

CSS - Cascade Style Sheet

¿Qué es CSS y para qué sirve?

CSS significa **Cascade Style Sheet** o hoja de estilo en cascada y nos permite crear reglas que definen propiedades visuales para nuestros elementos.

Ejemplo de código CSS:

```
span {  
  color: white;  
}
```

- En el código de ejemplo definimos que todos los elementos span de nuestro documento/s deben tener la tipografía de color blanco.
- De esta definición vemos que tenemos una forma de seleccionar elementos y que podemos aplicar una propiedad para modificar la forma en que se ve el contenido de la etiqueta seleccionada.

Ejemplo de código CSS:

```
selector {  
  nombrepropiedad: valor;  
}
```

- En CSS todo el tiempo vamos a estar utilizando estos dos conceptos:
 - Seleccionar elementos
 - Definir como queremos que se vean
- Según como utilizemos CSS podemos definir como se ve un elemento, un documento o todo nuestro sitio

IMPORTANTE

- Una herramienta muy útil a la hora de trabajar con CSS y JavaScript es la [Devtools de Chrome](#)

¿Cómo seleccionar elementos y definir el color de texto?

Color de texto

- Por medio de la propiedad **color** podemos establecer el color de tipografía que tiene un elemento
- Esta propiedad **color** acepta un color como valor, dentro de las opciones puede ser:
 - Nombre de color en ingles (white, red, blue, yellow, gray, etc)
 - Color en formato RGB (rgb(0,0,0)) donde cada valor va de 0 a 255 siendo 0 el apagado y 255 prendido
 - [Color Hexadecimal](#)
- Para saber más pueden visitar el [sitio de MDN](#)

Selector básico por tipo de elemento

- Utilizando el selector por tipo de elemento podemos agrupar la forma de aplicar estilos a los elementos según el nombre de su etiqueta
- Este es un selector general ya que va a seleccionar **TODOS** los elementos del mismo tipo

Ejemplo:

```
p {
  color: blue;
}

div {
  color: rgb(255, 0, 0);
}

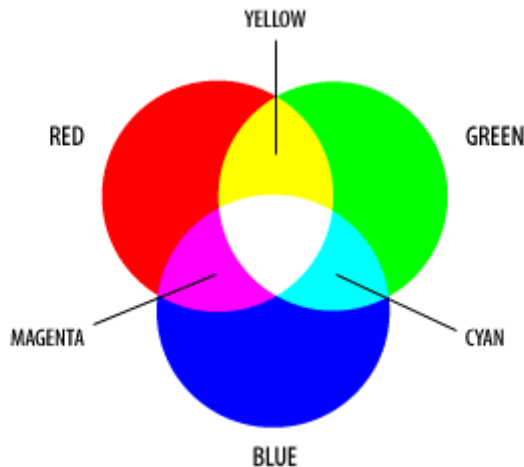
span {
  color: #00FF00;
}
```

<p>Texto en color azul</p>
 <p>Texto en color azul y verde</p>
 <div>Texto en color rojo</div>
 <div>Texto en color rojo y verde</div>
 Texto en color verde

Colores

- Los colores en las computadoras están compuestos por 3 colores básicos llamados RGB que vienen de **R**ed, **G**reen y **B**lue, es decir Rojo, Verde y Azul
- Los monitores estan compuestos de píxeles que son como pequeñas lucecitas que pueden mostrar estos colores
- Cuando la luz esta **apagada** obtenemos el color **negro**

- Cuando la luz está en su **máxima intensidad en todos los colores** obtenemos el color **blanco**
- Existen diferentes sistemas para especificar un color:
 - Colores por nombre
 - RGB
 - HEXA
 - HUE



Nombre de color

- Existen nombres de colores que podemos utilizar directamente y el browser sabe como mostrarlos
- La lista de colores que podemos utilizar no es muy extensa por lo cual nos limita a la hora de elegir un color
- El nombre del color es en inglés (ejemplo: white/blanco)
- Son fáciles de utilizar
- En cada nueva versión de CSS se agregan más colores
- Ejemplos de colores: **white, silver, gray, black, red, blue, green, orange, pink, brown and yellow**

Ejemplo:

```
body { color: black; }
p { color: gray; }
a { color: orange; }
```

[Lista de colores según versión de CSS](#)

RGB

- Para este color utilizamos la función de CSS llamada **rgb**
- Podemos asignar un valor entre 0 y 255 a cada color
- Si todos los valores son 0 obtenemos el color negro
- Si todos los valores son 255 obtenemos el color blanco
- El orden de los colores es el especificado en el nombre, es decir que el primer valor es para rojo, el segundo para verde y el tercero para azul
- Ejemplos:
 - rojo: `rgb(255, 0, 0)`
 - verde: `rgb(0, 255, 0)`
 - azul: `rgb(0, 0, 255)`

Ejemplo:

```
body {
  /* color negro */
  color: rgb(0, 0, 0);
}

p {
  /* color verde */
  color: rgb(0, 255, 0);
}
```

- Existe otra forma de utilizar rgb en CSS y es por medio de la función `rgba()` que agrega un canal más de transparencia
- Podemos asignar un valor entre 0 y 1 para el nivel de transparencia
- Se asigna como cuarto parámetro de la función

Ejemplo:

```
body {
  /* color rojo */
  color: rgba(255, 0, 0, 1);
}

p {
  /* color rojo más transparente */
  color: rgba(255, 0, 0, 0.4);
}
```

HEXA

- El sistema de HEXA representa los colores en Hexadecimal
- El color arranca con un numeral (#)
- Tenemos valores del 0 a la letra F
- Los números van del 0 al 9

- Continúan desde la letra A hasta la F
- Asignamos un par de letras para cada color del RGB
- Los dos primeros caracteres son para el rojo
- Los dos del medio son para el verde
- Los dos últimos son para el azul
- Si todos los valores son 0 tenemos el color negro (todo apagado)
- Si todos los valores son F tenemos el color blanco (todo prendido)
- Ejemplos:
 - blanco: #FFFFFF
 - negro: #000000
 - rojo: #FF0000
 - verde: #00FF00
 - azul: #0000FF

Ejemplo:

```
body {
  color: #FFFFFF;
}

p {
  color: #FF0000;
}
```

- En caso de que las dos letras del mismo color se repitan podemos utilizar una sola, por ejemplo para blanco puede ser #FFF o negro #000

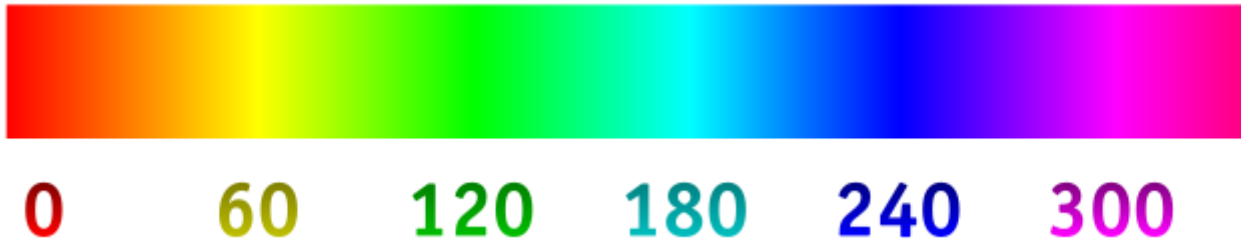
Ejemplo:

```
body {
  color: #FFF;
}
```

HSL

- Este sistema hace referencia a **H**ue, **S**aturation and **I**ntensity, es decir Matiz, Saturación e Intensidad
- Podemos establecer un valor entre 0 y 360 para el HUE según el color que

queremos



- Tanto saturación como intensidad aceptan un valor que va de 0 a 100%
- Ejemplo:
 - negro: hsl(0, 0%, 0%)
 - blanco: hsl(0, 0%, 100%)
 - rojo: hsl(0, 100%, 50%)

[Calculador de HSL](#) [Más sobre HUE](#) [Más sobre HSL](#) [Más sobre colores](#)

- En este sistema también contramos la función con alpha que funciona de la misma manera que el rgba.
- El cuarto valor acepta un número entre 0 y 1 para el nivel de transparencia que queremos
- Ejemplo: gris con transparencia: hsla(0, 100%, 100%, 0.5);}

¿Cómo agregar CSS?

