absolutas e indicando coordenadas que se superpongan. Inicializar la propiedad z-index de cada elemento de tal manera que quede más al frente el div de color verde, luego el azul y finalmente el rojo.

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Problema</title>  
  <meta charset="UTF-8">  
  <link rel="StyleSheet" href="estilos.css" type="text/css">  
</head>  
<body>  
  <div id="caja1">  
    <p>Esta es la primera caja.</p>  
  </div>  
  <div id="caja2">  
    <p>Esta es la segunda caja.</p>
```



```
</div>
<div id="caja3">
  <p>Esta es la tercera caja.</p>
</div>
</body>
</html>
```

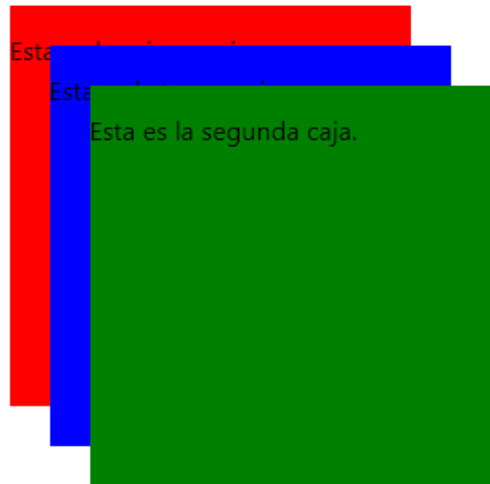
Definimos los tres div dentro de la página.

Luego la hoja de estilo que debemos definir es:

```
#caja1{
  position:absolute;
  background-color:red;
  left:10px;
  top:10px;
  width:200px;
  height:200px;
  z-index:1;
}

#caja2{
  position:absolute;
  background-color:green;
  left:50px;
  top:50px;
  width:200px;
  height:200px;
  z-index:3;
}

#caja3{
  position:absolute;
  background-color:blue;
  left:30px;
  top:30px;
  width:200px;
  height:200px;
  z-index:2;
}
```



Hemos definido los tres div con la propiedad position con el valor absolute. Luego vemos que aquel que tiene color verde (green) le asignamos a la propiedad z-index un valor 3 (el número puede ser cualquiera siempre y cuando sea el mayor de los z-index asignado a los otros dos div).

## 28.4. Posicionamiento fijo (position: fixed)

Esta es otra forma de disponer un elemento dentro de la página. La posición fija (fixed) ajusta el elemento en una coordenada fija con respecto a la ventana del navegador. Es una variante del posicionamiento absoluto. Recordemos que con el posicionamiento absoluto se dispone un elemento en una coordenada exacta con respecto a la página, pero con el posicionamiento fijo la posición es con respecto a

Por ejemplo, si se pide disponer dos div con posicionamiento fijo, uno en la parte superior y otro a la izquierda de la ventana.

[illegible]

La hoja de estilo a aplicar a esta página es:

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

#barrasuperior{
  position:fixed;
  left:0px;
  width:100%;
  height:50px;
  background-color:black;
  color:white;
}

#barralateral{
  position:fixed;
  left:0px;
  top:50px;
  width:200px;
  height:100%;
  background-color:#eee;
}

#contenido{
  padding-top:70px;
  padding-left:220px;
}
```



Hemos definido tanto para la barra superior como para la barra lateral el valor fixed para la propiedad position: position:fixed;

La barra superior comienza en la columna cero a nivel de píxeles y tiene un ancho de 100 por ciento (es decir ocupa siempre todo el ancho del navegador y una altura de 50px):

```
left:0px;
width:100%;
height:50px;
```

La barra lateral definimos los siguientes valores:

```
left:0px;
top:50px;
width:200px;
height:100%;
```

Hay que tener en cuenta que el elemento HTML con la propiedad fixed sale del flujo normal de los elementos de la página (igual como ocurre con el valor absolute), luego por eso definimos para el Div de contenido un padding para la parte superior e izquierdo donde se encuentran los div fixed:

```
padding-top:70px;
padding-left:220px;
```

## 28.5. Disposición de 2 columnas (position: absolute)

Empezaremos a ver como componer una página sin utilizar las tablas HTML (un medio muy utilizado hasta hace muy poco, ya que es ampliamente soportado por navegadores antiguos).

Una solución para crear una página con dos columnas es utilizar el posicionamiento absoluto:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<div id="columna1">
Aquí va el contenido de la columna 1. Aquí va el contenido
de la columna 1. Aquí va el contenido de la columna 1.
Aquí va el contenido de la columna 1. Aquí va el contenido de
  la columna 1.
...
Aquí va el contenido de la columna 1. Aquí va el contenido de
  la columna 1.
</div>
<div id="columna2">
Aquí va el contenido de la columna 2. Aquí va el contenido de la
columna 2.
Aquí va el contenido de la columna 2. Aquí va el contenido de la
columna 2.
...
Aquí va el contenido de la columna 2. Aquí va el contenido de la
columna 2.
</div>
</body>
</html>
```

La hoja de estilo para esta página es:

```
* {
  margin:0;
  padding:0;
}
#columna1 {
  position:absolute;
  top:0px;
  left:0px;
  width:200px;
  margin-top:10px;
  background-
color:#ffff55;
}
```

[illegible][illegible]

```
#columna2 {
  margin-left:220px;
  margin-right:20px;
  margin-top:10px;
  background-color:#ffffbb;
}
```

La primera regla de disponer el selector universal, es decir afecta a todos los elementos HTML, es sacar el margen y padding (generalmente dispondremos esta regla):

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

Cuando es cero no hace falta indicar la unidad (píxeles, em etc.)

Ahora la regla definida para la primera columna es:

```
#columna1 {
  position:absolute;
  top:0px;
  left:0px;
  width:200px;
  margin-top:10px;
  background-color:#ffff55;
}
```

Es decir, inicializamos la propiedad position con el valor absolute, con lo que debemos indicar la posición del div en la página por medio de las propiedades top y left, en este caso lo posicionamos en la columna 0 y fila 0 y además inicializamos la propiedad width, con lo cual le estamos indicando que esta columna tendrá siempre 200 píxeles de ancho.

Además inicializamos la propiedad margin-top con 10 píxeles, recordemos que todos los elementos tienen margin y padding cero.

Ahora veamos cómo inicializamos la segunda columna:

```
#columna2 {
  margin-left:220px;
  margin-right:20px;
  margin-top:10px;
  background-color:#ffffbb;
}
```

Esta regla no inicializa la propiedad position, por lo que el div ocupa la posición que le corresponde por defecto, es decir, empieza en la coordenada 0,0 de la página. El truco está en inicializar la propiedad margin-left con un valor mayor a 200, que es el ancho de la columna1.

El resto de propiedades que inicializamos son el margin-top, para que sea igual que la primera columna y el margin-right, para que no quede el texto pegado a la derecha.

Hay que tener cuidado con este formato de página ya que no es "responsive" o adaptativo. No importa el ancho en píxeles del dispositivo siempre estarán visibles las dos columnas.

## 29. Propiedad float

## 29.1. Propiedad float aplicada a una imagen

La propiedad **float** saca del flujo un elemento HTML. Esta propiedad admite tres valores:

- Left
- Right
- none

Cuando aplicamos esta propiedad al elemento HTML `img`, podemos hacer que el texto envuelva a la imagen.


Veamos un ejemplo:

[illegible]

```
</body>
</html>
```

La hoja de estilo:

```
img {
  float:right;
}
```



Es importante hacer notar que si no la flotamos a la imagen solo habrá una línea de texto a la derecha de la imagen.

## 29.2. Propiedad float aplicada a otros elementos HTML

La propiedad float permite como hemos visto con las imágenes tomar un elemento HTML y colocarlo lo más a la izquierda o derecha. Luego los otros elementos siguen el flujo alrededor del elemento que está flotante. Podemos flotar cualquier elemento HTML.

Normalmente cuando se utiliza la propiedad float, también debe inicializar la propiedad width para fijar el ancho del elemento.

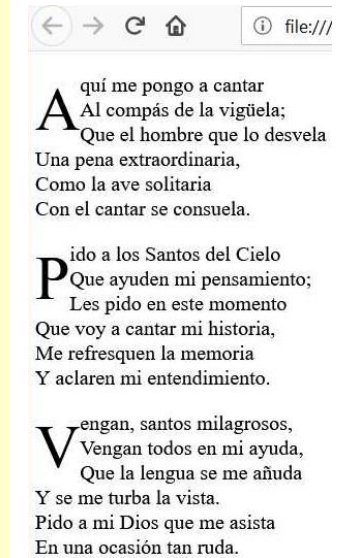
Cuando se define un elemento que flote el alto del mismo afecta como se disponen los demás elementos de la página, será distinto si es un texto que envolverá al elemento flotante o si se trata de una imagen o div.

Por ejemplo, si necesitamos mostrar una poesía y disponer la primera letra de cada párrafo que flote a izquierda, podríamos hacerlo del modo siguiente:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css"
type="text/css">
</head>
<body>

<p><span class="letra">A</span>quí me pongo a
cantar <br>
Al compás de la vigüela; <br>
Que el hombre que lo desvela <br>
Una pena extraordinaria, <br>
Como la ave solitaria <br>
Con el cantar se consuela.</p>

<p><span class="letra">P</span>ido a los Sant
Que ayuden mi pensamiento; <br>
Les pido en este momento <br>
Que voy a cantar mi historia,<br>
Me refresquen la memoria <br>
Y aclaren mi entendimiento. </p>
```



```
<p><span class="letra">V</span>engan, santos milagrosos,<br>
Vengan todos en mi ayuda,<br>
Que la lengua se me añuda <br>
Y se me turba la vista. <br>
Pido a mi Dios que me asista<br>
En una ocasión tan ruda. </p>

</body>
</html>
```

La hoja de estilo queda definida:

```
.letra {
  float:left;
  font-size:3rem;
}
```

Definimos la clase letra e inicializamos la propiedad float con el valor left y definimos el tamaño de fuente con 3 rem (es decir tres veces más grande que el valor de la fuente definida en el elemento raíz "html").

## 30. Propiedad clear

---

La propiedad clear se emplea combinándola con otros elementos que se hayan definido la propiedad float. La propiedad clear define que un elemento HTML no permita que flote en el caso que el elemento anterior se haya definido con dicha propiedad.

También se puede definir que no flote a izquierda o derecha únicamente.

Los valores posibles que podemos asignarle a la propiedad clear son:

- **Both**
- **Left**
- **Right**
- **none**

Implementaremos un problema para mostrar los resultados empleando la propiedad float únicamente y el resultado que se logra empleando la propiedad clear luego.

El ejemplo siguiente ilustraría como mostrar cuatro div flotando a izquierda de 250 píxeles; y hacer que los div de la izquierda tengan más texto que los de la derecha.

