

## HTML

### Introducción a HTML

HTML es el bloque básico con el que está construido internet. Todas las páginas web utilizan HTML. *No es un lenguaje de programación propiamente dicho, sino, es un lenguaje de Markup:* son lenguajes que incorporan al texto marcas o etiquetas que luego son interpretadas para darle información extra sobre la estructura del texto. En el caso de HTML, este será interpretado por los browsers, que también presentarán el código en forma gráfica.

HTML es la abreviatura de Hyper Text Markup Language:

- Hyper Text: "Hyper Texto" quiere decir [texto con links](#)
- Markup Language: Los "Lenguajes de Marcado" son lenguajes de programación basados en etiquetas que uno agrega a un texto para darle estructura e información adicional. A diferencia de los "Lenguajes de Scripting" que se usan para crear programas informáticos, los lenguajes de marcado son sólo reglas para ordenar un documento.

### Elementos básicos HTML

HTML define una serie de elementos (o etiquetas, o tags) que servirán para delimitar texto. Cada tag está encerrado en `< >` y tiene un nombre. Los tags se abren y se cierran, los tags de cerrado incluyen con un `/` en el principio del tag que cierra. Por ejemplo:

```
<element>
...
</element>
```



Algunos tags html, por su naturaleza, no necesitan tener nada *adentro*. Por lo tanto podemos abreviar su escritura y en vez de abrir y cerrar el tag, simplemente agregamos un `/` antes del bracket final.

```

```

### Atributos

En su mayoría de los atributos de un elemento son pares *nombre-valor*, separados por un signo de igual `=` y escritos en la etiqueta de comienzo de un elemento, después del nombre del elemento. El valor puede estar rodeado por comillas dobles o simples. Los atributos de los tags nos sirven para cambiar su comportamiento o *configurarlos*.

Por ejemplo, el tag `<img>` sirve para mostrar una imagen. Este tag recibe el atributo `src` que indica la [URL](#) de donde está la imagen que queremos mostrar.



## Tags

### **<html>**

El tag `<html>` va a contener a todos los demás tags dentro suyo. Este tag básicamente sirve para avisarle al browser que el contenido debe ser interpretado como `html`.

### **<head>**

Este tag sirve para contener tags que contengan información sobre el documento, pero es información que no queremos que se renderice. Comunmente contiene el *título* de la página y *links* a recursos externos que pueda usar la página (javascript o css).

### **<title>**

Es el título de la página, se mostrará en el tab del browser o en la parte superior (pero no en la página).

### **<body>**

En este tag estará encerrado todo lo que queramos que se vea en la pantalla.

Entonces, hasta ahora, un documento HTML se ve así:

```
<html>
  <head>
    <title>Es el título de nuestra página</title>
  </head>
  <body>
  </body>
</html>
```

Como ven, para mayor facilidad en la lectura y la estructuración del documento, el documento HTML se escribe [indentando \(o usando sangría\)](#).

(Todos los tags que presentaremos más abajo van siempre adentro de un tag `<body>` )

### **<p>**

Es el tag para los párrafos. Mostrará el texto contenido dentro en una nueva línea.

```
<p>Soy un párrafo</p>
```

## **`<span>`**

El elemento `span` es un contenedor de texto genérico. No inserta una nueva línea, como lo hace el elemento `p`. Sirve básicamente para darle estilo al texto.

## **`<div>`**

El elemento `div` es un *contenedor* genérico. Es usado principalmente para dar estilo, imaginen que es una caja (cuyo tamaño y color podés modificar *a piacere*), y que dentro podés poner otras cajas iguales.

## **`<a>`**

El tag `a` (del inglés *anchor*), nos permite crear *links* a otros documentos y páginas. Este tag recibe el *atributo* `href` que indica a dónde apunta el link.

```
<a href="http://www.soyhenry.com">Esto es un link!</a>
```

## **`<h1> ... <h6>`**

Son los tags de encabezado o títulos, están pensados del 1 al 6, para indicar la importancia del contenido y su jerarquía.

```
<h1>El título más importante!</h1>
<h3>título medianamente importante.</h3>
<h6>El título menos importante.</h6>
```

## **`<img>`**

Este tag nos permite mostrar imágenes en la pantalla. Necesita el atributo `src` que indica la *URL* de donde sacar la imagen a mostrar.

```

```

## **`<ul>`**

Este tag representa una lista desordenada (del inglés "unordered list"). Este tag está diseñado para contener otros tags de tipo *item*. También existe el tag `<ol>` que viene de "ordered list".