- Existen diferentes formas de aplicar CSS
 - Se puede aplicar a nivel elemento utilizando el atributo **style** de HTML
 - En un documento por medio de la etiqueta **style** en el head
 - Compartido entre varios documentos utilizando la etiqueta **link** en el head de nuestros documentos que queremos vincular

Agregar hoja de estilo en un elemento (atributo style)

- La forma más particular e individual que tenemos de utilizar css es utilizando el atributo **style** que tienen los elementos HTML
- Dentro del atributo style escribimos las propiedades visuales separadas por punto y coma

Ejemplo:

```
<span style="color: white;">Texto en color blanco!!</span>
```

- De esta forma establecemos que **un** elemento span en particular va a tener el color de texto blanco
- Si quiero que **dos** o más elementos tengan el mismo color lo hacemos de la siguiente manera:

Ejemplo:

```
<span style="color: white;">Texto en color blanco!!</span>  
<span style="color: white;">Otro texto en color blanco!!</span>
```

- Genial, ya tenemos 2 span con texto en blanco, ahora me pregunto, si tengo 50 span en el documento, los tengo que modificar uno a uno?, La respuesta es **NO** y ya vamos a ver cómo lo hacemos
- Cuando definimos estilos a nivel elementos lo estamos haciendo de forma individual
- Esta es una buena opción cuando necesitamos que si o si un elemento se vea de una determinada forma
- Utilizando esta definición de css se hace más difícil mantener o modificar nuestros elementos ya que para cambiar el color de una tipografía (como en el ejemplo) vamos a tener que modificar uno a uno todos los elementos (mucho trabajo)
- Si tenemos tan solo un documento puede ser una tarea relativamente simple pero si tenemos 300 documentos se hace más complicado
- Para evitar este problema necesitamos otra alternativa que nos permita definir los atributos visuales de nuestros elementos a nivel documento

Práctica

[Ejercicio 1](#)

Agregar hoja de estilo en un documento (style)

- Por medio de la etiqueta **style** podemos definir los estilos que queremos para nuestros elementos a nivel documento
- Utilizando selectores podemos cambiar la forma que se ve uno o muchos elementos del mismo tipo
- Para definir que el texto que estamos escribiendo dentro de la etiqueta **style** es css, utilizamos el atributo **type** con el valor "text/css"
- Si bien podemos omitir este atributo ya que HTML5 no lo requiere dejamos en manos del browser como interpretar el contenido de esta etiqueta

- Para estar más seguro y lograr mejor compatibilidad con browsers anteriores definimos el atributo **type** del elemento **style**

Ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Usando CSS</title>
    <style type="text/css">
      span {
        color: white;
      }
    </style>
  </head>
  <body>
    <span>Texto en color Blanco</span>
    <span>Texto en color Blanco</span>
    <span>Texto en color Blanco</span>
    <span>Texto en color Blanco</span>
    <span>Texto en color Blanco</span>
  </body>
</html>
```

- Con tan solo un cambio podemos establecer que todos nuestros elementos span se vean de otro color

Ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Usando CSS</title>
    <style type="text/css">
      span {
        color: red;
      }
    </style>
  </head>
  <body>
    <span>Texto en color Rojo</span>
    <span>Texto en color Rojo</span>
    <span>Texto en color Rojo</span>
    <span>Texto en color Rojo</span>
    <span>Texto en color Rojo</span>
  </body>
</html>
```

Práctica

[Ejercicio 2](#)

- Esta forma de utilizar los estilos está buena cuando necesitamos definir estilos para un documento

- Si bien esta forma es más general que definir los estilos en cada elemento, sigue siendo una individual a nivel documento
- Un sitio va a tener más de un documento y si queremos mantenerlo necesitamos una forma de poder unificar todos los estilos para todos nuestros documentos

Agregar hoja de estilo en un documento externo (link)

- Utilizando la etiqueta **link** podemos vincular un documento HTML con uno de CSS
- Todas las reglas que definamos en el documento CSS van a ser aplicadas en todos los documentos vinculados
- Esta es la mejor forma de unificar los estilos para nuestro sitio y es una forma más general de hacerlo
- La etiqueta **link** tiene los siguientes atributos y valores:
 - **href**: establece el path al documento CSS
 - **type**: al igual que en la definición de la etiqueta **style** especificamos que el contenido de este documento vinculado es "text/css"
 - **rel**: establece la relación con el otro documento y utilizamos el valor "stylesheet"

Ejemplo:

Archivo: styles.css

```
span {  
  color: white  
}
```

Archivo: index.html

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Index</title>  
    <link href="styles.css" type="text/css" rel="stylesheet" />  
  </head>  
  <body>  
    <span>Texto en color Blanco</span>  
    <span>Texto en color Blanco</span>  
    <span>Texto en color Blanco</span>  
    <span>Texto en color Blanco</span>  
    <span>Texto en color Blanco</span>  
  </body>  
</html>
```

Archivo: contact.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Contact</title>
    <link href="styles.css" type="text/css" rel="stylesheet" />
  </head>
  <body>
    <span>Texto en color Blanco</span>
    <span>Texto en color Blanco</span>
    <span>Texto en color Blanco</span>
    <span>Texto en color Blanco</span>
    <span>Texto en color Blanco</span>
  </body>
</html>
```

- Con tan solo un cambio en la hoja de estilo podemos modificar todos los documentos vinculados

Ejemplo:

Archivo: styles.css

```
span {
  color: red
}
```

Archivo: index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Index</title>
    <link href="styles.css" type="text/css" rel="stylesheet" />
  </head>
  <body>
    <span>Texto en color Rojo</span>
    <span>Texto en color Rojo</span>
    <span>Texto en color Rojo</span>
    <span>Texto en color Rojo</span>
    <span>Texto en color Rojo</span>
  </body>
</html>
```

Archivo: contact.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Contact</title>
    <link href="styles.css" type="text/css" rel="stylesheet" />
  </head>
  <body>
    <span>Texto en color Rojo</span>
    <span>Texto en color Rojo</span>
    <span>Texto en color Rojo</span>
    <span>Texto en color Rojo</span>
    <span>Texto en color Rojo</span>
  </body>
```

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

Práctica

Ejercicio 3

- CSS se llama Hoja de estilo en cascada ya que los elementos pueden **heredar** propiedades visuales de padre a hijo
- Si establecemos estilos al **body** todos los elementos que esten en el documento van a heredar esos atributos
- No todos los elementos son heredables
- Una buena forma de ver esto es utilizando la barra de [developer tools](#)
- Los estilos que definimos en un archivo externos pueden ser sobrescritos utilizando la etiqueta **styles**
- Los estilos definidos en style pueden ser sobrescritos utilizando el atributo **style** de los elementos
- De esta forma podemos ver que utilizamos conceptos generales y son pisados por los más individuales
- Este mismo concepto se da entre elementos
- Si definimos atributos visuales para el **body** podemos sobrescribir como se ven los **párrafos** por ejemplo dejando que el resto de los elementos hereden las propiedades declaradas en el **body**

Ejemplo:

styles.css

```
p {
  color: red;
}

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

<style type="text/css">
  p {
    color: blue;
  }
</style>

<p style="color: black;">Texto en color negro!!</p>
```

- Como podemos ver en este ejemplo el estilo que establecemos en el elemento va a ser el que quede definido finalmente
- El estilo definido en el documento (etiqueta style) pisa el que esta definido en el archivo general. En este caso importa el orden en que fueron llamados los estilos

- Como regla podemos decir que siempre ponemos los estilos más generales primero y después sobrescribimos lo que necesitamos

Práctica

[Ejercicio 4](#)

Selectores

- Un selector es la forma en la que elegimos a que elementos queremos asignarle atributos visuales con CSS
- Los selectores pueden ser generales o particulares, es decir que puedo seleccionar varios elementos o tan solo uno y de forma muy particular
- Podemos utilizar un solo selector de la siguiente:

Ejemplo:

```
selector {  
  // código CSS  
  // código CSS  
  // código CSS  
}
```

- Podemos utilizar más de un selector separados por coma:

Ejemplo:

```
selector1, selector2 {  
  // código CSS  
  // código CSS  
  // código CSS  
}
```

Tipo de etiqueta

- Es el selector que venimos utilizando en todos los ejemplos anteriores
- El selector por **tipo de etiqueta** nos permite elegir elementos por el **nombre de la etiqueta**
- Este tipo de selector retorna una colección de elementos ya que selecciona **todos** los elementos con el mismo nombre de etiqueta
- Por ejemplo podemos seleccionar los elementos p y h1 y definir que el color de texto sea blanco

Ejemplo:

```
p {
```

```
    color: white;
}

h1 {
    color: white;
}
```

- Dado que ambos elementos se van a ver de la misma forma podemos escribir los selectores de la siguiente forma:

Ejemplo:

```
p, h1 {
    color: white;
}
```

Práctica

[Ejercicio 5](#)

Selector universal

- Por medio del selector asterisco (*) podemos seleccionar todos los elementos
- Este selector se utiliza muchas veces para borrar estilos que vienen predefinidos en el browser