El archivo HTML queda:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<div id="caja1">
```



```
<p>caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1
caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1
caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1
...
caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1 caja1
</p>
</div>
<div id="caja2">

<p>
caja2 caja2 caja2 caja2 caja2 caja2 caja2 caja2 caja2 caja2
caja2 caja2 caja2 caja2 caja2 caja2 caja2 caja2 caja2 caja2
...
caja2 caja2 caja2 caja2 caja2 caja2 caja2 caja2 caja2 caja2
</p>
</div>
<div id="caja3">
<p>
caja3 caja3 caja3 caja3 caja3 caja3 caja3 caja3 caja3 caja3
...
caja3 caja3 caja3 caja3 caja3 caja3 caja3 caja3 caja3 caja3
</p>
</div>
<div id="caja4">
<p>
caja4 caja4 caja4 caja4 caja4 caja4 caja4 caja4 caja4 caja4
...
caja4 caja4 caja4 caja4 caja4 caja4 caja4 caja4 caja4 caja4
</p>
</div>
</body>
</html>
```

La hoja de estilo definida para esta página es:

```
#caja1 {
    float:left;
    width:250px;
    background-color:#ffc;
    margin:10px;
}
#caja2 {
    float:left;
    width:250px;
    background-color:#ffc;
    margin:10px;
}
#caja3 {
    float:left;
    width:250px;
    background-color:#ffc;
```

[illegible][illegible][illegible][illegible]

```
margin:10px;
}
#caja4 {
float:left;
width:250px;
background-color:#ffc;
margin:10px;
}
```

Como vemos cuando el div debe disponerse más abajo busca el primer espacio libre de derecha a izquierda.

Supongamos que queremos mostrarlo lo más a la izquierda posible, es decir que comience una nueva fila en pantalla. Para ello cuando definimos el estilo para la caja4 aparece la propiedad clear:

```
#caja4 {
    float:left;
    clear:left;
    width:250px;
    background-color:#ffc;
    margin:10px;
}
```

Con este cambio ahora el resultado en pantalla es el siguiente:

[illegible][illegible][illegible][illegible]

## 31. Disposición

### 31.1. Disposición de 2 columnas (propiedad float)

Una segunda forma de implementar una página con dos columnas es utilizar la propiedad float. Disponemos dos div. Al primero lo flotamos hacia la izquierda con un width fijo y el segundo se acomoda inmediatamente a la derecha.

```
#column2 {
  margin-left:210px;
}
```

Veamos un ejemplo:

La hoja de estilo:

[illegible]

Hay que tener cuidado con este formato de página ya que no es "responsive" o adaptativo. No importa el ancho en píxeles del dispositivo siempre estarán visibles las dos columnas.

## 31.2. Disposición de 2 columnas, cabecera y pie

Una estructura muy común en la web es la disposición de una cabecera de página, seguida de dos columnas y un pie de página.

Implementaremos un ancho fluido, es decir que el tamaño del contenido se adapta al tamaño de la ventana del navegador. Esto lo logramos iniciando la propiedad:

```
width:100%;
```

La implementación de esta estructura de página es la siguiente:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<div id="contenedor">
<div id="cabecera">
<h1>Aquí el título de la página</h1>
</div>
<div id="columna1">
<p>columna1. columna1. columna1.</p>
</div>
<div id="columna2">
<h2>Título de la columna</h2>
<p>Contenido de la columna2. </p>
</div>
<div id="pie">
Pie de página.
</div>
</div>
</body>
</html>
```

La hoja de estilo definida para esta página es la siguiente:

```
* {
  margin:0;
  padding:0;
}
#contenedor
{
  width:100%;
  border:1px solid #000;
  line-height:130%;
```

Aquí el título de la página	
columna1. columna1. columna1.	<b>Título de la columna</b> Contenido de la columna2.
Pié de página.	

```

    background-color:#f2f2f2;
}
#cabecera
{
    padding:10px;
    color:#fff;
    background-color:#becdfe;
    clear:left;
}
#columna1
{
    float:left;
    width:200px;
    margin:0;
    padding:1rem;
}
#columna2
{
    margin-left:210px;
    border-left:1px solid #aaa;
    padding:1rem;
}
#pie {
    padding:10px;
    color:#fff;
    background-color:#becdfe;
    clear:left;
}

```

Hay algunas propiedades claves que debemos analizar en la regla #contenedor:

```
width:100%;
```

Con esto estamos indicando que siempre ocupe todo el espacio en ancho del navegador, indistintamente de la resolución de pantalla o el tamaño de ventana del navegador.

Luego, tanto para la cabecera como para el pie, tenemos:

```
clear:left;
```

La propiedad clear hace que un elemento no tenga elementos flotantes a su lado. Eso es lo que queremos para la cabecera y el pie.

Hay que tener en cuenta que este formato de página no es "responsive".

### 31.3. Disposición de 3 columnas, cabecera y pie

Una modificación al concepto anterior nos permite agregar una tercera columna flotando a derecha, lo único que hay que tener cuidado es que dentro del HTML debemos disponer los div de la columna 1 y 3 en primer lugar, ya que son los que se flotan, y por último, la columna 2, que es la central:

```

<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<div id="contenedor">
<div id="cabecera">
<h1>Aquí el título de la página</h1>
</div>
<div id="columna1">
<p>columna1. </p>
</div>
<div id="columna3">
<p>columna3.</p>
</div>
<div id="columna2">
<h2>Título de la columna</h2>
<p>Contenido de la columna2.</p>
</div>
<div id="pie">
Pie de página.
</div>
</div>
</body>
</html>

```

La hoja de estilo es:

```

* {
  margin:0;
  padding:0;
}
#contenedor
{
  width:100%;
  border:1px solid #000;
  line-height:130%;
  background-color:#f2f2f2;
}
#cabecera
{
  padding:10px;
  color:#fff;
  background-color:#becdfe;
  clear:left;
}
#columna1
{
  float:left;

```

Aquí el título de la página		
columna1.	<b>Título de la columna</b> Contenido de la columna2.	columna3.
Pie de página.		

```

width:200px;
margin:0;
padding:1rem;
}
#columna2
{
margin-left:210px;
margin-right:230px;
border-left:1px solid #aaa;
border-right:1px solid #aaa;
padding:1rem;
}
#columna3
{
float:right;
width:200px;
margin:0;
padding:1rem;
}
#pie {
padding:10px;
color:#fff;
background-color:#becdfe;
clear:left;
}

```

## 32. Diseño de ancho fijo

---

El diseño de una página de ancho fijo no cambia la disposición de los elementos HTML a medida que el visitante modifica la ventana del navegador (agrandando o achicando). Cuando el espacio asignado no entra en la ventana aparece la barra de scroll.

Las medidas normalmente se indican en píxeles.

Para implementar una página con un diseño de ancho fijo de 960 píxeles que se muestre centrado en la página que disponga de una cabecera con un menú, el cuerpo principal con tres columnas y finalmente un pie de página.

El archivo html es el siguiente:

```

<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>

<div id="cabecera">
  <ul>

```

```

        <li><a href="">Opcion 1</a></li>
        <li><a href="">Opcion 2</a></li>
        <li><a href="">Opcion 3</a></li>
    </ul>
</div>

<div class="columna1">
    <p>
        columna1 columna1 columna1 columna1 columna1 columna1
        ...
        columna1 columna1 columna1 columna1 columna1 columna1
    </p>
</div>
<div class="columna2">
    <p>
        columna2 columna2 columna2 columna2 columna2 columna2
        ...
        columna2 columna2 columna2 columna2 columna2 columna2
    </p>
</div>
<div class="columna3">
    <p>
        columna3 columna3 columna3 columna3 columna3 columna3
        ...
        columna3 columna3 columna3 columna3 columna3 columna3
    </p>
</div>

<div id="pie">
    <p>Copyright</p>
</div>
</body>

```

Luego la hoja de estilo que se aplica a la página es:

```

* {
    margin:0;
    padding:0;
}

body {
    width: 960px;
    margin: 0 auto;
}

#cabecera{
    background-color: #eee;
    padding: 10px;
    margin: 10px;
}

#pie {

```



```

background-color: #eee;
padding: 10px;
margin: 10px;
clear: both;
}

.columna1, .columna2, .columna3 {
background-color: #eee;
width: 280px;
float: left;
margin: 10px;
padding: 10px;
}

li {
display: inline;
padding: 5px;
}

```

En el elemento body indicamos en píxeles el ancho de la página y que aparezca centrado:

```

body {
width: 960px;
margin: 0 auto;
}

```

Las columnas son de 280 píxeles de ancho más el margin y padding y las flotamos a izquierda:

```

.columna1, .columna2, .columna3 {
background-color: #eee;
width: 280px;
float: left;
margin: 10px;
padding: 10px;
}

```

Para el pie de página no nos olvidamos de inicializar la propiedad clear:

```

#pie {
background-color: #eee;
padding: 10px;
margin: 10px;
clear: both;
}

```

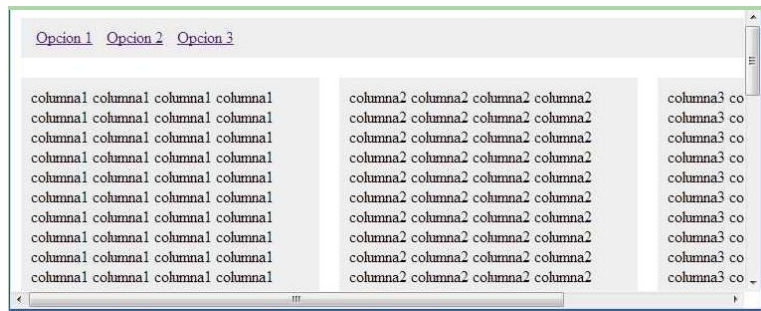
Luego si accedemos a esta página desde un navegador y el ancho de la ventana del navegador supera los 960 píxeles veremos un espacio en blanco a ambos lados.



Si por el contrario abrimos la página y la ventana del navegador en ancho es más pequeña a 960 píxeles aparece la barra de scroll en la parte inferior:

Como ventajas del diseño de ancho fijo podemos nombrar: tenemos mayor control sobre el aspecto y la posición de los elementos HTML en la página, podemos controlar los largos de las líneas de texto sin tener en cuenta el tamaño de la ventana del usuario, el tamaño de las imágenes es la misma en relación al resto de la página.

Como desventajas del empleo de este tipo de diseño es que podemos tener grandes espacios vacíos a izquierda y derecha en monitores grandes y en dispositivos con anchos pequeño nos obligará hacer constantemente scroll a izquierda y derecha.



### 31.4. Diseño de ancho líquido

Cuando empleamos diseño de ancho líquido los elementos se contraen o expanden según el ancho de la ventana del navegador.

El diseño líquido requiere emplear porcentajes para especificar el ancho de cada div de forma que el diseño se estira o contrae para adaptarse al tamaño de la ventana del navegador.

Como ejemplo podemos implementar una página con un diseño de ancho líquido. Disponer en ella una cabecera con un menú, el cuerpo principal con tres columnas y finalmente un pie de página.