## **`<li>`**

Son los tags que contienen los *items* de la lista ("list item").

```
<ul>
  <li>
    <span>Elemento uno</span>
  </li>
  <li>
    <p>Podemos anidar cualquier tipo de tag adentro</p>
  </li>
  <li>
    <span>tercer elemento</span>
  </li>
</ul>
```

# CSS

## Introducción a CSS

Como vimos, HTML nos sirve para dar estructura al contenido. En las primeras épocas de internet las páginas eran así. De hecho, todavía esta online la [primera página web](#). Como ven es bastante aburrida.

Luego se introdujo el concepto de CSS (Cascading Style Sheets); una forma de poder agregar color y estilos en nuestras páginas!

### Reglas CSS

Básicamente una regla CSS está compuesta por un atributo o propiedad y un valor. Según el atributo que usemos y el valor que le pongamos a ese atributo vamos a obtener resultados visuales distintos en nuestro html.

Por ejemplo:

```
html {  
  color: red; /* "Color" es la propiedad y "red" el valor */  
  font-size: 12px; /* "font-size" es la propiedad y "12px" el valor */  
}
```

En este ejemplo, vemos dos atributos: `color` y `font-size`, el primero permite modificar el color del texto, en este caso está seteado a `red`; el segundo indica el tamaño del texto, en este caso `12px`.

Es importante notar que distintos atributos pueden recibir distintos valores, generalmente los que indican un color reciben un color (`red`, `blue`, etc...), los que son medidas reciben una medida (`12px`, `15px`, etc...), y hay otras propiedades que reciben valor específicos, por ejemplo: la propiedad `border` (que dibuja un borde alrededor de un elemento) recibe tres valores: el color del borde, el ancho de la línea y el tipo de línea (punteada, continua, etc...).

Hay muchos atributos CSS disponibles, más de los que podemos recordar. Así que no se asusten, con el tiempo van a empezar a memorizar estas propiedades. Pueden ver una lista completa [acá](#).

### Formas de dar estilo

Antes de empezar a dar estilos, necesitamos una forma de decirle al browser qué vamos a darles reglas de estilo.

Hay varias formas de lograr esto (más adelante veremos en detalle como funcionan cada una):

- usando el atributo `style`: esta es la forma primitiva más simple, básicamente le damos reglas a cada tag html.
- usando el tag `<style/>`: Se utiliza este tag en el `<head>` del documento HTML, con esto logramos agrupar todas las reglas que luego queremos que se apliquen a los elementos HTML.
- Usar el tag `<link/>`: Este método nos permite definir las reglas CSS en un documento separado e *importarlo* a nuestra página (la ventaja que tiene es que podemos importar el mismo CSS a varias páginas).

### Atributo style

Todos los tags HTML pueden recibir el atributo `style`. Este atributo indica las reglas CSS (que veremos más abajo), que se aplicaran sólo al elemento que las tiene.

```
<h1 style="color:blue;">Esto es un título Azul</h1>
```

Pros:

- Fácil de escribir y leer.
- Cómo se aplican a un sólo elemento no hay forma de confundirse y que se aplique la regla a un elemento no deseado.

Cons:

- La regla aplica a un sólo elemento, si quisieramos que varios elementos tengan la misma regla, deberíamos copypastear!

### `<style>`

El tag `<style>`, que se escribe en el `<head>` del documento, nos permite escribir reglas que se aplicaran a uno o varios elementos html. Es importante notar que con esta forma, podremos darle estilo a muchos elementos de una sola vez, pero sólo a elementos que estén en el mismo documento.

```
<html>
  <head>
    <style>
      /*!-- acá van las reglas -->*/
    </style>
  </head>
  <body>
  </body>
</html>
```

Pros:

- Lugar central donde podemos escribir las reglas CSS del documento
- Podemos compartir reglas entre varios elementos iguales

Cons:

- No podemos compartir las reglas con *otro* documento HTML.
- Hay que prestar atención a las reglas, y a qué elementos se aplican.