Ejemplo:

```
* {
    color: white;
}
```

Selector por clase

- Los elementos HTML tiene un atributo **class** que nos permite asignarles un nombre de clase
- En CSS podemos seleccionar los elementos por nombre de clase utilizando un punto y el nombre de la clase
- Este selector afecta a todos los elementos que tengan el nombre de clase seleccionado

Ejemplo:

En CSS:

```
.rojo {
    color: red;
}
```

```
}
```

En HTML:

```
<p class="rojo">Este texto es ROJO</p>
```

- Podemos hacer un selector más particular si queremos seleccionar sólo algunos elementos que tengan una determinada clase
- Si queremos seleccionar sólo los títulos H1 que tienen una clase rojo podemos hacerlo de la siguiente forma:

Ejemplo:

En CSS:

```
h1.rojo {  
  color: red;  
}
```

En HTML:

```
<h1 class="rojo">Este texto es ROJO</h1>  
<p class="rojo">Este texto es NEGRO</p>
```

- Este último selector sólo selecciona los elementos del tipo h1 que tienen la clase rojo

Selector por ID

- Los elementos HTML tienen un atributo ID que nos permite seleccionarlos de forma única
- Con CSS podemos seleccionar los elementos por ID utilizando el símbolo de numeral (#) y el nombre del ID del elemento
- Este es un selector individual ya que solo modifica un solo elemento
- Si queremos seleccionar el elemento que tiene el ID principal y establecer el texto en azul lo podemos hacer de la siguiente manera:

Ejemplo:

En CSS:

```
#principal {  
  color: blue;  
}
```

En HTML:

```
<div id="principal">Contenido de mi div en AZUL</div>
```

Práctica

[Ejercicio 6](#)

Hijos

- Podemos seleccionar los elementos que sólo son hijos directos de un elemento
- Utilizamos el símbolo mayor que (>) para elegir a los hijos de un elemento

Ejemplo:

```
div > p {  
  color: red;  
}  
  
<p>Texto de color negro</p>  
<div>  
  <p>Texto de color ROJO</p>  
  <p>Texto de color ROJO</p>  
  <h1>Texto de color negro</h1>  
</div>
```

- En el ejemplo anterior vemos como podemos seleccionar sólo los párrafos que son hijos de un elemento div

Práctica

[Ejercicio 7](#)

Selector descendiente

- El selector descendiente permite seleccionar cualquier elemento

Práctica

[Ejercicio 8](#)

- Ahora que sabemos varios selectores podemos ver más propiedades de CSS
- Existen más selectores y los vamos a seguir viendo más adelante

Tipografías

Familias

- Por medio de la propiedad **font-family** se puede establecer una lista de familias tipográficas
- Dado que todas las máquinas pueden no tener todas las tipografías instaladas podemos seleccionar más de una
- Las tipografías las separamos con comas. Ejemplo: Arial, serif
- El browser va a elegir cada tipografía de izquierda a derecha hasta encontrar una que tenga
- Los nombres de las tipografías se escriben con comillas dobles en caso de que el nombre contenga espacios, ejemplo: "Times New Roman", Georgia, serif
- Se recomienda poner una tipografía por default (serif, sans-serif, monospace, cursive o fantasy) para lograr al menos el estilo deseado
- También se pueden utilizar web fonts como pueden ser las de [Google](#)
- Para saber más sobre la propiedad **font-family** pueden ver el [MDN](#)

Ejemplos de tipografías por defecto

A Serif headline

Serif paragraph text.

A Sans-Serif headline

Sans-Serif paragraph text.

A Monospace headline

Monospace paragraph text.

A Cursive headline

Cursive paragraph text.

A Fantasy headline

Fantasy paragraph text.

Ejemplo:

```
body {  
  font-family: Arial, sans-serif;  
  color: black;  
}  
  
h1 {  
  font-family: "Times New Roman", serif;  
  color: gray;  
}
```



```
}
```

- En este ejemplo definimos:
 - El texto del documento tiene que ser negro y con tipografía Arial o sans-serif
 - Los títulos h1 son grises y con tipografía Times New Roman o serif

Práctica

[Ejercicio 9](#)

Tamaño

- Por medio de la propiedad **font-size** podemos establecer el tamaño de la tipografía
- Acepta como valor:
 - **Píxeles:** establecemos el número de píxeles y la palabra **px**
 - Es un valor fijo
 - Es ideal para pantallas de computadoras
 - No escalan bien para dispositivos mobile
 - No se pueden agrandar en browsers viejos
 - **Porcentaje:** es un % del valor establecido
 - Es relativo
 - Escala bien
 - Es fácil para cambiar la relación de todos los tamaños ya que definimos el valor inicial y después lo cambiamos en un solo lugar (el resto es todo relativo)
 - **EMS:** este valor representa una unidad del tamaño establecido
 - Si establecemos el tamaño a 10px, 1em representa 10px y 2em 20px
 - Si no se establece un valor toma el por default del browser (16px en general)
 - Se dice que toma el ancho y alto de la letra M
 - Es una unidad relativa al padre
 - Escala bien para mobile
 - Funciona en cascada
 - Un truco es establecer el tamaño del sitio en 62.5% tomando que el browser tiene 16px nos queda una tipografía inicial de 10px

- Luego podemos utilizar valores como 1.8em para lograr una tipografía de 18px
- **Points:** es una unidad para imprimir
 - 27 puntos representan una pulgada es decir 2,54 cm
 - Son precisos a la hora de imprimir
- Para aprender más sobre el tema podemos seguir leyendo este buen artículo de alistapart.com [\(en inglés y muy buen sitio\)](http://alistapart.com)
- Una buena sección para mirar del mismo sitio: [Typography Web](http://typographyweb.com)
[Fonts](http://typographyweb.com)

px	em	percent
5px	0.3125em	31.25%
6px	0.3750em	37.50%
7px	0.4375em	43.75%
8px	0.5000em	50.00%
9px	0.5625em	56.25%
10px	0.6250em	62.50%
11px	0.6875em	68.75%
12px	0.7500em	75.00%
13px	0.8125em	81.25%
14px	0.8750em	87.50%
15px	0.9375em	93.75%
16px	1.0000em	100.00%
17px	1.0625em	106.25%
18px	1.1250em	112.50%
19px	1.1875em	118.75%
20px	1.2500em	125.00%
21px	1.3125em	131.25%
22px	1.3750em	137.50%
23px	1.4375em	143.75%
24px	1.5000em	150.00%
25px	1.5625em	156.25%

Ejemplo:

```
body { font-size: 100%; }  
  
p { font-size: 12px; }  
  
a { font-size: 1em; }
```

Práctica

[Ejercicio 10](#)

Bold

- Por medio de la propiedad **font-weight** podemos establecer si el peso de nuestra tipografía
- Podemos establecer si es **normal** o **bold**
- También acepta valores numéricos entre 100 y 900
- Otra opción es utilizar valores relativos al padre por medio de **lighter** (más liviano) o **bolder** (más pesado)
- Estas últimas opciones no son soportadas por todos los browsers
- Podemos aprender más sobre esta propiedad en el [sitio de MDN](#)

Ejemplo:

```
.negrita { font-weight: bold; }  
  
h1 { font-weight: 900; }  
  
<p>Texto en <span class="negrita">NEGRITA</span></p>  
<h1>Texto con negrita en 900 de peso</h1>
```

Práctica

[Ejercicio 11](#)

Itálica

- Por medio de la propiedad font-style podemos establecer si queremos que nuestra tipografía sea normal, itálica o oblicua
- Los valores que acepta esta propiedad están en inglés:
 - normal
 - italic
 - oblique
- Podemos aprender más sobre esta propiedad en el [sitio de MDN](#)

Ejemplo:

```
.italica { font-style: italic; }  
.oblicuo { font-style: oblique; }  
<p>Texto en <span class="italica">Italica</span></p>  
<h1>Texto en <span class="oblicuo">Oblicuo</span></h1>
```

Transformación de texto

- Por medio de la propiedad **text-transform** podemos transformar las mayúsculas y minúsculas de nuestros textos
- A diferencia de lo que venimos viendo en este caso se utiliza text en lugar de font ya que hace referencia al texto y no a la tipografía
- Acepta los siguientes valores: **uppercase**: pasa todo el texto a mayúscula **lowercase**: pasa todo el texto a minúscula **capitalize**: convierte la primer letra de cada palabra a mayúscula
- Podemos aprender más sobre esta propiedad en el [sitio de MDN](#)

Ejemplo:

```
h1 { text-transform: uppercase; }  
p { text-transform: lowercase; }  
<h1>texto todo en mayúscula</h1>  
<p>TEXTO TODO EN MINUSCULA</p>
```