El archivo html es el siguiente:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>

<div id="cabecera">
  <ul>
    <li><a href="">Opcion 1</a></li>
    <li><a href="">Opcion 2</a></li>
    <li><a href="">Opcion 3</a></li>
  </ul>
</div>

<div class="columna1">
  <p>
    columna1 columna1 columna1 columna1 columna1 columna1
    ...
  </p>
</div>
```

```

        columna1 columna1 columna1 columna1 columna1 columna1
    </p>
</div>
<div class="columna2">
    <p>
        columna2 columna2 columna2 columna2 columna2 columna2
        ...
        columna2 columna2 columna2 columna2 columna2 columna2
    </p>
</div>
<div class="columna3">
    <p>
        columna3 columna3 columna3 columna3 columna3 columna3
        ...
        columna3 columna3 columna3 columna3 columna3 columna3
    </p>
</div>

<div id="pie">
    <p>Copyright</p>
</div>
</body>

```

Como podemos observar la estructura de la página HTML no sufre cambios con respecto al diseño con ancho fijo. Donde si podemos identificar cambios es en la hoja de estilo:

```

* {
    margin:0;
    padding:0;
}

body {
    width: 90%;
    margin: 0 auto;
}

```

```

#cabecera{
    background-color: #eee;
    padding: 10px;
    margin: 10px;
}

```

```

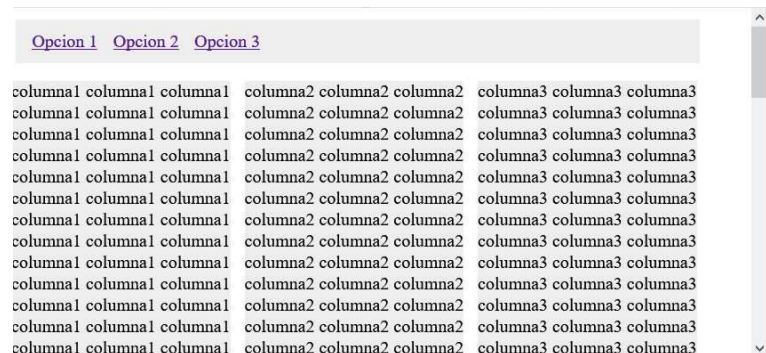
#pie {
    background-color: #eee;
    padding: 10px;
    margin: 10px;
    clear:both;
}

```

```

.columna1, .columna2, .columna3 {
    background-color: #eee;
}

```



```
width: 31%;  
float: left;  
margin: 1%;  
}  
  
li {  
display: inline;  
padding: 5px;  
}
```

En el elemento body normalmente definimos un ancho de alrededor de 90% para dejar un pequeño espacio a izquierda y derecha de la ventana del navegador. Con (margin: 0 auto;) estamos centrando el contenido:

```
body {  
width: 90%;  
margin: 0 auto;  
}
```

En las columnas ahora tenemos que indicar un ancho en porcentajes para cada una de ellas:

```
.columna1, .columna2, .columna3 {  
background-color: #eee;  
width: 31%;  
float: left;  
margin: 1%;  
}
```

Como ventajas este diseño podemos nombrar: La página se amplía para ocupar toda ventana del navegador por lo que hay más espacio en una pantalla grande, si el visitante del sitio tiene una pantalla pequeña la página puede contrato para ajustarlo sin el usuario tenga que hacer scroll izquierda y derecha.

Como desventajas de este diseño podemos nombrar: La página puede verse muy distinta en pantalla muy grandes o muy pequeñas, en pantalla muy grandes las líneas de texto pueden ser muy largas por lo que se hace muy dificultoso la lectura, de forma similar si el ancho es muy estrecho también su lectura, algunos elementos pueden salir de su espacio asignado.

Hay que tener cuidado con este formato de página ya que no es "responsive" o adaptativo. No importa el ancho en píxeles del dispositivo siempre estarán visibles las tres columnas.

## 33. Selectores

---

### 33.1. Selector de hijos

Los selectores nos permiten aplicar estilos a un elemento o conjunto de elementos de una página web. Repasemos un poco los selectores vistos hasta este momento:

Selector de un tipo de elemento:

```
h1 {  
font-family:Arial;  
}
```

Se aplica el tipo de fuente Arial a todos los elementos h1 de la página sin excepción.

```
h1.principal {
  font-family:Arial;
}
```

Se aplica el tipo de fuente Arial a todos los elementos h1 que tienen definida la clase "principal"

```
h1#titulo1 {
  font-family:Arial;
}
```

Se aplica el tipo de fuente Arial al elemento h1 que tiene definida la propiedad id con el valor "titulo1".

**Selectores de descendientes:** Cuando se definen dos elementos HTML seguidos por uno o más espacios en blanco, las propiedades afectan a los elementos con el segundo elemento contenidos dentro del primer elemento, aunque haya etiquetas intermedias:

```
ul em{
  color:#f00;
}
```

Todos los elementos "em" se mostrarán de color rojo siempre y cuando estén contenidos por un elemento "ul".

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<h1>Esto es una <em>prueba</em>.</h1>
<ul>
  <li>Opción 1 (es la más <em>fácil</em>)</li>
  <li>Opción 2 (es de dificultad <em>media</em>)</li>
  <li>Opción 2 (muy <em>compleja</em>)</li>
</ul>
</body>
</html>
```

El archivo .css:

```
li em{
  color:#f00;
}
```

**Esto es una *prueba*.**

- Opción 1 (es la más *fácil*)
- Opción 2 (es de dificultad *media*)
- Opción 2 (muy *compleja*)

En este ejemplo las palabras fácil, media y compleja aparecen de color rojo. Esto debido que están encerradas entre las marcas "em" y están contenidas en un "ul".

Por otro lado la palabra "prueba" no aparece de color rojo ya aunque está encerrada con el elemento "em" dicho elemento no está contenido en una lista no ordenada "ul".

Ahora sí, veamos para aplicar "Selector de hijos". La sintaxis es similar a los selectores descendientes con la diferencia de que reemplazamos el espacio en blanco por el carácter >

```
h1>em {  
  color:#0f0;  
}
```

Aquí solo se aplica el color #0f0 si em es un elemento hijo directo de h1.

Para ver esto más detenidamente, se propone como ejemplo plantear una página que contenga en un div dos párrafos y una lista desordenada. Dentro de la lista definir tres ítems con un párrafo cada uno. Luego definir un selector para que pinte de color azul solo los párrafos que dependen directamente del div.

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Problema</title>  
  <meta charset="UTF-8">  
  <link rel="StyleSheet" href="estilos.css" type="text/css">  
</head>  
<body>  
<div>  
  <p>Primer párrafo</p>  
  <p>Segundo párrafo</p>  
  <ul>  
    <li><p>opcion 1</p></li>  
    <li><p>opcion 2</p></li>  
    <li><p>opcion 3</p></li>  
  </ul>  
</div>  
</body>  
</html>
```

Primer párrafo

Segundo párrafo

- opcion 1
- opcion 2
- opcion 3

```
div>p{  
  color:#00f;  
}
```

Los dos párrafos: "Primer párrafo" y "Segundo párrafo" aparecen de color azul ya que son hijos directos del elemento HTML div, en cambio los tres párrafos contenidos en la lista no se les aplica el estilo definido ya que no son hijos directos del div.

## 33.2. Selector de hermano adyacente y hermano general

El "**selector de hermano adyacente**" permite seleccionar un elemento HTML que viene inmediatamente después de otro elemento HTML. Los dos elementos deben tener el mismo elemento padre. Por ejemplo:

```
<body>
<h1>Titulo 1</h1>
<p>Esto es el primer párrafo.</p>
<p>Esto es el segundo párrafo.</p>
<p>Esto es el tercer párrafo.</p>
</body>
```

Mediante la regla de estilo:

```
h1+p{
  color:#f00;
}
```

Seleccionamos el primer párrafo que le sigue al elemento h1. Luego el primer párrafo se pinta de rojo.

Si aplicamos este otro estilo al mismo bloque HTML:

```
p+p{
  color:#f00;
}
```

Se pintan de rojo el segundo y tercer párrafo ya que a cada uno de estos párrafos le antecede otro párrafo (no se pinta el primer párrafo porque no le antecede un párrafo sino un "h1")

Ahora el "**selector de hermano general**" permite seleccionar todos los hermanos a diferencia del selector de hermano adyacente que selecciona el primero.

Si tenemos el siguiente fragmento:

```
<body>
<h1>Titulo 1</h1>
<p>Esto es el primer párrafo.</p>
<p>Esto es el segundo párrafo.</p>
<p>Esto es el tercer párrafo.</p>
</body>
```

Y le aplicamos la regla:

```
h1~p{
  color:#f00;
}
```

Como resultado se aplica el color rojo a todos los párrafos adyacentes que tienen como hermano el elemento "h1".

Una página completa que muestra el empleo de estos dos selectores:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
```

```

</head>
<body>
<h1>Titulo 1</h1>
<p>Esto es el primer párrafo.</p>
<p>Esto es el segundo párrafo.</p>
<p>Esto es el tercer párrafo.</p>

<h2>Titulo 2</h2>
<p>Esto es el primer párrafo.</p>
<p>Esto es el segundo párrafo.</p>
<p>Esto es el tercer párrafo.</p>
</body>
</body>
</html>

```

## Titulo 1

Esto es el primer párrafo.

Esto es el segundo párrafo.

Esto es el tercer párrafo.

## Titulo 2

Esto es el primer párrafo.

Esto es el segundo párrafo.

Esto es el tercer párrafo.

```

h1+p{
    color:#f00;
}

h2~p{
    color:#f00;
}

```

### 33.3. Selector de atributo

Ahora veremos otra forma de seleccionar elementos HTML mediante el acceso de sus atributos.

Recordemos que un elemento HTML está normalmente constituido por una marca de comienzo, un valor, la marca de cierre y sus atributos:

```
<a href="http://www.google.com.ar" target="_blank">Titulo Principal</a>
```

En este ejemplo el elemento "a" tiene definido dos atributos: href y target.

Hay una serie de posibilidades de utilizar el selector de atributo que pasaremos a ver:

1. Con el selector de atributo podemos verificar si un elemento HTML tiene definida un atributo determinado para así poder aplicar un estilo. Por ejemplo apliquemos estilos distintos a los elementos "a" que tienen definido el atributo target:

```

<!DOCTYPE html>
<html>
<head>
    <title>Problema</title>
    <meta charset="UTF-8">
    <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>

```



```

<body>
<h1>Enlaces a buscadores.</h1>
<ol>
  <li>
    <a href="http://www.google.com">Ir a google</a>
  </li>
  <li>
    <a href="http://www.bing.com">Ir a bing</a>
  </li>
  <li>
    <a href="http://www.yahoo.com" target="_blank">Ir a yahoo</a>
  </li>
</ol>
</body>
</html>

```

## Enlaces a buscadores.

```

a[target]{
  color:#f00;
}

```

1. [Ir a google](#)
2. [Ir a bing](#)
3. [Ir a yahoo](#)

Esta regla se aplica a todos los elementos de tipo "a" que tienen definida la propiedad llamada "target". Es decir, el tercer enlace de la página aparece de color rojo ya que tiene definida dicha propiedad. Los dos primeros enlaces no se le aplica esta regla de CSS porque no tienen definida la propiedad target.

2. Una segunda posibilidad es no solo verificar si un elemento HTML define una determinada propiedad sino controlar si su contenido almacena un determinado valor.