## <link>

Con el tag `<link>` dentro del `<head>` del documento, vamos a poder *importar* un archivo css que contenga varias reglas CSS. Funciona similar al tag `<style/>` anterior. Pero ahora tenemos la ventaja que podemos *compartir* el mismo archivo css con varios documentos HTML.

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
  </body>
</html>
```

Pros:

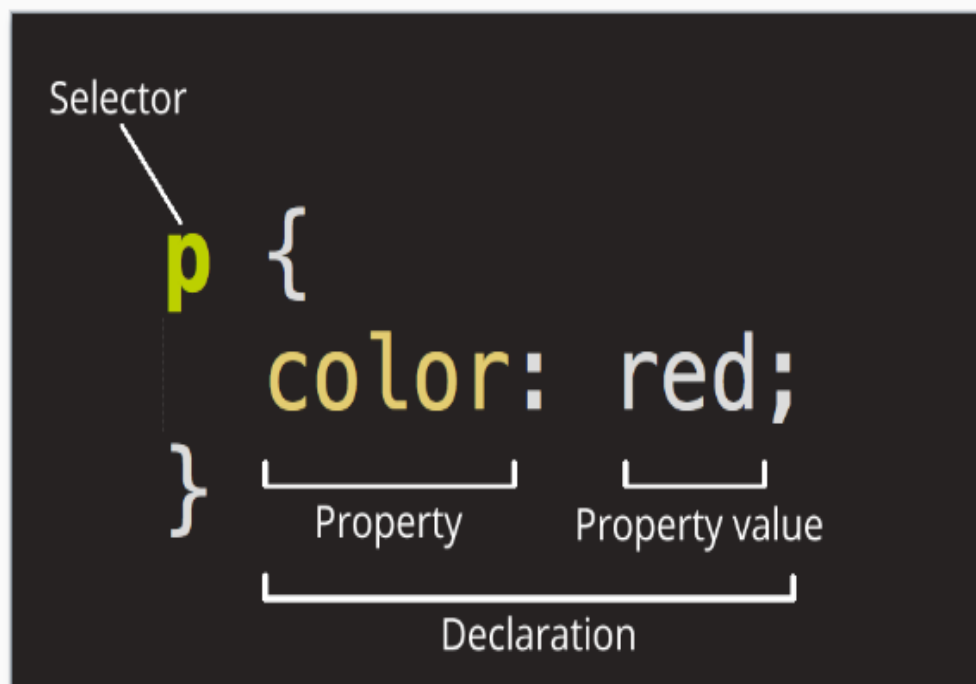
- Lugar central donde podemos escribir las reglas CSS del documento
- Podemos compartir reglas entre varios elementos iguales
- Podemos compartir reglas entre varios documentos HTML

Cons:

- Hay que prestar atención a las reglas, y a qué elementos se aplican.

## Selectores

Para poder aplicar reglas de estilo a los elementos html, necesitamos una forma de saber cómo seleccionar los elementos a los que deseamos aplicar las reglas, para esto sirven los *selectores CSS*.



Hay varios tipos de selectores, los más básicos son los de tipo, donde indicamos a qué clase de elementos se van a aplicar las reglas, el ejemplo de arriba usa un selector de tipo. Está diciendo: *aplicarle a todos los elementos de tipo `<p>` la regla de texto color rojo*.

El selector de tipo se puede usar con cualquier tipo de tag: `p`, `div`, `body`, etc. Otra forma de usar selectores poniéndole un *nombre o identificador* a cada elemento HTML. Para esto existe un *atributo* que pueden recibir todos los tags llamados: `id` y `class`.

**Id**: son nombre que sólo pueden aparecer una sólo vez en el documento. Es super específico y sirve para seleccionar UN solo elemento en particular.

**Class**: podemos asignarle el nombre de una clase a un grupo de elementos html.

### Selectores básicos

- Selector de tipo: Selecciona todos los elementos que coinciden con el nombre del elemento especificado.  
Sintaxis: `elname`  
Ejemplo: `input` se aplicará a cualquier elemento `<input>`.
- Selector de clase:  
Selecciona todos los elementos que tienen el atributo de class especificado.  
Sintaxis: `.classname`  
Ejemplo: `.index` seleccionará cualquier elemento que tenga la clase "index".
- Selector de ID  
Selecciona un elemento basándose en el valor de su atributo id. Solo puede haber un elemento con un determinado ID dentro de un documento.  
Sintaxis: `#idname`  
Ejemplo: `#toc` se aplicará a cualquier elemento que tenga el ID "toc".
- Selector universal  
Selecciona todos los elementos. Opcionalmente, puede estar restringido a un espacio de nombre específico o a todos los espacios de nombres.  
Sintaxis: `*[ns|]`  
Ejemplo: `*` se aplicará a todos los elementos del documento.
- Selector de atributo  
Selecciona elementos basándose en el valor de un determinado atributo.  
Sintaxis: `[attr] [attr=value] [attr~=value] [attr|=value] [attr^=value] [attr$=value] [attr*=value]`  
Ejemplo: `[autoplay]` seleccionará todos los elementos que tengan el atributo "autoplay" establecido (a cualquier valor).

## Anatomía de las reglas de estilo

Ahora que sabemos como *seleccionar* los elementos a los que queremos aplicar las reglas podemos escribir qué reglas queremos que se apliquen. Para el ejemplo vamos a usar la etiqueta `<style>`.