Práctica

[Ejercicio 12](#)

Alineado de textos

- Por medio de la propiedad text-align podemos establecer como queremos alinear los textos de forma horizontal
- Acepta los siguientes valores:
 - **left**: alinea todo a la izquierda
 - **right**: alinea todo a la derecha
 - **center**: alinea todo al centro
 - **justify**: hace que el texto ocupe toda la caja (justificado)
- Podemos aprender más sobre esta propiedad en el [sitio de MDN](#)

Práctica

[Ejercicio 13](#)

Decorado

- Por medio de la propiedad `text-decoration` establecemos si queremos decorar nuestra tipografía según los valores posibles:
 - **none:** saca todo tipo de decorado (es útil para hacer que los hipervínculos no tengan una raya debajo del texto)
 - **underline:** nos muestra el texto subrayado
 - **overline:** agrega una línea sobre el texto
 - **line-through:** el texto aparece tachado con una línea en el medio (es útil a la hora de tachar elementos de una lista)
 - **blink:** Volvemos a los 90's y nuestros textos parpadean
- Podemos aprender más sobre esta propiedad en el [sitio de MDN](#)

```
a { text-decoration: none; }  
h1 { text-decoration: underline; }
```

- En browsers más nuevo se puede establecer el color y tipo de línea que utilizamos
- Esta opción todavía [no es soportada por todos los browsers](#)
 - Primer parámetro el tipo de decorado
 - Segundo parámetro el tipo de línea (más sobre esto cuando vemos bordes)
 - Finalmente establecemos el color de la línea

```
h1 {  
  text-decoration: underline wavy red;  
}
```

Práctica

[Ejercicio 14](#)

Indentación

- Podemos indentar nuestro texto utilizando la propiedad **text-indent**
- Esta propiedad nos permite establecer un valor que va a funcionar como margen izquierdo de la primer palabra de la primer línea.
- Acepta valores negativos
- Podemos aprender más sobre esta propiedad en el [sitio de MDN](#)

Ejemplo:

```
p { text-indent: 20px; }
```

Práctica

[Ejercicio 15](#)

Sombra

- Para establecer la sombra de nuestro texto utilizamos la propiedad **text-shadow**
- Esta propiedad utiliza varios valores:
 - El primer parámetro maneja el eje x de nuestra sombra, es decir que va de izquierda a derecha
 - El segundo parámetro maneja el eje, es decir que va de arriba a abajo
 - El tercer parámetro maneja el blur (difuminar) de la sombra
 - El cuarto parámetro es el color de la sombra
- Podemos aprender más sobre esta propiedad en el [sitio de MDN](#)
- Los browsers actuales [soportan bien esta propiedad](#)

Ejemplo:

```
p { text-shadow: 4px 2px 0px #000000; }
```

- Para jugar con estas opciones podemos usar el [siguiente sitio](#)

Práctica

[Ejercicio 16](#)

Espaciado o aire

- Podemos establecer la distancia tanto entre letras como palabras
- Para manejar la distancia entre letras utilizamos **letter-spacing**
- Para manejar la distancia entre palabras utilizamos **word-spacing**
- Podemos aprender más sobre esta propiedad en el sitio de MDN:
 - [letter-spacing](#)
 - [word-spacing](#)

```
body {  
  letter-spacing: 1px;  
}  
  
p {
```

```
word-spacing: 2px;
}
```

Alto de línea

- Podemos establecer el alto de línea de un texto por medio de la propiedad **line-height**
- Acepta un valor numérico
- Podemos aprender más sobre esta propiedad en el [sitio de MDN](#)

Ejemplo:

```
p {
  font-size: 12px;
  line-height: 18px;
}
```

Práctica

[Ejercicio 17](#)

Alineado Vertical

- Podemos alinear los textos de forma vertical utilizando la propiedad **vertical-align**
- Acepta los siguientes valores:
 - baseline
 - sub
 - super
 - top
 - text-top
 - middle
 - bottom
 - text-bottom
- Algo común es creer que esta propiedad nos permite alinear los textos de forma vertical en los elementos en bloque pero no funciona de esa forma
- Esta propiedad nos permite alinear el contenido de los elementos en línea
- Como sabemos las imágenes son elementos en línea al igual que los spans, hipervínculos, etc

- Si dentro de un párrafo tenemos una imagen, texto y otros elementos en línea, podemos establecer como va a ser la relación entre todos estos elementos que son hijos del párrafo
- De esta forma podemos hacer que un texto este bien alineado a una foto
- Un buen sitio para leer sobre este tema es [css tricks \(en inglés\)](#)

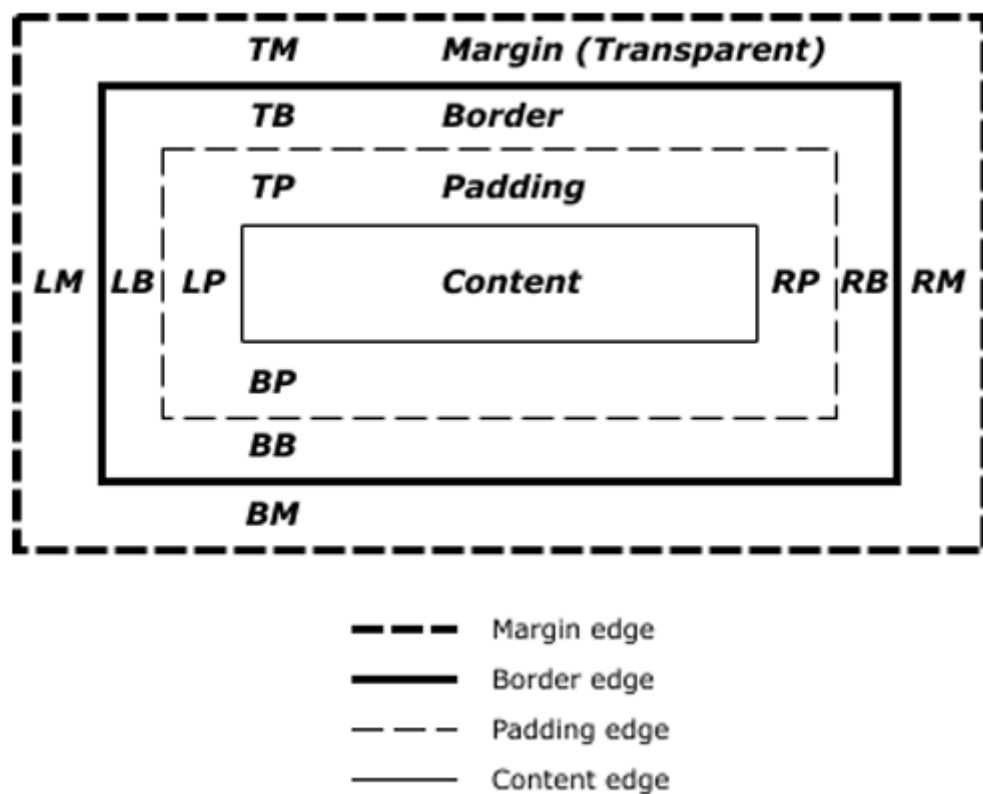
```
img { vertical-align: bottom; }
```

Práctica

[Ejercicio 18](#)

Modelo de Caja

- Podemos pensar los elementos de HTML como una caja
- Este concepto se conoce como box model



- **content:** es el contenido del elemento, por ejemplo un texto u otros elementos

- **padding:** es un margen interno entre el borde y el contenido, podemos establecer diferentes valores para la parte superior, inferior, izquierda y derecha
- **border:** es el borde de la caja y se le pueden aplicar diferentes estilo
- **margin:** es el margen externo de la caja, podemos establecer diferentes valores para la parte superior, inferior, izquierda y derecha
- En HTML existen elementos del tipo en bloque (block) y en línea (inline)
- Estos tipos de elementos se diferencian en la forma que se comportan y como se aplican los atributos de la caja
- Los elementos en línea por ejemplo ignoran los márgenes superior e inferior pero si aplica los de izquierda y derecha
- También los elementos en línea ignoran las propiedades de ancho y alto (lo vemos más adelante)

IMPORTANTE

- Para calcular el ancho de un elemento tenemos que tener en cuenta:
 - Ancho del elemento
 - Espacio asignado para padding (margen interno)
 - Ancho del borde
 - Espacio asignado a los márgenes
- Sumando todos estos valores obtenemos el ancho real de los elementos
- Algunos browsers viejos manejaban el modelo de la caja de otra forma y esto traía incompatibilidad entre ellos