Confeccionemos una página que muestre todos los enlaces "a" de la página que tengan inicializado el atributo href con el valor # con una línea tachada (que represente que no tenemos acceso a dicho hipervínculo):

```

<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<h1>Enlaces a buscadores.</h1>
<ol>
  <li>
    <a href="http://www.google.com">Ir a google</a>
  </li>
  <li>
    <a href="#">Ir a bing</a>
  </li>
  <li>
    <a href="#" target="_blank">Ir a yahoo</a>
  </li>
</ol>

```

```
</li>
</ol>
</body>
</html>
```

## Enlaces a buscadores.

```
a[href="#"]{
  text-decoration:line-through;
}
```

1. [Ir a google](#)
2. [Ir a bing](#)
3. [Ir a yahoo](#)

Como podemos ver entre los corchetes no solo hacemos referencia al nombre de la propiedad, sino que disponemos el carácter = y la cadena con la que estamos comparando el contenido del atributo.

En nuestro ejemplo tenemos dos enlaces que tienen almacenados en la propiedad href la cadena "#" por lo que dichos enlaces, al cumplir la regla del selector se aplica a text-decoration el valor line-through.

3. Podemos también controlar si el valor de la propiedad comienza, finaliza o tiene en su interior una determinada cadena de caracteres. Veamos la sintaxis con un ejemplo:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<h1>Enlaces a buscadores.</h1>
<ol>
  <li>
    <a href="http://www.google.com">Ir a google</a>
  </li>
  <li>
    <a href="www.bing.com.ar">Ir a bing</a>
  </li>
  <li>
    <a href="www.yahoo.com">Ir a yahoo</a>
  </li>
</ol>
</body>
</html>
```

## Enlaces a buscadores.

```
a[href^="http"] {  
  color:#f00;  
}
```

```
a[href$=".ar"] {  
  color:#0f0;  
}
```

```
a[href*="yahoo"] {  
  color:#00f;  
}
```

1. [Ir a google](#)
2. [Ir a bing](#)
3. [Ir a yahoo](#)

Hay que tener en cuenta que utilizamos la sintaxis ^= para controlar si el atributo comienza con la cadena que le indicamos entre comillas (es decir el primer enlace cumple la regla ya que la propiedad href comienza con esos cuatro caracteres):

```
a[href^="http"] {  
  color:#f00;  
}
```

La sintaxis \$= sirve para controlar si el valor del atributo finaliza con la cadena de caracteres indicada (el segundo enlace cumple esta regla de estilo):

```
a[href$=".ar"] {  
  color:#0f0;  
}
```

Por último, con la sintaxis \*= controlamos si el atributo contiene en alguna parte la cadena indicada (en nuestro ejemplo el tercer enlace contiene en su interior la cadena "yahoo"):

```
a[href*="yahoo"] {  
  color:#00f;  
}
```

## 34. Pseudo-clases

### 34.1. Pseudo-clases: first-child y last-child

En los próximos conceptos veremos otras formas de seleccionar elementos HTML de una página para aplicarle estilos.

La pseudoclase first-child nos permite seleccionar el primer elemento hijo de otro elemento padre y de manera similar la pseudoclase last-child selecciona el último elemento hijo.

Para confeccionar una página web en la que se pide disponer una lista ordenada de 4 elementos, fijar con color verde el primer elemento y color rojo el último elemento de la lista.

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Problema</title>  
  <meta charset="UTF-8">
```

```
<link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
  <ol>
    <li>Primer opción.</li>
    <li>Segunda opción.</li>
    <li>Tercer opción.</li>
    <li>Cuarta opción.</li>
  </ol>
</body>
</html>
```

```
ol li:first-child {
  color:green;
}

ol li:last-child {
  color:red;
}
```

1. Primer opción.  
2. Segunda opción.  
3. Tercer opción.  
4. Cuarta opción.

Recordemos que a las pseudoclases se les antecede el carácter : y en forma seguida indicamos el nombre de la pseudoclase previo a los dos puntos indicamos el nombre del elemento a seleccionar:

```
ol li:first-child {
  color:green;
}
```

Podemos también expresarlo sin anteceder el elemento ol, es decir sin indicar "selector descendiente":

```
li:first-child {
  color:green;
}
```

De forma similar para aplicar un estilo al último ítem de la lista ordenada utilizamos la

```
pseudoclase last-child:
ol li:last-child {
  color:red;
}
```

## 34.2. Pseudo-clases: nth-child y nth-last-child

Selecciona uno o más elementos según los valores que le pasemos como parámetro a esta pseudo-clase.

### - Pasando los valores odd (impar) y even (par):

Si a la pseudoclase nth-child le pasamos el valor odd nos seleccionará todos los elementos de las posiciones impares:

```
tr:nth-child(odd) {  
    background-color:red;  
}
```

Con este selector fijamos de color rojo de fondo de todas las filas de la tabla que ocupan posiciones impares.

Veamos un ejemplo de fijar colores rojo para las filas pares de una tabla y color verde para las impares:

```
<!DOCTYPE html>  
<html>  
<head>  
    <title>Problema</title>  
    <meta charset="UTF-8">  
    <link rel="StyleSheet" href="estilos.css" type="text/css">  
</head>  
<body>  
<table>  
    <tr>  
        <td>fila 1</td><td>fila 1</td>  
    </tr>  
    <tr>  
        <td>fila 2</td><td>fila 2</td>  
    </tr>  
    <tr>  
        <td>fila 3</td><td>fila 3</td>  
    </tr>  
    <tr>  
        <td>fila 4</td><td>fila 4</td>  
    </tr>  
</table>  
</body>  
</html>
```

```
tr:nth-child(odd) {  
    background-color:red;  
}
```

```
tr:nth-child(even) {  
    background-color:green;  
}
```

```
table {  
    border-collapse: collapse;  
    width:80%;  
    margin:0 auto;  
}
```

```
td {  
    border: 1px solid #000;  
    text-align:center;
```

fila 1	fila 1
fila 2	fila 2
fila 3	fila 3
fila 4	fila 4

```
padding:0.5rem;
}
```

Con estas dos reglas tenemos seleccionadas las filas impares:

```
tr:nth-child(odd) {
  background-color:red;
}
```

Y las filas pares:

```
tr:nth-child(even) {
  background-color:green;
}
```

### - Pasando un valor entero:

Si pasamos un valor entero estamos indicando la posición del elemento que queremos acceder en forma específica.

Por ejemplo podemos seleccionar la tercer fila de la tabla con la siguiente sintaxis:

```
tr:nth-child(3) {
  background-color:red;
}
```

Como resultado podemos ver que solo se aplica este estilo a la fila indicada:

fila 1	fila 1
fila 2	fila 2
fila 3	fila 3
fila 4	fila 4

### - Usando una expresión:

Es la forma más rica y compleja que tenemos para indicar una secuencia de selección. Debemos plantear una expresión con la siguiente estructura:

$an+b$

Donde en a y b debemos indicar valores enteros (podemos omitir b) y n es un carácter (siempre lo disponemos)

Es más fácil con ejemplos poder entender como seleccionamos elementos:

```
tr:nth-child(3n) {
  background-color:red;
}
```

Con esta expresión se seleccionan las filas 3, 6, 9 etc. Como resultado tenemos:

fila 1	fila 1
fila 2	fila 2
fila 3	fila 3
fila 4	fila 4
fila 5	fila 5
fila 6	fila 6
fila 7	fila 7

Hay que entender esa expresión como una actividad repetitiva donde n va tomando los valores: 0, 1, 2, 3, etc.

Si queremos partir del primero e ir avanzando de a dos planteamos la expresión:

```
tr:nth-child(3n+1) {  
    background-color:red;  
}
```

Como resultado tenemos:

fila 1	fila 1
fila 2	fila 2
fila 3	fila 3
fila 4	fila 4
fila 5	fila 5
fila 6	fila 6
fila 7	fila 7

Cuando n vale 0 el resultado de  $3n+1$  es 1, luego cuando n vale 1 el resultado es 4 y así sucesivamente.

Si queremos partir del segundo e ir avanzando tres filas luego debemos especificar el siguiente selector:

```
tr:nth-child(3n+2) {  
    background-color:red;  
}
```

Como resultado tenemos:

fila 1	fila 1
fila 2	fila 2
fila 3	fila 3
fila 4	fila 4
fila 5	fila 5
fila 6	fila 6
fila 7	fila 7

### - Pseudo-clase: nth-last-child:

El objetivo de esta pseudo-clase es igual a nth-child pero invierte el orden de los elementos HTML, es decir el primero es el último, el segundo es el anteúltimo y así sucesivamente.

Veamos con un ejemplo su funcionamiento, imprimamos de color rojo la última fila de una tabla, luego las dos anteriores dejarlas sin cambios, pintar la anterior a estas dos de rojo y así sucesivamente hasta el comienzo de la tabla.

Como podemos observar en este problema no sabemos cuantas filas tiene la tabla y queremos que siempre la última sea roja indistintamente la cantidad de filas de la misma. La pseudo-clase nth-child no nos sirve pero si tiene sentido nth-last-child:

```
<!DOCTYPE html>  
<html>  
<head>  
    <title>Problema</title>  
    <meta charset="UTF-8">
```

```

<link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<table>
  <tr>
    <td>fila 1</td><td>fila 1</td>
  </tr>
  <tr>
    <td>fila 2</td><td>fila 2</td>
  </tr>
  <tr>
    <td>fila 3</td><td>fila 3</td>
  </tr>
  <tr>
    <td>fila 4</td><td>fila 4</td>
  </tr>
  <tr>
    <td>fila 5</td><td>fila 5</td>
  </tr>
  <tr>
    <td>fila 6</td><td>fila 6</td>
  </tr>
  <tr>
    <td>fila 7</td><td>fila 7</td>
  </tr>
  <tr>
    <td>fila 8</td><td>fila 8</td>
  </tr>
</table>
</body>
</html>

```

```

tr:nth-last-child(3n+1) {
  background-color:red;
}

```

```

table {
  border-collapse:
collapse;
  width:80%;
  margin:0 auto;
}

```

```

td {
  border: 1px solid #000;
  text-align:center;
  padding:0.5rem;
}

```

fila 1	fila 1
fila 2	fila 2
fila 3	fila 3
fila 4	fila 4
fila 5	fila 5
fila 6	fila 6
fila 7	fila 7
fila 8	fila 8

Como vemos la regla de estilo queda definida:



```
tr:nth-last-child(3n+1) {  
  background-color:red;  
}
```

Cuando  $n$  vale cero tenemos:  $3 \cdot 0 + 1$  que nos genera el valor 1, luego el primer elemento, pero recorriendo desde el final es la fila 8.

### 34.3. Pseudo-clases: nth-of-type, nth-last-of-type, first-of-type y last-of-type

La pseudo-clase nth-of-type selecciona el elemento HTML que es el enésimo elemento hermano del mismo nivel de su TIPO.