```
<style>
  body {}

  .divClass {}

  #divId {}
</style>
```

En el ejemplo de arriba vemos tres selectores. El primero es para el elemento `body`, el segundo para todos los elementos de la clase `divClass` y el tercero para el elemento que tenga el id: `divId`. Dentro de los `{ }` vamos a escribir todas las reglas que queremos que se apliquen a esos elementos.

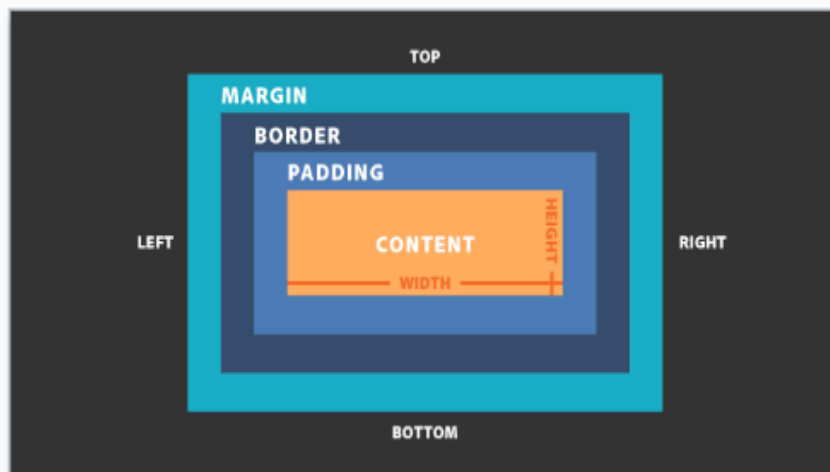
## Aplicando reglas

Ahora que tenemos los elementos seleccionados podemos empezar a agregar las reglas que habíamos visto antes.

```
div {
  propiedad: valor;
}
```

## Box Model

Para poder entender y luego manipular la forma en que los elementos HTML aparecen distribuidos en la página, tenemos que aprender cómo son representados estos en el browser.



En un documento html cada elemento es representado como una *caja rectangular* y en CSS cada una de estas cajas tiene 4 capas que podemos manipular. Esto se conoce como *Modelo de Caja* o *Box Model*.

Yendo desde afuera hacia adentro, las capas son:

- `margin`: el espacio que separa al elemento de los otros elementos. Si los pensamos como cajas, es el espacio entre las cajas.
- `border`: el "borde de la caja". Podemos hacerlo visible con diferentes grosores, estilos y colores, como ya hicimos varias veces en ejercicios anteriores.
- `padding`: el espacio entre el borde de la caja y su contenido. En la metáfora de la caja, podríamos por ejemplo tener una caja grande con algo chiquito adentro, osea que "habría mucho padding".
- `content`: el contenido de la caja. Por ejemplo el texto en un h1, otros tags anidados, etc, todo lo que esté contenido en el elemento.

## height y width

Podemos decirle al navegador exactamente qué tan *alto* y *ancho* queremos que sea nuestro elemento (contenido), esto se usa en `div`, `img` y otros elementos basados en la altura (para determinar el tamaño del texto, necesitaremos usar una propiedad de estilo diferente). Los valores de tamaño pueden estar en muchas medidas diferentes, pero el más común es el píxel "px".