Color de fondo

- La propiedad **background-color** nos permite establecer un color de fondo
- Podemos utilizar esta propiedad para nuestros diseños y también resaltar la caja de un elemento
- Acepta un color como valor (cualquiera de los que vimos)

Ejemplo:

```
div {  
  background-color: #eee;  
}
```

- En este ejemplo establecemos que todos los divs tengan un color de fondo gris claro
- También existe el color **transparent** que es el valor por default

Ejemplo:

```
div {  
  background-color: transparent;  
}
```

Práctica

[Ejercicio 19](#)

Ancho y Alto

- Por medio de las propiedades **width** y **height** podemos establecer el ancho y alto de la caja
- Acepta medidas en px, % o ems

Ejemplo:

```
div {  
  width: 200px;  
  height: 200px;  
}
```

- En el ejemplo anterior establecemos el alto y el ancho de la caja en 200px es decir que estamos haciendo un cuadrado
- También podemos establecer valores mínimos o máximos para un elemento
- Utilizamos min-width y max-width para establecer el mínimo y máximo ancho de la caja
- Podemos hacer lo mismo con el alto utilizando min-height y max-height para establecer el mínimo y máximo alto de la caja

Ejemplo:

```
div {  
  min-height: 100px;  
  min-width: 100px;  
}
```

- En el ejemplo anterior establecemos que mínimo alto y ancho de una caja
- En caso de necesitar ser más grande la caja se va a agrandar pero nunca va a ser menor a 100px

Práctica

[Ejercicio 20](#)

Contenido Extra

- Existe una propiedad llamada **overflow** que nos permite manejar como se debe comportar el contenido extra de nuestro sitio
- Muchas veces el browser no sabe como cortar un texto y simplemente se ve mal
- Esta propiedad acepta los valores **hidden** y **scroll**
- Con hidden ocultamos el contenido extra
- Con scroll permitimos que el usuario pueda scrollear el contenido

Ejemplo:

```
div {  
  overflow: scroll;  
}
```

Práctica

[Ejercicio 21](#)

Bordes:

- Por medio de la propiedad border podemos establecer el borde de la caja
- Acepta como valores:
 - Primero el ancho: utiliza medidas en pixeles o cualquiera de las otras medidas vistas
 - Segundo parámetro: es el estilo de borde que queremos
 - Tercer parámetro: es el color del borde

Ejemeplo:

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

- Cada uno de estos valores se pueden establecer de forma individual:
- Para el ancho utilizamos border-width
- Para el diseño utilizamos border-style
- PArá el color utilizamos border-color

Ejemplo:

```
div {  
  border-width: 1px;  
  border-style: solid;  
  border-color: red;  
}
```

- Es lo mismo si utilizo la propiedad borde o la sumatoria de las otras propiedades (border-width, style y color)
- Para los estilos de los bordes existen los siguientes valores:
 - solid
 - dotted
 - dashed
 - double
 - groove
 - ridge
 - inset
 - outset
- También podemos establecer los bordes de manera individual (top right bottom left)
- Por ejemplo para establecer los anchos del borde podemos utilizar las siguientes propiedades:
 - border-top-width
 - border-right-width
 - border-bottom-width
 - border-left-width
- también podemos establecer valores individuales utilizando una sola propiedad con muchos valores:
- border-width: top right bottom left;

Ejemplo:

```
div {  
  border-width: 2px 1px 1px 2px;  
}
```

- En este caso estamos estableciendo:
 - 2px para el borde superior (top)

- 1px para el borde derecho (right)
 - 1px para el borde inferior (bottom)
 - 2px para el borde de la izquierda (left)
- Si queremos el mismo valor para el borde superior e inferior y derecha e izquierda podemos utilizar solo dos valores:
- border-width: top/bottom right/left;
- border-width: 2px 1px;
- Es decir que establecemos 2px para el borde superior e inferior y 1px para el borde derecho e izquierdo
- El border style funciona de la misma forma:
 - border-top-style
 - border-left-style
 - border-right-style
 - border-bottom-style
- También podemos establecer 4 valores en el mismo orden que los anteriores utilizando la propiedad border-style
- También podemos establecer un color de borde para cada uno de ellos:
 - border-top-color
 - border-right-color
 - border-bottom-color
 - border-left-color
- También podemos establecer todos los valores del borde utilizando la propiedad **border**
- Los valores que espera son: border-width border-style border-color

Ejemplo:

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

Práctica

[Ejercicio 22](#)

Aire interno de la caja

- Por medio de la propiedad **padding** podemos establecer un margen interno de la caja
- Este concepto establece un aire interno a la caja
- Acepta distintos tipos de medida
- Al utilizar la propiedad **padding** con un sólo valor estamos estableciendo todos los lados iguales (top, right, bottom y left)

Ejemplo:

```
div {  
  padding: 20px  
}
```

- También lo podemos establecer para cada uno de los lados de forma individual:
 - padding-top
 - padding-bottom
 - padding-left
 - padding-right

Ejemplo:

```
div {  
  padding-top: 20px;  
  padding-bottom: 25px;  
  padding-left: 30px;  
  padding-right: 40px;  
}
```

- Otra opción para hacer esto es utilizar 4 o 2 valores al igual que lo hizimos con los bordes
 - padding: top/bottom right/left;
 - padding: 10px 20px;
 - padding: top right bottom left;
 - padding: 10px 15px 12px 25px;

Ejemplo:

```
div {  
  padding: 20px 40px 25px 30px;  
}
```

Ejemplo:

```
div {
```

```
padding: 20px 40px;
}
```

Práctica

[Ejercicio 23](#)

Márgenes

- Por medio de la propiedad **margin** podemos establecer el margen de la caja
- Esta propiedad soporta cualquier unidad de medida (px, em, %)
- El margen es exterior al los elementos y comienza luego del border
- Nos permite distanciar elementos entre sí
- Si establecemos un color de fondo y un margen no se va a ver más color de fondo como con el padding ya que el margen es externo
- Los valores funcionan igual que en el padding
 - Un solo valor establece todos los márgenes iguales
 - Podemos establecer 2 valores en caso de que queremos el mismo valor para top/bottom y left/right
 - Podemos establecer 4 valores en caso de querer utilizar la propiedad **margin** para establecer valores individuales para cada margen
 - Esta forma de escribir los márgenes también va en sentido de las agujas del reloj: top right bottom left

Ejemplo:

```
div {
  margin: 20px;
}
```

Ejemplo:

```
div {
  margin: 20px 30px;
}
```

Ejemplo:

```
div {
  margin: 20px 30px 35px 40px;
}
```

- También podemos establecer los márgenes de forma individual solamente utilizando las siguientes propiedades:
 - margin-top

- margin-bottom
- margin-left
- margin-right

Ejemplo:

```
div {  
  margin-top: 20px;  
  margin-right: 30px;  
  margin-bottom: 35px;  
  margin-left: 40px;  
}
```

Práctica

[Ejercicio 24](#)

- Podemos establecer el valor **auto** para que el browser establezca el valor de forma automática
- Utilizando este valor para los márgenes izquierda y derecha más establecer un valor de ancho de caja podemos centrar un elemento

Ejemplo:

```
div {  
  width: 500px;  
  margin: 0 auto;  
}
```

- En el ejemplo anterior establecemos lo siguiente:
 - El margen top/bottom en 0 establece que esa caja no tiene márgenes superior e inferior
 - El margen left/right en **auto** establece que esa caja tiene que tener estos márgenes en automático
 - El ancho de la caja es de 500px
 - Al hacer toda esta combinación de propiedades logramos centrar de forma horizontal
 - Esta técnica funciona bien con los elementos en bloque
- Recordemos que para centrar un elemento inline podemos establecer la propiedad **text-align:center** al padre del elemento que queremos centrar de forma horizontal

Práctica

[Ejercicio 25](#)

Sombra

- Utilizando la propiedad **box-shadow** establecemos una sombra a la caja
- funciona de la misma forma que la sombra del texto pero sobre la caja
- Acepta los siguientes valores: offset-x offset-y blur-radius spread-radius color
- Esta propiedad acepta valores negativos

Ejemplo:

```
div {  
  box-shadow: 1px 2px 5px -3px #aaa;  
}
```

- En el ejemplo anterior establecemos los siguientes valores de sombra de la caja:
 - 1 px de offset-x
 - 2 px de offset-y
 - 5 px de blur-radius
 - -3px spread-radius
 - Color de sombra #aaa (un gris oscuro)
- Para ver como funciona esta propiedad podemos utilizar la [siguiente herramienta](#)

Práctica

[Ejercicio 26](#)

- También podemos establecer que la sombra es interna a la caja si utilizamos el valor **inset** como primer valor
- En caso de no establecer este valor el browser asimila que la sombra es externa a la caja
- El resto de los valores sigue funcionando igual sólo que se desplazan un lugar