En cambio, nth-child selecciona el elemento HTML que es el enésimo elemento hermano (teniendo en cuenta la posición de elementos de distinto tipo)

Veamos un ejemplo para entender la mecánica de funcionamiento de la pseudo-clase nth-of-type, si tenemos la siguiente página:

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Problema</title>  
  <meta charset="UTF-8">  
  <link rel="StyleSheet" href="estilos.css" type="text/css">  
</head>  
<body>  
  <div id="contenido">  
    <h1>Título</h1>  
    <p>Párrafo 1</p>  
    <p>Párrafo 2</p>  
    <p>Párrafo 3</p>  
  </div>  
</body>  
</html>
```

**Título**

Párrafo 1

Párrafo 2

Párrafo 3

Si queremos pintar de rojo el primer párrafo utilizando la pseudo-clase nth-child debemos especificar el siguiente selector:

```
#contenido p:nth-child(2) {  
  color:red;  
}
```

Es importante entender que en ese nivel hay cuatro elementos (un h1 y tres párrafos) y mediante nth-child debemos indicar el segundo elemento si queremos acceder al primer párrafo.

Entonces para este tipo de problemas donde los elementos HTML de un nivel sean de distintos tipos y queremos acceder solo a los de un determinado tipo debemos emplear este nuevo conjunto de pseudo-clases que solo tienen en cuenta los elementos del mismo tipo.

Para pintar de color rojo el primer párrafo contenido en el div debemos utilizar la sintaxis más adecuada:

```
#contenido p:nth-of-type(1) {  
  color:red;
```

```
}
```

### 34.4. Pseudo-clases: only-child y only-of-type

La pseudo-clase **only-child** se aplica cuando en dicho nivel hay un solo elemento. Por ejemplo:

```
<ul id="fechainicio">
  <li>10/10/2015</li>
</ul>
```

Luego si especificamos la siguiente regla:

```
#fechainicio li:only-child {
  color:blue;
}
```

El li se pinta de rojo ya que hay un solo elemento en ese nivel.

Si tenemos por ejemplo:

```
<div id="equipo1">
  <h1>Participantes</h1>
  <p>Pablo Martinez.</p>
</div>
```

Y aplicamos el estilo:

```
#equipo1 p:only-child {
  color:red;
}
```

No se selecciona el párrafo ya que no es el único elemento en dicho nivel (hay un h1 y un párrafo)

Ahora la pseudo-clase **only-of-type** se aplica cuando en dicho nivel hay un solo elemento de ese tipo (no importa cuantos otros elementos hay de otro tipo en el mismo nivel)

Si tenemos ahora:

```
<div id="equipo1">
  <h1>Participantes</h1>
  <p>Pablo Picasso.</p>
</div>
```

con la regla:

```
#equipo1 p:only-of-type{
  color:red;
}
```

Si se pinta de rojo el párrafo ya que en ese nivel hay solo un elemento de tipo párrafo.

Veamos una página y una hoja de estilo que prueban estas dos pseudo-clases:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
```

```

<body>
  <h1>Fecha de inicio.</h1>
  <ul id="fechainicio">
    <li>10/10/2023</li>
  </ul>
  <h1>Fechas finales.</h1>
  <ul id="fechafin">
    <li>10/12/2023</li>
    <li>24/12/2023</li>
  </ul>
  <div id="equipo1">
    <h1>Participantes</h1>
    <p>Pablo Picasso.</p>
  </div>
  <div id="equipo2">
    <h1>Participantes</h1>
    <p>Ana De Armas.</p>
    <p>Pedro Duque.</p>
  </div>
</body>
</html>

```

```

#fechainicio li:only-child {
  color:blue;
}

#fechafin li:only-child {
  color:blue;
}

#equipo1 p:only-of-type {
  color:red;
}

#equipo2 p:only-of-type {
  color:red;
}

```

## Fecha de inicio.

- 10/10/2023

## Fechas finales.

- 10/12/2023
- 24/12/2023

## Participantes

Pablo Picasso.

## Participantes

Ana De Armas.

Pedro Duque.

### 34.5. Pseudo-clase: empty

La pseudo-clase empty permite definir una regla css a un elemento html que no tenga elementos hijos y tampoco texto.

Por ejemplo:

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

Para probar esta pseudo-clase confeccionaremos una página que muestre un rectángulo amarillo cuando el div esté vacío. En caso que el div contenga datos no se mostrará el rectángulo amarillo.

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
  <div></div>
  <p>Esto es una prueba de la pseudo-clase empty.</p>
</body>
</html>
```

```
div:empty {
  width:80%;
  margin:0 auto;
  height:50px;
  background-color:yellow;
}
```

Este estilo se aplica a todos los div vacíos de la página. Tenemos como resultado:



Esto es una prueba de la pseudo-clase empty.

Si probamos agregar algún elemento HTML o texto dentro del div vemos que no cumple la regla de estilo y tenemos como resultado que no aparece el rectángulo amarillo:

Texto dentro del div.

Esto es una prueba de la pseudo-clase empty.

Hay que tener cuidado con los elementos HTML que parecen vacíos como por ejemplo:

```
<div>
</div>
```

Parece vacío el div pero en realidad tiene el salto de línea (estamos en presencia de texto en el div y por lo tanto no se verifica la regla empty)

Para que se aplique la regla de estilo con la pseudo-clase empty el div debe estar escrito en el archivo HTML:

```
<div></div>
```

## 35. Pseudo-elementos

---

### 35.1. Pseudo-elementos: first-letter y first-line

Los pseudo-elementos a diferencia de las pseudo-clases afectan a parte de un elemento HTML y no al elemento completo.

Los primeros dos pseudo-elementos que veremos son first-letter y first-line.

El pseudo-elemento first-letter permite seleccionar la primera letra del contenido de un elemento HTML y aplicarle un estilo. Veamos por ejemplo si queremos que la primera letra de un párrafo aparezca con un tamaño de fuente mayor y otro color:

```
<p>Esto es la prueba del pseudo-elemento first-letter.</p>
```

Definimos:

```
p::first-letter {  
  font-size:1.5rem;  
  color:red;  
}
```

Tenemos como resultado:

**E**sto es la prueba del pseudo-elemento first-letter.

La sintaxis para definir un pseudo-elemento es distinta a las pseudo-clases ya que le anteceden dos caracteres :: en lugar de uno.

Si no tuviéramos los pseudo-elementos este problema lo debemos resolver encerrando el primer carácter de un párrafo con el elemento HTML "span" y asignarle un estilo a dicho elemento. Luego gracias a esto tenemos un contenido del archivo HTML mucho más limpio.

#### - first-line:

El pseudo-elemento first-line permite seleccionar la primera línea de texto de un elemento HTML para aplicar un estilo.

Si tenemos la siguiente página:

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Problema</title>  
  <meta charset="UTF-8">  
  <link rel="StyleSheet" href="estilos.css" type="text/css">  
</head>  
<body>  
  <p>El pseudo-elemento first-line permite aplicar  
  un estilo a la primera línea del contenido de un  
  elemento HTML. Tengamos en cuenta que la cantidad  
  de datos a mostrar en la primera línea depende del  
  ancho de la ventana del navegador.</p>  
</body>  
</html>
```

```
p::first-line {
  text-decoration:underline;
}
```

Tenemos como resultado en el navegador:

El pseudo-elemento first-line permite aplicar un estilo a la primer línea del contenido de un elemento HTML. Tengamos en cuenta que la cantidad de datos a mostrar en la primer línea depende del ancho de la ventana del navegador.

Tengamos en cuenta que si redimensionamos el navegador solo los caracteres de la primera línea aparecen subrayados:

El pseudo-elemento first-line permite aplicar un estilo a la primer línea del contenido de un elemento HTML. Tengamos en cuenta que la cantidad de datos a mostrar en la primer línea depende del ancho de la ventana del navegador.

## 35.2. Pseudo-elementos: before y after

El pseudo-elemento **before** se utiliza para agregar algún contenido previo al contenido actual del elemento. Debemos inicializar la propiedad content con el dato a agregar.

El pseudo-elemento **after** agrega contenido al final del elemento.

Veamos un problema para entender su funcionamiento. Supongamos que queremos que cada vez que en la página utilizamos el elemento HTML strong aparezca su contenido precedido por el carácter "¡" y que finalice con el carácter "!". Podemos utilizar los Pseudo-elementos before y after que agreguen estos caracteres.

Teniendo la página:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
<p>SQL, Structure Query Language (Lenguaje de Consulta
Estructurado) es un lenguaje de programacion para
trabajar con base de datos relacionales como
<strong>MySQL</strong>, Oracle, etc.</p>

<p><strong>MySQL</strong> es un interpretador
de SQL, es un servidor de base de datos.</p>

<p><strong>MySQL</strong> permite crear base
de datos y tablas, insertar datos, modificarlos,
eliminarlos, ordenarlos, hacer consultas y
realizar muchas operaciones, etc., resumiendo:
administrar bases de datos.</p>
```

```
<p>Ingresando instrucciones en la línea  
de comandos o embebidas en un lenguaje como  
PHP nos comunicamos con el servidor. Cada  
sentencia debe acabar con punto y coma (;).</p>  
</body>  
</html>
```

```
strong::before {  
  content:"¡";  
}  
  
strong::after {  
  content:"!";  
}
```

Como podemos ver donde en la página definimos los elementos strong se le añadieron un carácter al principio y otro al final:

SQL, Structure Query Language (Lenguaje de Consulta Estructurado) es un lenguaje de programación para trabajar con base de datos relacionales como ¡MySQL!, Oracle, etc.

¡MySQL! es un interpretador de SQL, es un servidor de base de datos.

¡MySQL! permite crear base de datos y tablas, insertar datos, modificarlos, eliminarlos, ordenarlos, hacer consultas y realizar muchas operaciones, etc., resumiendo: administrar bases de datos.

Ingresando instrucciones en la línea de comandos o embebidas en un lenguaje como PHP nos comunicamos con el servidor. Cada sentencia debe acabar con punto y coma (;).

Podemos agregar cualquier contenido pero no elementos HTML (h1, h2, p, table etc.).

## 36. CSS media queries

Los **media queries** son una funcionalidad que aparece con CSS2 y se la mejora con CSS3.

Los media queries nos permiten definir una serie de reglas CSS dependiendo del medio donde se ejecuta (pantalla, impresora, tv, etc.) y las características propias del medio como puede ser el ancho en píxeles del dispositivo.

Actualmente donde más se los emplean es para implementar sitios adaptables al dispositivo donde se ejecutan: celular, tablet, pc etc.

Para ver su funcionamiento implementaremos una página que se adapte el "div" principal dependiendo del ancho del dispositivo.

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Problema</title>  
  <meta charset="UTF-8">  
  <link rel="StyleSheet" href="estilos.css" type="text/css">  
</head>  
<body>  
  <div class="container">
```

```
<h1>Título</h1>
<p>Datos del contenedor</p>
</div>
</body>
</html>
```

```
.container {
  width:100%;
  margin:0 auto;
  height:200px;
  background-color:#eee;
}

@media (min-width: 576px) {
  .container {
    max-width: 540px;
    background-color:#ddd;
  }
}

@media (min-width: 768px) {
  .container {
    max-width: 720px;
    background-color:#ccc;
  }
}

@media (min-width: 992px) {
  .container {
    max-width: 960px;
    background-color:#bbb;
  }
}

@media (min-width: 1200px) {
  .container {
    max-width: 1140px;
    background-color:#ccc;
  }
}
```

Una media query se define con la palabra clave "media" antecedida por el carácter @. Luego entre paréntesis indicamos la condición que debe cumplir una propiedad de la hoja de estilos. En caso que se verifique verdadero el valor de la propiedad se ejecuta todo el bloque encerrado entre las llaves:

```
@media (min-width: 576px) {
  .container {
    max-width: 540px;
    background-color:#ddd;
  }
}
```



Por ejemplo, si abrimos el navegador y lo redimensionamos a una medida inferior a 576 píxeles aparece algo similar a esto:



La condición (`min-width: 576px`) se verifica con falso, luego no se ejecuta la regla:

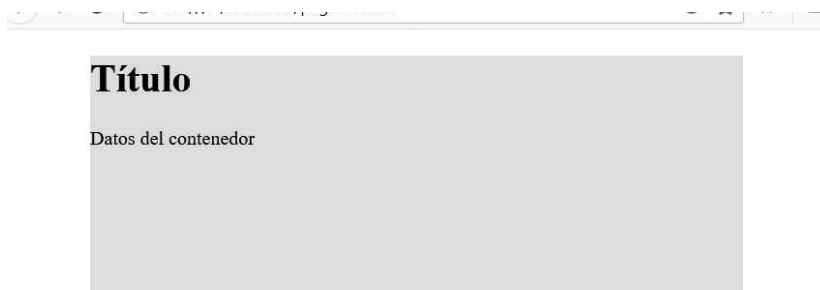
```
.container {  
  max-width: 540px;  
  background-color:#ddd;  
}
```

Es decir la clase container queda con el valor definido previamente:

```
.container {  
  width:100%;  
  margin:0 auto;  
  height:200px;  
  background-color:#eee;  
}
```

Ahora comencemos a aumentar el tamaño del ancho del navegador y podemos comprobar que cuando alcanza o supera el valor de 576 píxeles el ancho máximo se define con el valor 540px, esto debido a que se actualiza el estilo para la clase container por ser verdadero el media query definido (`min-width: 576px`):

```
.container {  
  max-width: 540px;  
  background-color:#ddd;  
}
```



Si seguimos aumentando el tamaño del ancho y alcanzamos o superamos los 768px luego también se verifica verdadera la segunda media query:

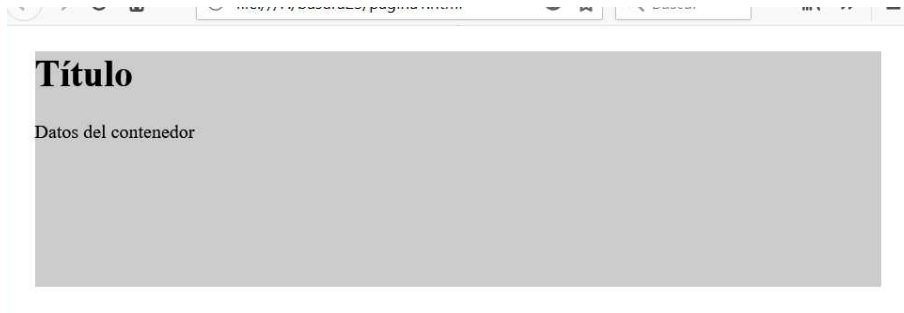
```
@media (min-width: 768px) {  
  .container {  
    max-width: 720px;  
  }
```

```

    background-color:#ccc;
}
}

```

Esto hace que el valor final para la propiedad max-width y background-color queden definidas con estos valores:



También definimos para tamaños mayores o iguales a 992px y 1200px:

```

@media (min-width: 992px) {
  .container {
    max-width: 960px;
    background-color:#bbb;
  }
}
@media (min-width: 1200px) {
  .container {
    max-width: 1140px;
    background-color:#ccc;
  }
}

```

Es importante el orden que definimos las media queries para este problema ya que por ejemplo si el dispositivo tiene un ancho de 1920 píxeles todas las condiciones se verifican verdaderas pero las propiedades max-width y background-color quedan definidas por la última asignación:

```

@media (min-width: 1200px) {
  .container {
    max-width: 1140px;
    background-color:#ccc;
  }
}

```

## Acotaciones

Podemos definir un media query para un determinado dispositivo, por ejemplo para una impresora:

```

@media print and (min-width:1000px) {
  ...
}

```

Cuando no disponemos el medio por defecto es para todos, el ejercicio anterior podíamos haberlo codificado como:

```
@media all and (min-width: 576px) {
  .container {
    max-width: 540px;
    background-color:#ddd;
  }
}
```

Es decir que si el media query se aplica a todos los medios podemos obviar la parte "all and".

Si analizamos el código fuente del framework [Bootstrap](#) veremos que hace un uso muy importante de los media queries para implementar el concepto de las páginas adaptables.

## 37. Flexbox

Flexbox nos permite implementar interfaces de usuario y maquetación de páginas web en forma más sencilla de como las veníamos haciendo del 2000 al 2014.

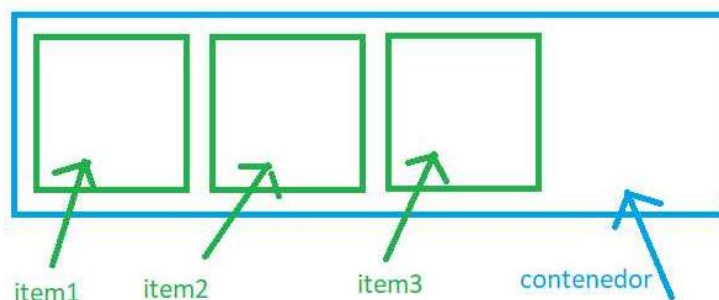
La mayoría de los navegadores ya implementan la funcionalidad de Flexbox o en castellano "Cajas flexibles".

Si queremos conocer su especificación técnica podemos hacerla en el sitio del [World Wide Web Consortium \(W3C\)](#), de todos modos no es un buen lugar para aprender de su funcionamiento.

Gran cantidad de diseñadores están empleando la funcionalidad de Flexbox y las nuevas versiones de framework de CSS como Bootstrap 4 también lo incorporan para su funcionamiento.

Flexbox nos facilita la maquetación del sitio web y todos sus componentes. También facilita la adaptación de la página a distintos tamaños de pantalla

Un Flexbox o caja flexible consiste en un elemento contenedor y en su interior una serie de ítems:



Para definir un elemento como contenedor de cajas flexibles debemos definir la propiedad display con el valor "flex", con esto ya tenemos una caja flexible.

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
```

```

<div id="contenedor1">
  <div>"Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum."</div>
  <div>"Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur."</div>
  <div>"Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum."</div>
</div>
</body>
</html>

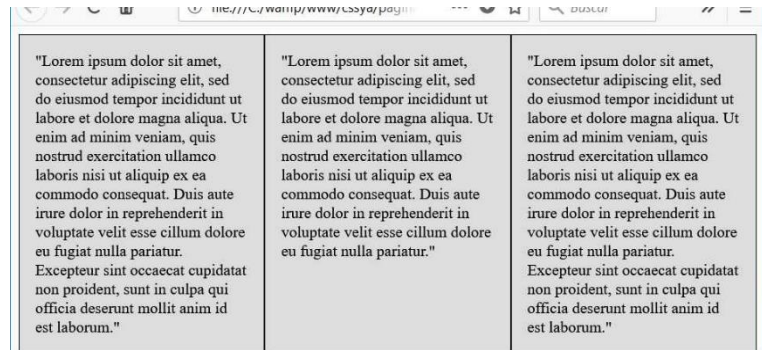
```

```

#contenedor1 {
  display: flex;
  background-color: #ddd;
}

#contenedor1 div {
  border: 1px solid black;
  padding: 1rem;
}

```



Debemos cargar en la propiedad "display" el valor "flex" para que el #contenedor1 se transforme en un contenedor de cajas flexibles:

```

#contenedor1 {
  display: flex;
  background-color: #ddd;
}

```

Luego los tres div contenidos se transforman en cajas flexibles que se redimensionan en ancho y alto automáticamente cuando cambiamos el tamaño del navegador.

Veremos que hay un conjunto de propiedades que nos permiten definir la ubicación, tamaño etc. de los ítems dentro del contenedor.

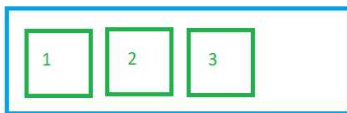
### 37.1. Flexbox - (flex-direction aplicada al contenedor)

La primera propiedad que podemos configurar cuando implementamos una caja flexible se llama flex-direction. Por defecto esta propiedad se inicializa con el valor "row".

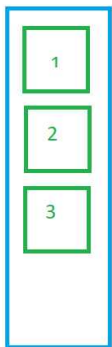
Los valores que podemos asignarle a la propiedad "flex-direction" son:

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

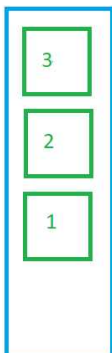
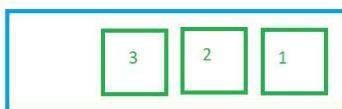
Cuando la propiedad flex-direction almacena el valor "row" los ítems se ubican uno al lado del otro es decir en forma horizontal y se respetan el orden de los ítems (valor por defecto):



Si almacenamos el valor "column" en la propiedad flex-direction luego los ítems se disponen uno debajo del otro:



Tenemos la posibilidad que los ítems se muestren al revés, es decir el último sea primero, el ante anteúltimo sea segundo y así sucesivamente:



Podemos probar que sucede visualmente con cada uno de estos cuatro valores de la propiedad flex-direction mediante este programa en Javascript :



El espacio en negro pertenece al contenedor pero las cajas flexibles no lo requieren debido a su poco contenido. Si agregamos en alguna de las cajas más contenido veremos que el espacio en negro disminuye o inclusive desaparece.

La página y su hoja de estilo para obtener dicho resultado es:

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
  <div id="contenedor1">
    <div id="item1">Caja 1</div>
    <div id="item2">Caja 2</div>
    <div id="item3">Caja 3</div>
  </div>
</body>
</html>
```

```
#contenedor1{
  display: flex;
  flex-direction: column;
  background-color: #000;
  height: 300px;
  color:white;
  font-size:2rem;
}

#item1 {
  background-color: #0078A6;
}
#item2 {
  background-color: #0084B6;
}
#item3 {
  background-color: #008CC1;
}
```

Si queremos que la caja flexible sea vertical es necesario asignar a la propiedad flex-direction el valor "column". Cuando es horizontal no es necesario asignar el valor "row" ya que es el valor por defecto que se inicia la propiedad.

### 37.2. Flexbox - (justify-content aplicada al contenedor)

La segunda propiedad que podemos cambiar a un contenedor de cajas flexibles se llama justify-content.

La propiedad justify-content controla como se distribuyen los ítems a lo largo del contenedor.

Por defecto la propiedad justify-content almacena el valor "flex-start".

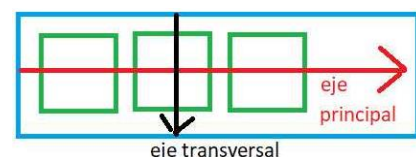
Los valores que podemos asignarle a la propiedad "justify-content" son:

- **flex-start**
- **flex-end**
- **center**
- **space-between**
- **space-around**

Para entender como se distribuyen los ítems en el contenedor podemos ejecutar este programa Javascript que modifica en forma dinámica la propiedad "justify-content", seleccione flex-direction el valor "row" y luego cambie por cada uno de los valores posibles la propiedad "justify-content":

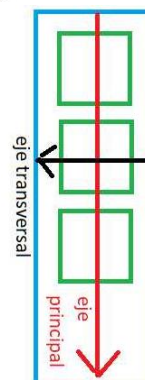


Cuando la propiedad flex-direction almacena el valor "row" el eje principal es el horizontal y el eje transversal es el vertical.