```
div {  
  height: 400px;  
  width: 400px;  
}
```

## margin

El margen es el área transparente alrededor del elemento que deseas que no choque con nada. Es la capa más externa en el Modelo de caja.

## border

Borde establecerá un *borde* alrededor de su elemento, puedes determinar el tamaño, color y estilo del borde. Puede encontrar una lista de estilos de borde aquí: <https://developer.mozilla.org/en-US/docs/Web/CSS/border>. El borde está fuera del padding, pero dentro del margen.

```
div {  
  border: 1px solid black;  
}
```

## padding

El padding es el área transparente entre el borde y el contenido, es similar al margen, pero para adentro



## Cálculo del box model

Cuando establecemos el *alto* y el *ancho* de un elemento a través de la regla css `height` y `width`, sólo estamos configurando el contenido. Para calcular la altura y el ancho reales, tenemos que tener en cuenta el *padding*, el *borde* y el *margen*.

- `padding` es un área transparente alrededor del contenido.
- `border` se envolverá alrededor del relleno
- `margin` es el área transparente más externa que envuelve toda la caja.

Por ejemplo. Si establecemos la altura del contenido en 20 px y el ancho en 20 px, el relleno en 5 px, el borde en 1 px y el margen en 10 px.

Altura real = 25px (contenido) + 2 \* 5px (relleno, cada lado) + 2 \* 1 (borde de cada lado) + 2 \* 10 (margen, cada lado) = 57px

Ancho real = 25px (contenido) + 2 \* 5px (relleno, cada lado) + 2 \* 1 (borde de cada lado) + 2 \* 10 (margen, cada lado) = 57px

Saber esto nos ayudará a dimensionar y posicionar nuestros elementos correctamente.



## Otras propiedades CSS

### *background*

El `background` se puede establecer en una variedad de reglas, la más común sería establecer el fondo en un color o una imagen. Ambos se muestran a continuación.

```
.divClass {  
  background: red;  
}  
  
#divId {  
  background: url ('http://imageurl.com/image.jpg');  
}
```

### *color*

El color se usa sólo para texto. Establecerá el color de tu texto

### *font-size*

No podemos usar ancho o alto para el texto, pero podemos determinar el tamaño de la fuente utilizada. Puede usar cualquier unidad de tamaño aquí que usaría con una fuente en un procesador de textos (px, em, in, etc.). El más popular es px.

## Hojas de estilo externas y el elemento `<link>`

Hemos explicado cómo usar el elemento html `<style>`. Esto está bien si tiene una página web muy pequeña y un estilo mínimo, pero la mayoría de las páginas comenzarían a sentirse abarrotadas muy rápidamente si incluimos todo nuestro CSS en el HTML. Afortunadamente, tenemos una solución para eso, hojas de estilo externas y el elemento `<link>`.

Una hoja de estilo externa es simplemente otro archivo con el tipo de archivo `.css`. Convencionalmente, este archivo se llama algo así como "style.css". Podemos tomar todas las reglas de estilo que escribimos entre las etiquetas `<style>` y transferirlas directamente al archivo css. No necesitamos incluir nada más, solo las reglas de estilo.

Una vez que tengamos una hoja de estilo externa creada, necesitaremos asegurarnos de que el navegador lea ese archivo y aplique las reglas a nuestra página. Le decimos al navegador que busque ese archivo utilizando el elemento `<link>`. Podemos eliminar las etiquetas `<style>` y en su lugar agregar el elemento `<link>`. Dentro del elemento de enlace, necesitaremos proporcionar la ubicación y el tipo de archivo que estamos vinculando. Utilizaremos dos banderas, la bandera "rel" y la bandera "href".

La bandera rel solo le dirá al navegador qué tipo de archivo es y cómo procesarlo. En nuestro caso lo configuraremos como "hoja de estilo"

La bandera href le dirá al navegador dónde encontrar el archivo. Si el archivo está en la misma carpeta que nuestro archivo html, podemos configurarlo en: `"/styles.css"` (esta ruta será relativa)

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

Ahora que tenemos nuestra hoja de estilo externa vinculada a nuestro archivo HTML, deberíamos ver las reglas de estilo que establecemos reflejadas en nuestra página.

# Introducción al posicionamiento

Armar un layout y hacer que todo se vea limpio es lo que la mayoría de gente espera hacer cuando empiezan a aprender CSS. Posicionar elementos HTML en la página con CSS es posiblemente la habilidad más poderosa que tiene CSS, aunque también puede ser la más frustrante. En esta lección aprenderemos distintas formas de posicionar elementos en la página.

## La propiedad *display*

Esta propiedad es una de las más importante de posicionamiento en CSS. Podemos usarla para controlar cómo se muestra el contenido en relación a los elementos alrededor de este, y cómo se comportan en la pantalla.