Cuando modificamos la propiedad "justify-content" los ítems se distribuyen con respecto al eje principal del contenedor.

Probemos ahora de seleccionar en la propiedad "flex-direction" el valor "column" y pasemos a ver que sucede con cada valor que le podemos asignar a "justify-content". Los espacios ahora aparecen en forma vertical, esto debido a que ha cambiado el eje principal y el eje transversal.



La propiedad "justify-content" tiene sentido cuando hay espacio no ocupado por los ítems y la distribución de dicho espacio es:

- **flex-start**: los ítems aparecen pegados al inicio del contenedor.
- **flex-end**: aparecen pegados al final.
- **center**: se agrupan todos en el centro.
- **space-between**: se distribuyen los ítems ocupando todo el espacio disponible, con espacios iguales entre ellos, pero sin dejar espacio al inicio y al final.
- **space-around**: se distribuyen los ítems ocupando todo el espacio disponible, con espacios iguales entre ellos, pero dejando espacio al inicio y al final.

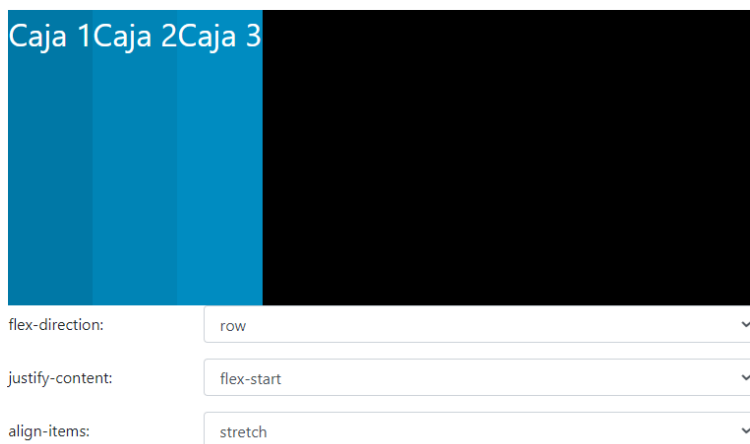
### 37.3. Flexbox - (align-items aplicada al contenedor)

La tercera propiedad que afecta al contenedor se llama "align-items". Es similar a la propiedad que vimos anteriormente "justify-content", pero la alineación es con respecto al eje transversal.

Los valores que podemos asignarle a la propiedad "align-items" son:

- **stretch**
- **flex-start**
- **flex-end**
- **center**
- **baseline**

Para entender como se distribuyen los ítems en el contenedor podemos ejecutar este programa Javascript que modifica en forma dinámica la propiedad "align-item":



El valor por defecto que almacena la propiedad "align-items" es "stretch" (stretch significa estirar, es decir el ítem ocupa en el eje secundario todo el espacio posible que permite el contenedor)

Los otros valores posibles puede probarlos modificando la propiedad "align-items" y pueden ser:

- **stretch** Los ítems ocupan toda la altura o anchura dependiendo del valor de la propiedad "flex-direction".
- **flex-start** Los ítems se ubican en la parte superior o izquierda del contenedor dependiendo del valor de la propiedad "flex-direction".
- **flex-end** Los ítems se ubican en la parte inferior o derecha del contenedor dependiendo del valor de la propiedad "flex-direction".



- **center** Los ítems aparecen centrados con respecto al eje transversal.
- **baseline** Los ítems se alinean respecto al eje transversal del contenedor teniendo en cuenta las líneas base de cada ítem, esta se define en base a los textos de los ítems.

### 37.4. Flexbox - (flex-wrap aplicada al contenedor)

La propiedad "flex-wrap" puede almacenar alguno de estos tres valores:

- **nowrap**
- **wrap**
- **wrap-reverse**

El valor por defecto que almacena la propiedad "flex-wrap" es "nowrap". Esto hace que todos los ítems se ubiquen siempre en el eje principal y en una misma línea independientemente a la cantidad de ítems que dispongamos.

Podemos hacer que el contenedor muestre ítems en varias líneas cuando no entran en una fila disponiendo en la propiedad "flex-wrap" el valor "wrap".



Como vemos en el ejemplo superior tenemos 7 cajas y las mismas si bien tienen un ancho de 100 píxeles, Flexbox se encarga de darles otro ancho para que sigan entrando en la misma fila debido a que la propiedad "flex-wrap" almacena "nowrap".

Probemos de modificar la propiedad "flex-wrap" con el valor "wrap" y podremos comprobar que cuando no entran más ítems en una línea continúan en la siguiente.

### 37.5. Flexbox - (align-content aplicada al contenedor)

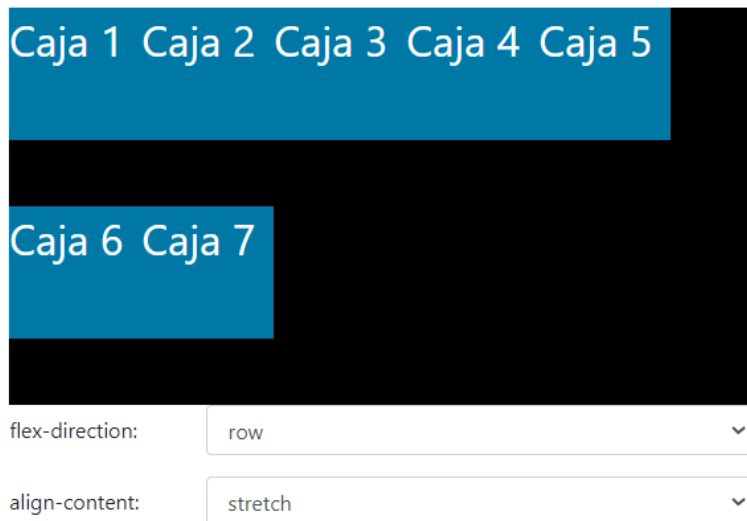
La propiedad "align-content" solo es aplicable cuando disponemos un Flexbox con varias líneas (es decir que hemos cargado el valor "wrap" en la propiedad "flex-wrap")

Los valores posibles de la propiedad "align-content" son:

- **stretch**
- **flex-start**
- **flex-end**
- **center**
- **space-between**
- **space-around**

Por defecto la propiedad "align-content" almacena el valor "stretch".

Probemos cambiar el valor "align-content" y veamos como se modifican las ubicaciones de los ítems:

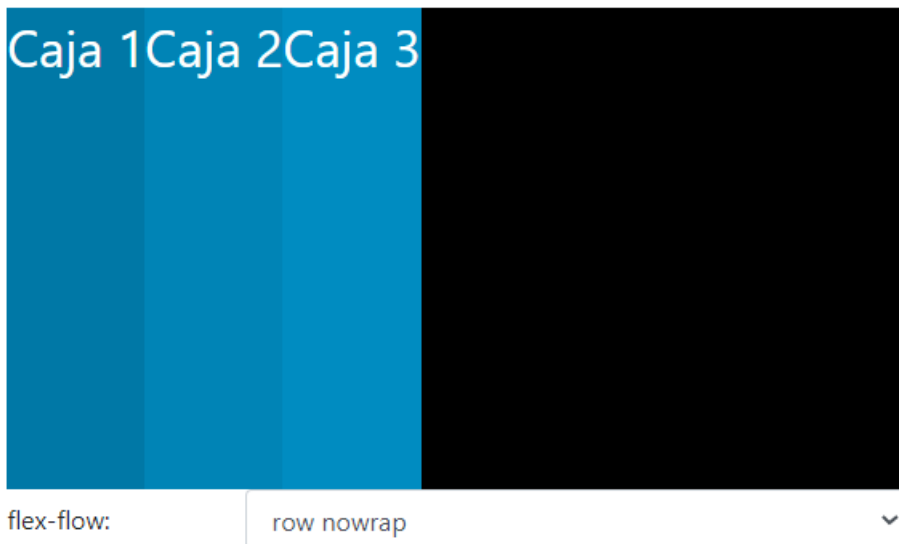


### 37.6. Flexbox - (flex-flow aplicada al contenedor)

La propiedad flex-flow es el formato resumido para inicializar las propiedades "flex-direction" y "flex-wrap".

La sintaxis para definir esta propiedad es:

flex-flow: "valor del flex-direction" "valor del flex-wrap".



```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
```

```

<div id="contenedor1">
  <div class="item">Caja 1</div>
  <div class="item">Caja 2</div>
  <div class="item">Caja 3</div>
  <div class="item">Caja 4</div>
  <div class="item">Caja 5</div>
  <div class="item">Caja 6</div>
  <div class="item">Caja 7</div>
</div>
</body>
</html>

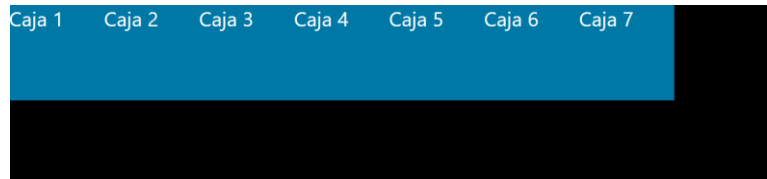
```

```

#contenedor1{
  display: flex;
  flex-flow: row wrap;
  background-color:
#000;
  height: 400px;
  color:white;
  font-size:2rem;
}

.item {
  background-color: #0078A6;
  width:150px;
  height:150px;
}

```



Vemos que se ha definido la propiedad flex-flow con el valor:

```
flex-flow: row wrap;
```

### 37.7. Flexbox - (flex-grow aplicada a los ítems)

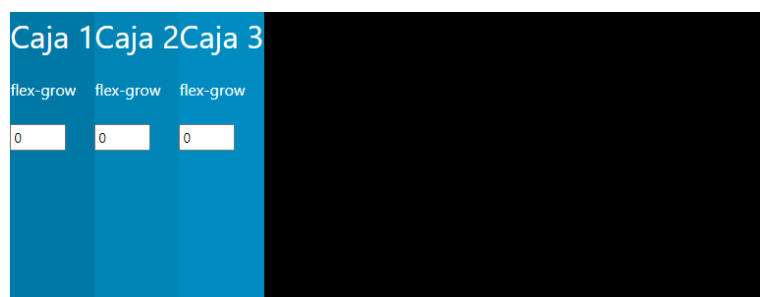
La propiedad flex-grow es la primera que veremos que se aplica a los elementos que están dentro del contenedor, es decir a los ítems.

Esta propiedad almacena un valor entero y controla que tanto crecerá el ítem rellenando el espacio no utilizado por los otros ítems.

El valor por defecto de la propiedad flex-grow es cero (no tiene una unidad de medida ya que indica un factor de crecimiento)

Veamos un ejemplo para entender esta propiedad. Si tenemos tres ítems probemos de definir el factor de crecimiento con el valor 1 para el primer ítem y dejemos con cero los otros dos flex-grow.

Inicialmente la propiedad flex-grow de cada ítem tiene un



valor 0 y podemos ver que a la derecha aparece en negro todo el espacio del contenedor no ocupado por las cajas flexibles. Inmediatamente que cambiamos el valor por un 1 para el flex-grow de la primera caja flexible comprobamos que todo ese espacio disponible es acaparado por dicha caja.