```
div {  
  display: <valor de la propiedad>;  
}
```

Hay dos tipos de elementos "display" ya incluidos en HTML; "block" e "inline",

### Elementos *block*

Un elemento "block" siempre arrancará en una nueva línea, y siempre tomará el ancho máximo del contenedor en el cual se encuentre. ¿Recuerdas que en la lección anterior aprendimos que el elemento `<p>` siempre empieza en una nueva línea? Es porque es un elemento "block", como también lo son los `div` y los `<h1-6>`.

### Elementos *inline*

Los elementos "inline" son opuestos a los "block", dado que no comenzarán en una nueva línea y sólo tomarán el espacio suficiente que necesiten para mostrar la información dentro del mismo. Los elementos `<span>`, `<a>` e `<img>` son inline.

Podemos controlar cómo un elemento se comporta usando la propiedad "display". Si queremos que un elemento "inline" actúe como uno "block", le definimos:

```
div {  
  display: block;  
}
```

Y vice-versa.

### *none*

Definir como valor "none" hará que el elemento desaparezca completamente. No debe ser confundido con la regla `visibility: hidden;` dado que esta hace que el elemento sea invisible pero no lo elimina de la página (el espacio continuará ocupado por algo). Si un elemento está definido como `display: none;` no habrá signos de él en la página.

### *flex*

Flex es una nueva herramienta que nos ofrece CSS3, la cual nos da la habilidad de controlar en qué parte de la página queremos que estén nuestros ítems. Hablaremos sobre ella más adelante en esta lección.

### *grid*

Esta es una característica de CSS3 que nos permitirá crear sistemas de grillas dentro de un elemento.

## La propiedad *position*

Esta propiedad especificará qué tipo de método de posicionamiento usará un elemento HTML. Hay 5 métodos diferentes disponibles (aquí veremos 4 de ellos).

```
div {  
  position: <valor de la propiedad>;  
}
```

### *static*

Este es el posicionamiento por defecto de un elemento, definir un elemento como "estático" no afectará al comportamiento del mismo de ninguna manera.

### *relative*

Usar este valor mantendrá el elemento posicionado como si fuese estático, pero nos permitirá usar otros métodos de posicionamiento de elementos que veremos enseguida.

### *fixed*

Definir un elemento como "fijo" hará que éste quede fijo en un lugar de la pantalla, sin importar cuánto naveguemos, "scroleemos" o movamos la pantalla, el elemento se quedará en ese lugar. Los casos de uso pueden ser un header o la barra de navegación de un sitio web.


### *absolute*

"absolute" es muy parecido a "fixed", excepto que el elemento quedará anclado de forma relativa al elemento *parent* (siempre que el elemento padre tenga alguna posición definida, excepto "static").

## Propiedades de posición

Ahora que hemos definido nuestros métodos de posicionamiento al estilo que queremos usar, podemos arrancar a posicionar nuestro elemento. (Nota: esto funciona para cada método de posición que no sea "static", dado que no afecta al elemento de ninguna forma).

### *top*, *left*, *right* y *bottom*

Después de haber definido nuestro método de posicionamiento, podemos usar las propiedades "top", "left", "right" y "bottom" para acomodar nuestro elemento. El valor que le des a cada uno determinará qué tan lejos del borde quedará nuestro elemento. Por ejemplo, si queremos que nuestro elemento esté en la esquina superior izquierda (con una posición fija), podemos usar lo siguiente: 