Probemos ahora de disponer también un 1 en la segunda caja. Comprobaremos que el espacio disponible se reparte entre las dos primeras cajas flexibles.

Un valor distinto a uno tiene sentido si queremos que se reparta ese espacio con otra proporción, por ejemplo si disponemos un 2 en el primer ítem y un 1 en el segundo ítem luego significa que el espacio disponible se reparte dando 2/3 partes para el primer ítem y 1/3 parte para el segundo ítem.

La propiedad flex-grow no puede almacenar un valor negativo.

Podemos realizar el ejemplo de definir un Flexbox con tres ítems. Hacer que el segundo ítem ocupe todo el espacio libre mediante la propiedad "flex-grow".

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
  <div id="contenedor1">
    <div id="item1">Caja 1</div>
    <div id="item2">Caja 2</div>
    <div id="item3">Caja 3</div>
  </div>
</body>
</html>
```

```
#contenedor1{
  display: flex;
  background-color: #000;
  height: 300px;
  color:white;
  font-size:2rem;
}

#item1 {
  background-color: #0078A6;
}
#item2 {
  flex-grow: 1;
  background-color: #0084B6;
}
#item3 {
  background-color: #008CC1;
}
```



Solo el elemento #item2 se inicializa la propiedad flex-grow:

```
flex-grow: 1;
```

Los otros dos ítems tienen por defecto el valor 0 en la propiedad flex-grow.

### 37.8. Flexbox - (flex-shrink aplicada a los ítems)

La segunda propiedad aplicada a los ítems de una caja flexible se llama "flex-shrink" y almacena un valor entero no negativo. Este valor representa un factor de encogimiento o contracción del ítem.

Cuando no hay más espacio para repartir en el Flexbox los ítems comienzan a contraerse para entrar en el contenedor.

El valor por defecto almacenando en la propiedad "flex-shrink" es el 1, esto quiere decir que cuando empezamos a reducir el ancho del contenedor los ítems se encogen en forma proporcional.

Si queremos que uno o más ítems no se encojan debemos asignarle a la propiedad "flex-shrink" el valor cero.

Probemos de asignar el valor 0 al segundo ítem:

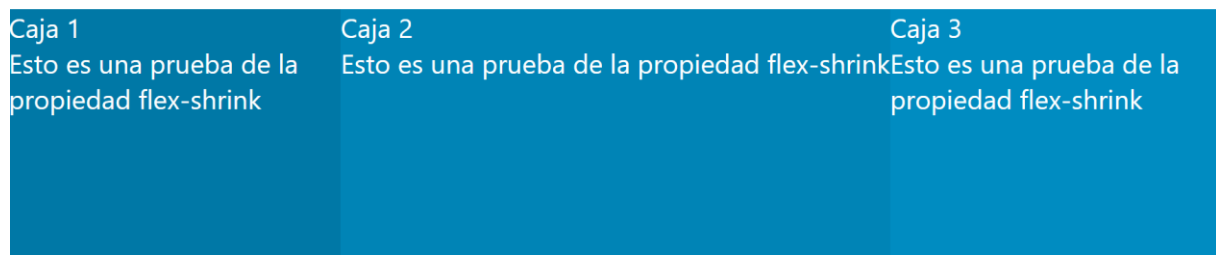


Ahora vamos a definir, como ejemplo, un Flexbox con tres ítems. Hacer que el segundo ítem no pueda encogerse empleando la propiedad "flex-shrink".

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
  <div id="contenedor1">
    <div id="item1">Caja 1<br>Esto es una prueba de la propiedad
flex-shrink</div>
    <div id="item2">Caja 2<br>Esto es una prueba de la propiedad
flex-shrink</div>
    <div id="item3">Caja 3<br>Esto es una prueba de la propiedad
flex-shrink</div>
  </div>
</body>
</html>
```

```
#contenedor1{
  display: flex;
  background-color: #000;
  height: 300px;
  color:white;
  font-size:2rem;
}

#item1 {
  background-color: #0078A6;
}
#item2 {
  flex-shrink: 0;
  background-color: #0084B6;
}
#item3 {
  background-color: #008CC1;
}
```



Solo al elemento `#item2` le asignamos un valor 0 a la propiedad "flex-shrink", las otras almacenan un valor 1 y por lo tanto se pueden contraer:

```
flex-shrink: 0;
```

### 37.9. Flexbox - (flex-basis aplicada a los ítems)

La propiedad flex-basis define el ancho con respecto al eje principal (si el contenedor es horizontal luego flex-basis representa el width, en cambio si el contenedor está en forma vertical la propiedad flex-basis representa el height)

El tamaño base que definimos en la propiedad flex-basis puede no llegar a cumplirse dependiendo de las otras propiedades vistas que se aplican a los ítems.

Esta propiedad es fundamental para definir los tamaños de los ítems. Podemos emplear las diferentes unidades de medida que conocemos píxeles, rem, em, % etc.

El valor por defecto que almacena la propiedad "flex-basis" es "auto", es decir que se calcula su medida en forma automática.

Probemos de cambiar la propiedad "flex-basis" para los tres ítems con los valores 20% para el primer ítem 60% para el segundo y 20% para el tercero:



Como ejemplo construiremos un Flexbox con tres ítems. Hacer el primero y último ítem tengan una medida de 20% y un valor 60% para el ítem central.

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
  <div id="contenedor1">
    <div id="item1">Caja 1</div>
    <div id="item2">Caja 2</div>
    <div id="item3">Caja 3</div>
  </div>
</body>
</html>
```

```
#contenedor1{
  display: flex;
  background-color: #000;
  height: 300px;
  color:white;
  font-size:2rem;
}

#item1 {
  flex-basis: 20%;
  background-color: #0078A6;
}
#item2 {
  flex-basis: 60%;
  background-color: #0084B6;
}
#item3 {
  flex-basis: 20%;
  background-color: #008CC1;
}
```



### 37.10. Flexbox - (flex aplicada a los ítems)

La propiedad flex es el formato resumido de las propiedades "flex-grow", "flex-shrink" y "flex-basis".

El orden de valores en caso que indiquemos los tres debe ser:

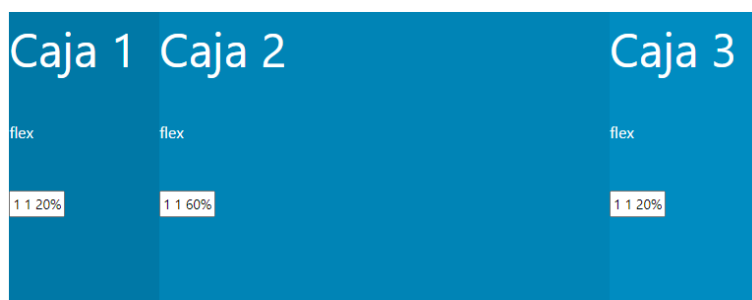
`flex: valor del flex-grow valor del flex-shrink valor del flex-basis;`

Ejemplos:

`flex: 1 0 60%;`

Estamos cargando un 1 en la propiedad flex-grow, un 0 en la propiedad flex-shrink y un 60% en flex-basis.

Probar distintos valores y ver como responden las cajas flexibles:



Hay posibilidades de inicializar la propiedad flex con menos de tres valores y que los otros queden con valores por defecto:

`flex: 1 2; /* flex-basis queda con su valor por defecto */`

`flex: 1 20%; /* flex-shrink queda con su valor por defecto */`

`flex: none; /* se almacenan los valores 0 0 auto*/`

`flex: auto; /* Especificamos los valores por defecto de cada propiedad */`

Ahora, un ejemplo: Definir un Flexbox con tres ítems. Inicializar la propiedad flex de cada ítem.

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
  <div id="contenedor1">
    <div id="item1">Caja 1</div>
```



```
<div id="item2">Caja 2</div>
<div id="item3">Caja 3</div>
</div>
</body>
</html>
```

```
#contenedor1{
  display: flex;
  background-color: #000;
  height: 300px;
  color:white;
  font-size:2rem;
}

#item1 {
  flex: 1 1 20%;
  background-color: #0078A6;
}
#item2 {
  flex: 0 0 60%;
  background-color: #0084B6;
}
#item3 {
  flex: 0 1 20%;
  background-color: #008CC1;
}
```



### 37.11. Flexbox - (order aplicada a los ítems)

La propiedad "order" de los ítems nos permiten indicar en que orden deben aparecer los ítems en el contenedor.

Por defecto la propiedad "order" almacena el valor 0. Los ítems aparecen dentro del contenedor en el mismo orden que los escribimos en el archivo html.

Podemos cambiar el flujo de los ítems dentro del contenedor indicando en la propiedad order valores enteros a partir de 0.

Cambiar la propiedad "order" del primer ítem por el valor 2 y el del último ítem por el valor 0. El ítem central fijar el valor 1:



Definiremos un Flexbox con tres ítems. Mediante la propiedad "order" cambiar hacer que aparezcan en forma inversa a como se los dispuso en el archivo HTML.

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
  <div id="contenedor1">
    <div id="item1">Caja 1</div>
    <div id="item2">Caja 2</div>
    <div id="item3">Caja 3</div>
  </div>
</body>
</html>
```

```
#contenedor1{
  display: flex;
  background-color: #000;
  height: 300px;
  color:white;
  font-size:2rem;
}

#item1 {
  order: 2;
  background-color: #0078A6;
}
#item2 {
  order: 1;
  background-color: #0084B6;
}
#item3 {
  order: 0;
  background-color: #008CC1;
}
```



Caja 3Caja 2Caja 1

### 37.12. Flexbox - (align-self aplicada a los ítems)

La propiedad align-self de un ítem nos permite especificar la alineación dentro del contenedor flexible respecto al eje transversal.

Los valores que podemos asignarle a la propiedad "align-self" son:

- **auto**
- **flex-start**
- **flex-end**
- **center**
- **stretch**
- **baseline**

La propiedad "align-self" almacena por defecto el valor "auto".

Probemos a cambiar a los distintos valores posibles a la propiedad "align-self":



Como ejemplo vamos a definir un Flexbox con 6 ítems. Especificar un valor distinto para cada uno de los ítems en la propiedad "align-self"

```
<!DOCTYPE html>
<html>
<head>
  <title>Problema</title>
  <meta charset="UTF-8">
  <link rel="StyleSheet" href="estilos.css" type="text/css">
</head>
<body>
  <div id="contenedor1">
    <div id="item1">Caja 1</div>
    <div id="item2">Caja 2</div>
    <div id="item3">Caja 3</div>
    <div id="item4">Caja 4</div>
    <div id="item5">Caja 5</div>
    <div id="item6">Caja 6</div>
  </div>
```

```
</body>
</html>
```

```
#contenedor1{
  display: flex;
  background-color: #000;
  height: 300px;
  color:white;
  font-size:2rem;
}

#item1 {
  align-self: auto;
  background-color: #0078A6;
}
#item2 {
  align-self: flex-start;
  background-color: #0084B6;
}
#item3 {
  align-self: flex-end;
  background-color: #008CC1;
}

#item4 {
  align-self: center;
  background-color: #0078A6;
}
#item5 {
  align-self: stretch;
  background-color: #0084B6;
}
#item6 {
  align-self: baseline;
  background-color: #008CC1;
}
```