```
div {  
  position: fixed;  
  top: 0;  
  left: 0;  
}
```

Si lo quisiésemos 10 píxeles debajo del límite superior y 10 píxeles del borde derecho:

```
div {  
  position: fixed;  
  top: 10px;  
  right: 10px;  
}
```

***Nota: existe otra forma mas fácil con flexbox***

# Introducción a Flexbox

Introducido en CSS3, Flexbox es una nueva e interesante característica. La misma nos permite posicionar nuestros elementos en relación a su *parent* y entre ellos. Ya no necesitamos aplicar "hacks" para cosas como centrar elementos. Esto nos permite que el diseño "mobile-friendly" sea excelente y nos hace dedicar menos tiempo tratando de posicionar elementos como corresponde. Flexbox se puede complicar muy rápido, pero veremos los aspectos básicos a continuación.

## *display: flex e inline-flex*

Como mencionamos anteriormente en la sección de la propiedad "display", uno de los valores puede ser "flex", esto hace que cualquier contenedor sea un "flex block". También podemos usar "inline-flex" para hacer que el elemento sea "flex" e "inline", aunque para la mayor parte del tiempo, usaremos simplemente "flex".

## *justify-content y align-items*

Ahora que nuestro contenedor (elemento) es "flex", podemos imaginarlo como una grilla con columnas que van de izquierdo a derecha y filas que van de arriba a abajo. Podemos usar las propiedades "justify-content" y "align-items" para decirle al contenedor dónde queremos que estén los elementos en la grilla. En principio, "justify-content" aplicará al movimiento de izquierda a derecha (fila), y "align-items" lo hará de arriba a abajo (columna). Tenemos unas reglas que debemos aplicar a cada una de estas reglas:

- `center` : centrará el elemento (o grupo de elementos) a lo largo de un eje en el que aplica esta regla.
- `flex-start` : Este es el valor por defecto de cada "flex box", mostrará todos los elementos en un grupo al comienzo de una fila o columna.
- `flex-end` : es lo opuesto a `flex-start` , mostrará los elementos al final de un grupo al comienzo de una fila o columna.
- `space-between` : Esta regla espaciará uniformemente el elemento o los elementos a lo largo de la fila o columna. El primer elemento estará como `flex-start` y el último como `flex-end` .
- `space-around` : Similar a `space-between` , pero aplicará márgenes igualitarios entre cada elemento, por lo que ningún elemento estará directamente sobre el borde del contenedor.

Ejemplo: si quisiésemos nuestros elementos centrados en el medio exacto de un "flex box", usaríamos lo siguiente:

```
div {  
  display: flex;  
  justify-content: center;  
  align-items: center;  
}
```

## *flex-direction*

Esta propiedad puede cambiar cómo el navegador interpreta `justify-content` (JC) y `align-items` (AI). El valor por defecto es "row" (fila), y esto funciona en la mayoría de los casos, pero algunas veces queremos cambiar cómo funciona la dirección del contenido.

- `row` : es la dirección por defecto. JC aplica de izquierda a derecha, AI aplica de arriba a abajo.
- `column` : Esto invertirá qué propiedad controla qué dirección. JC aplicará de arriba a abajo y AI de izquierda a derecha.
- `row-reverse` : Sólo invierte la dirección de JC de derecha a izquierda, no afecta a AI.
- `column-reverse` : Sólo invierte la dirección de AI de abajo a arriba, no afecta a JC.

## *align-self*

Por último, cubriremos una propiedad más avanzada llamada "align-self". La misma será aplicada a un elemento dentro del "flex box" del cual queremos separar el control de `align-items` . Si le damos la propiedad `align-self` , podemos colocarla en cualquier lugar a lo largo del eje de elementos de alineación que queramos. (Nota: NO existe `justify-self` , esta es la razón principal por la que los desarrolladores cambiarán la dirección de "flex").