Ejemplo:

```
div {  
  box-shadow: inset 1px 2px 5px -3px #aaa;  
}
```

Práctica

[Ejercicio 27](#)

Cambiar la forma en la que se comporta un elemento

- Utilizando la propiedad **display** podemos cambiar la forma en que se comportan los elementos
- Podemos cambiar un elemento en bloque para que se comporte como uno en línea y viceversa
- Esta propiedad soporta los siguientes valores:
 - **inline:** Establece que este elemento es en línea
 - **block:** Establece que este elemento es en bloque
 - **inline-block:** Establece que este elemento se comporte como en línea pero con propiedades de caja como si fuera en bloque
 - **none:** Este elemento no se ve pero tampoco pertenece al documento, es como si no existiera. Si vemos el código fuente vamos a ver que el elemento existe sólo que el browser lo ignora
 - **hidden:** Oculta un elemento pero sigue siendo parte del documento
 - **visible:** Establecemos que este elemento tiene que ser visible

Práctica

[Ejercicio 28](#)

Borders redondeados

- Por medio de la propiedad **border-radius** podemos establecer bordes redondeados
- Acepta valores numéricos
- NO acepta valores negativos
- Esta propiedad acepta dos valores:
 - **Largo**
 - **Percentage**
- Para probar distintos valores y ver como queda podemos utilizar el [siguiente generador](#)
- Para entender más sobre cómo funciona esta propiedad pueden consultar el [sitio de MDN](#)

Ejemplo:

```
div {
```

```
border: 1px solid red;
border-radius: 40px 50px;
}
```

- También podemos establecer un valor individual para cada uno de las esquinas:
 - border-top-right-radius
 - border-top-left-radius
 - border-bottom-right-radius
 - border-bottom-left-radius

Ejemplo:

```
div {
  border: 1px solid red;
  border-top-right-radius: 40px 50px;
  border-top-left-radius: 50px 30px;
}
```

Práctica

[Ejercicio 29](#)

Imagen de fondo

- Existen algunas propiedades que nos permiten utilizar imagenes como fondo de la caja
- Por medio de la propiedad **background-image** podemos establecer una o varias imagenes de fondo
- Acepta como valor la ruta de una imagen
- Para poder establecer el valor de la ruta de la imagen utilizamos la función **url()**
- Las rutas de las imagenes pueden ser relativas o absoultas
- Hasta ahora vimos como establecer el color de fondo utilizando la propiedad **background-color**
- Para saber más sobre esta propiedad podemos entrar en el [sitio de MDN](#)

Ejemplo:

```
div {
  background-image: url(/path/a/la/imagen.png);
}
```

- Para establecer más de una imagen establecemos valores separados por coma

**Ejemplo:*

```
div {  
  background-image: url(/path/a/la/imagen1.png), url(/path/a/la/imagen2.png);  
}
```

Práctica

[Ejercicio 30](#)

Repetir el fondo

- Por defecto la imagen que establecemos como fondo se va a repetir tantas veces como pueda
- Para controlar la forma en que se repite la imagen de fondo utilizamos la propiedad **background-repeat**
- Esta propiedad acepta los siguientes valores:
 - repeat: es el valor por default y repite la imagen todo lo que puede tanto en el eje x como en el y
 - repeat-x: repite la imagen sólo en el eje x
 - repeat-y: repite la imagen sólo en el eje y
 - no-repeat: no repite la imagen, es decir que se muestra tan solo una vez
- Para saber más sobre esta propiedad podemos entrar en el [sitio de MDN](#)

Ejemplo:

```
div {  
  background-image: url(/path/a/la/imagen.png);  
  background-repeat: no-repeat;  
}
```

Práctica

[Ejercicio 31](#)

- Es importante recordar que trabajar con imagenes muchas veces es más fácil si lo hacemos en la medida que necesitamos
- Dado que el browser tiene que descargar cada foto de manera individual también conviene tener estas imagenes optimizadas para que pesen menos

Posicionar el fondo

- Por defecto la imagen de fondo se posiciona en el borde izquierdo superior del fondo
- Podemos cambiar la posición de la imagen de fondo utilizamos la propiedad **background-position**
- Esta propiedad acepta los siguientes valores:
 - El primer valor especifica el eje x y tiene los siguientes valores:
 - Unidad de medida (puede ser %, px, em)
 - left
 - center
 - right
 - El segundo valor especifica el eje y y tiene los siguientes valores:
 - Unidad de medida (puede ser %, px, em)
 - top
 - center
 - bottom

Ejemplo:

```
/* Posicionamos la imagen a 10px a la derecha y 20px hacia abajo de la
esquina izquierda superior */
div {
  background-position: 10px 20px;
}

/* Posiciona la imagen al centro (entre izquierda y derecha) y en el borde
inferior de la caja */
section {
  background-position: center bottom;
}
```

- Podemos ver como cambiar distintos valores para posicionar las imagenes de fondo:
 - left top
 - left center
 - left bottom
 - center top
 - center center
 - center bottom
 - right top
 - right center
 - right bottom

Práctica

[Ejercicio 32](#)

Fijar el fondo:

- Por medio de la propiedad **background-attachment** podemos establecer si queremos fijar o esrolear el fondo según el valor seleccionado:
 - fixed: la imagen se queda fija y el resto del documento parece como si flota sobre la imagen de fondo, esto pasa ya que al establecer esta propiedad en scroll la imagen de fondo se scrollea con el resto del documento.
 - scroll: la imagen sube y baja según el usuario scrolea

Ejemplo:

```
div {  
  background-image: url(foto.png);  
  background-attachment: fixed;  
}  
  
div.scroll {  
  background-image: url(foto.png);  
  background-attachment: scroll;  
}
```

Práctica

[Ejercicio 33](#)

Establecer todos los valores juntos:

- Por medio de la propiedad **background** podemos establecer todos los valores en una sola propiedad respetando el siguiente orden:
 - background-color
 - background-image
 - background-repeat
 - background-attachment
 - background-position

Ejemplo:

```
div {  
  background: red url(foto.png) no-repeat fixed center center  
}
```

Práctica

[Ejercicio 34](#)

Color de fondo con gradiente

- Para establecer un degradado utilizamos la propiedad **linear-gradient**
- Esta propiedad acepta los siguientes valores:
 - Angulo: Establece el angulo que queremos utilizar, ejemplo: 60deg
 - Primer color: Establece cual va a ser el primer color, ejemplo: blue,
 - Segundo color: Establece cual va a ser el segundo color, ejemplo: green

Ejemplo:

```
div {  
  background: linear-gradient( 60deg, blue, green );  
}
```

Práctica

[Ejercicio 35](#)

Pseudo elementos y clases (más selectores)

Pseudo elementos

- Un pseudo elemento es una forma de seleccionar una parte de un elemento o subset de elementos seleccionado
- Pseudo elementos:
 - ::after: Permite establecer un contenido al final del elemento/s seleccionado/s
 - ::before: Permite establecer un contenido al inicio del elemento/s seleccionado/s
 - ::first-letter: Permite seleccionar la primer letra de un texto
 - ::first-line: Permite seleccionar la primer línea de un texto
 - ::selection: Permite seleccionar la selección que hace el usuario
- Para before y after podemos utilizar la propiedad **content** para establecer un contenido ([MDN](#))

Ejemplo:

```
p::after {  
  content: "?";  
}
```




```
p::first-letter {
  font-size: 30px;
  color: red;
}

div::selection {
  background-color: pink;
}
```

Práctica

[Ejercicio 36](#)

Pseudo clases

- Las pseudo clases describen un estado particular de un elemento
- Existe una gran cantidad de pseudo clases y pueden encontrar el listado completo en el sitio de [MDN](#)
- En este curso vamos a utilizar las siguientes pseudo clases:
 - :active: Selecciona un elemento que esta activo (por ejemplo un link al que le estamos haciendo click)
 -  Representa un hipervinculo que no fue visitado
 - :visited: Representa un hipervinculo que fue visitado
 - :hover: Selecciona un elemento al que estamos posicionando el cursor encima
 - :first-child: Representa el primer elemento
 - :last-child: Representa el último elemento
 - :nth-child(): Podemos seleccionar un elemento en cualquier posición nth-child(2).
 - También podemos utilizar los valores **odd** para representar los impares y **even** para los pares

Ejemplo:

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

a:visited {
  color: violet;
}

li:first-child {
  color: red;
}

tr:nth-child(odd) {
```

```
background-color: gray;
}
```

Práctica

[Ejercicio 37](#)

Cursor

- Por medio de la propiedad **cursor** cambiamos la forma en al que se ve el cursor en nuestros documentos/elementos
- Esta propiedad acepta los siguientes valores:
 - **auto**: El browser define como se tiene que ver según el elemento o acción que pueda hacer
 - **crosshair**: Es una cruz como si fuera un selector
 - **default**: Es el cursor por defecto, generalmente es una flecha
 - **pointer**: Es la manito que vemos al hacer click en un elemento
 - **move**: Con este cursor le damos a entender al usuario que puede hacer drag de un elemento ya que son flechas para distintos lados
 - **text**: Es el selector de texto como si queremos escribir en algún input
 - **wait**: Es una forma de decirle al usuario que tiene que esperar que alguna acción se termine
 - **help**: Es un signo de pregunta como podemos ver si queremos ayuda o más información sobre un tema
- Utilizando estos cursores podemos darle a entender al usuario que acción se puede realizar, por ejemplo si ponemos la manito arriba de un elemento el usuario asume que puede hacer click (por ahora sólo es un efecto visual)
- Para saber más de esta propiedad pueden leer el [sitio de MDN](#)

****Ejemplo: ****

```
body { cursor: pointer; }
button { cursor: help; }
```

Práctica

[Ejercicio 38](#)

Listas

Diseño de listas

- Por medio de la propiedad **list-style-type** podemos establecer que diseño tiene la lista
- Para listas desordenadas podemos utilizar:
 - none
 - disc
 - circle
 - square
- Para listas ordenadas:
 - decimal
 - decimal-leading-zero
 - lower-alpha
 - upper-alpha
 - lower-roman
 - upper-roman

Ejemplo:

```
ul {  
  list-style-type: none;  
}  
  
ol {  
  list-style-type: decimal-leading-zero;  
}
```

- De esta forma cambiamos la forma en que se ven las listas

Posicionamiento

- También podemos definir la posición donde queremos posicionar el diseño/dibujo de la tabla
- Para esto tenemos la propiedad **list-style-position** que acepta 2 valores:
 - **outside:** Es la posición normal que tienen las listas
 - **inside:** Establece que el diseño tiene que ser parte del elemento li y se ve dentro de él

Ejemplo:

```
ul {  
  list-style-position: inside;  
}  
  
ol {
```

```
list-style-position: inside;
}
```

Imágenes

- Podemos establecer como diseño una imagen por medio de la propiedad **list-style-image**
- Acepta como valor una ruta de una imagen por medio de la función **url(imagen.png)**

Ejemplo:

```
ul {
  list-style-image: url("garfield.png");
}
```

Todo junto

- Podemos escribir todas estas propiedades utilizando la propiedad **list-style**
- El orden de los valores es: list-style: list-style-type list-style-position list-style-image

Ejemplo:

```
ul {
  list-style: square inside;
}
ol {
  list-style: upper-roman outside;
}
```

Práctica

[Ejercicio 39](#)

Tablas

- Para las tablas podemos utilizar muchas de las propiedades que ya vimos como border, width, padding, height, background-color, margin, etc
- Nos quedan por conocer algunas propiedades que nos permiten modificar los bordes y como se comportan en las tablas
- Estas propiedades son: **border-spacing** y **border-collapse**

Espaciado de borde

- Por medio de la propiedad **border-spacing** podemos establecer la distancia que tienen que tener los bordes entre si, esto va a dar más espacio exterior entre las celdas
- Acepta un valor para todos los lados
- Acepta dos valores siendo el primero el valor horizontal y el segundo el vertical
- Para saber más sobre esta propiedad pueden entrar en el [sitio de MDN](#)

Ejemplo:

```
table {  
  border-spacing: 10px 20px;  
}
```

Práctica

[Ejercicio 40](#)

Agrupar Bordes

- Por medio de la propiedad **border-collapse** podemos establecer los siguientes comportamientos de la tabla y sus bordes:
- **collapse**: Los bordes son colapsados en un solo borde, es decir que no se pueden distanciar usando la propiedad **border-spacing**
- **separate**: Los bordes están sueltos por lo cual los podemos distanciar unos con otros

Práctica

[Ejercicio 41](#)

Forms

- Los formularios y los inputs tienen las mismas propiedades que vimos hasta el momento

Práctica

[Ejercicio 42](#)

Práctica Especial Sitio 01

[Ejercicio 43](#)

Layouts

- Como vimos los elementos pueden ser en bloque (No dejan que otro elemento este en la misma linea) y en linea (Se rodea de texto)
- Es común agrupar elementos en secciones usando un elemento div o si queremos también podemos utilizar los nuevos elementos de HTML5 para cada sección de nuestro documento
- En CSS podemos tener distintos tipos de layout para mostrar nuestros elementos

Posicionamiento Normal

- Cada elemento en bloque aparece en una línea nueva haciendo que el resto del contenido baje como se espera. Este es el comportamiento por default
- Para establecer el posicionamiento de los elementos utilizamos la propiedad **position**
- Si queremos que un elemento se comporte de manera normal no tenemos que agregar ninguna propiedad ya que como dijimos es la forma default
- En caso de tener que establecer que un elemento tiene que volver a tener posición normal utilizamos el valor **static**

Ejemplo:

```
div {  
  position: static;  
}
```

Posicionamiento Relativo

- Esta opción hace que un elemento ocupe el lugar que ocuparía en el flow regular (normal) pero nos da la opción de mover su posición con valores de top, left, right y bottom.
- Para establecer que un elemento se posiciona de manera relativa tenemos que setear el valor de la propiedad **position** en **relative**
- Podemos decir que del lugar donde esta el elemento lo podemos mover en distintas direcciones pero siempre relativo al lugar que ocupa.
- El resto de los elementos ocupa su lugar normal
- Los valores top, left, right y bottom aceptan una unidad de medida que puede ser pixeles u otros valores de los vistos

Ejemplo:

```
p {  
  position: relative;  
  top: 10px;  
  left: 20px;  
}
```

Práctica

[Ejercicio 44](#)

Posicionamiento absoluto

- Con este posicionamiento podemos establecer la posición absoluta que queremos que ocupe un elemento en relación a su contenedor. Este elemento es removido del flow normal por lo cual no afecta al resto de los elementos. Estos elementos posicionados de manera absoluta son scrolleados acompañando el scroll del usuario.

Ejemplo:

```
p {  
  position: absolute;  
  top: 10px;  
  left: 20px;  
}
```

Práctica

[Ejercicio 45](#)

Posicionamiento Fixed

- Este posicionamiento es similar al absoluto pero en lugar de ser posicionado de forma absoluta al elemento padre lo hace en relación al browser, es decir que este elemento se posiciona usando las coordenadas del browser y no del elemento padre. El resto de los elementos siguen funcionando de la misma forma, es decir que hacen como si este elemento fixed no existiera.

Ejemplo:

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

Práctica

[Ejercicio 46](#)

z-index

- Al cambiar la forma en que se comportan o posicionan los elementos podemos controlar el nivel de profundidad que utiliza cada uno utilizando la propiedad **z-index**, es decir que si dos elementos estan posicionados en el mismo lugar podemos hacer que uno se vea arriba del otro aumentando el nivel de z-index. Por ejemplo si tengo 2 elementos en la misma posición fixed puedo cambiar el nivel de uno a z-index:1 y el otro a z-index:2 y de esta forma hacer que el que tiene mayor nivel de z-index se vea arriba de todo.

Ejemplo:

```
p {  
  z-index: 100  
}
```

Práctica

[Ejercicio 47](#)

Elementos Flotantes

- Podemos establecer que un elemento flote hacia la izquierda o derecha de un elemento, el resto de los elementos se posicionan al costado de dicho elemento.
- Utilizamos la propiedad **float** y los valores **left** (izquierda) o **right** (derecha)
- Al utilizar elementos flotando conviene establecer un **width**

Ejemplo:

```
div {  
  float: left;  
  width: 200px;  
}
```

- Al tener elementos flotando el resto de los elementos se van a posicionar al costado del elemento flotante
- Si queremos que los elementos vuelvan a tener el flow normal podemos utilizar la propiedad **clear** que acepta los valores **left**, **right** o **both**

- Por medio de esta propiedad establecemos que no queremos tener elementos flotantes a la izquierda, derecha o ninguno de los dos

Ejemplo:

```
div {  
  clear: left;  
}
```

- En este ejemplo el div no va a tener a nadie flotando a su izquierda y se recupera el flow normal de los elementos

Práctica

Ejercicio 48

- Podríamos tener 3 secciones que esten flotando para hacer una parte del sitio en 3 columnas

Ejemplo:

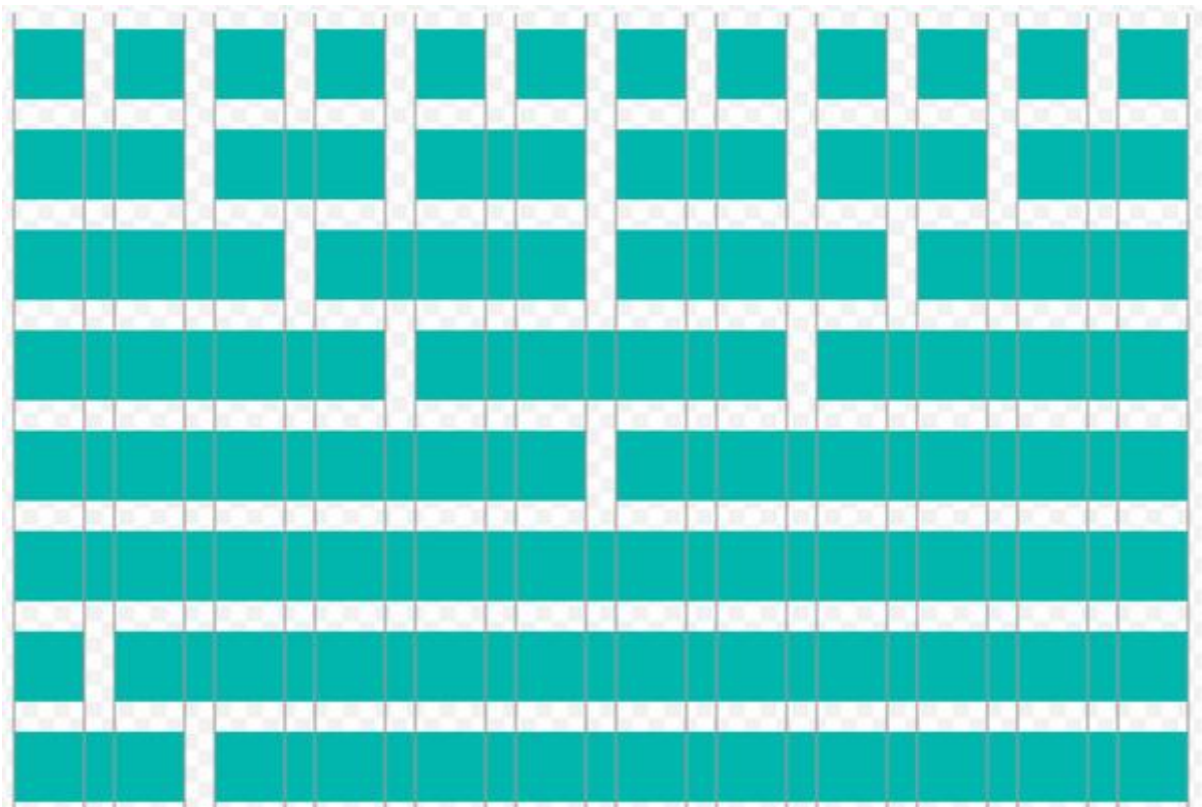
```
div {  
  width: 300px;  
  float: left;  
}
```

- Podemos aprender más en el sitio de <http://es.learnlayout.com>
- Otra buena fuente es [Learn CSS Positioning in Ten Steps \(inglés pero se entiende\)](#)
- Actualmente se piensa el diseño web como si fuera imprenta (o los diarios) y existe el concepto de grilla
 - Podemos ver el sitio de 960.gs que tiene buenos ejemplos
 - Básicamente la página se divide en pequeñas secciones donde el ancho máximo es 960px
 - Tenemos máximo 12 columnas
 - Si queremos usar toda una sección utilizamos las 12 columnas
 - Si queremos utilizar 2 columnas donde una es más grande y la otra más chica podemos elegir una columna de 4 espacios y otra de 8
 - Si queremos hacer una página de 2 lados nada más y ocupar la mitad para cada sección podemos hacer dos columnas de 6 y 6
 - En caso de querer hacer una página a 3 columnas podríamos dividirlo en 3 secciones de 4 columnas cada una

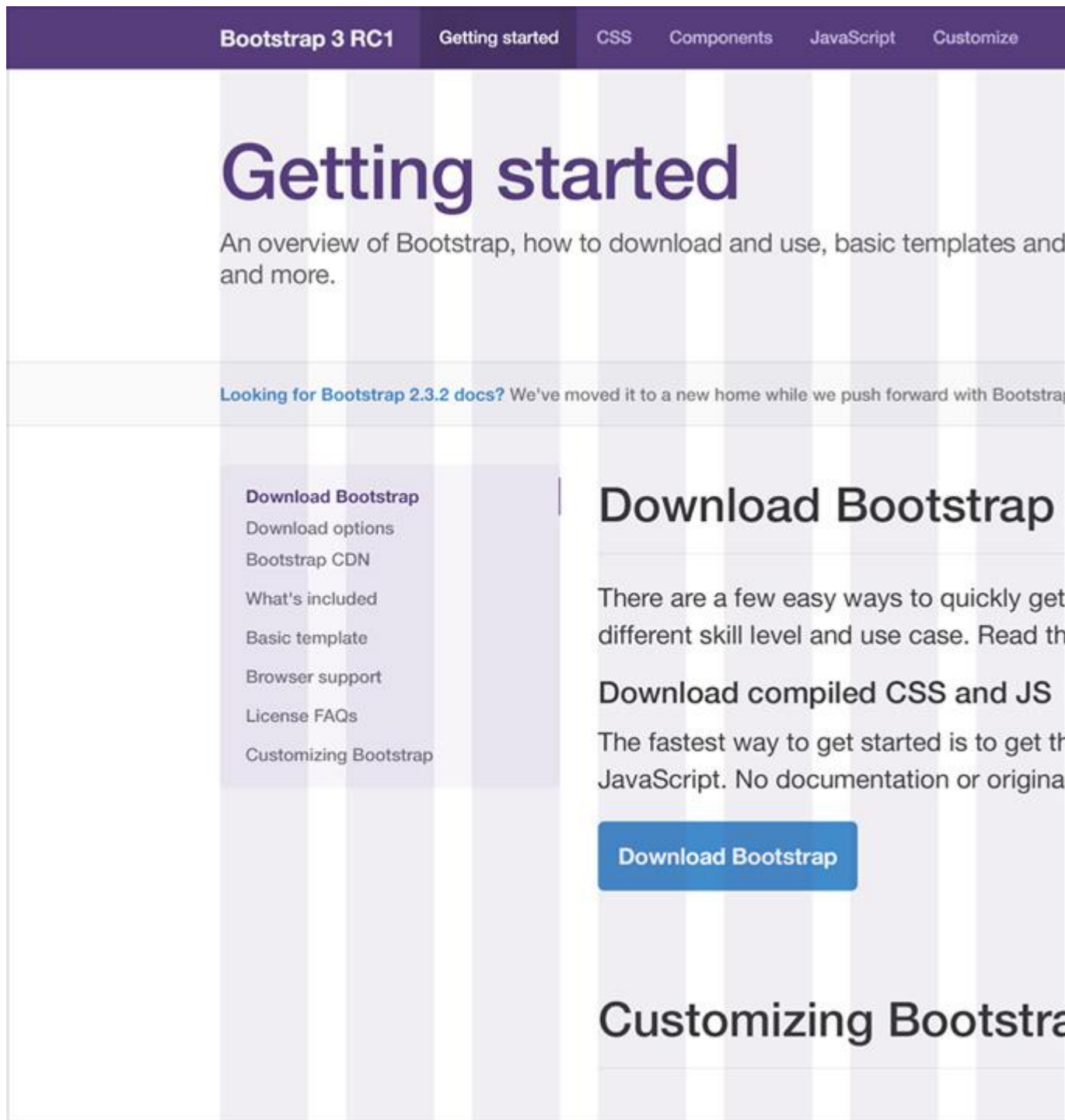
- Lo importante es que la sumatoria de columnas de 12 como máximo
- Como vemos se pueden utilizar muchas opciones y la grilla nos abstrae de la forma de crear todo esto
 - Algunos sistemas de grillas son más flexibles
 - Se pueden conseguir grillas de 16 columnas
 - Bootstrap también trae su propio [sistema de grilla](#)
 - Foundation también: [grillas](#)

Ejemplos de grillas:

Ejemplo de distintas opciones



Ejemplo de sitio



Media Queries

- Por medio de Media Query podemos establecer atributos visuales dependiendo del tipo de dispositivo que esta viendo el documento
- Varios de las librerías o frameworks de vista que mencionamos (bootstrap o foundation) ya vienen preparado para soportar este tipo de detección y adaptarse de la mejor forma

- Utilizamos **@media** para establecer que vamos a detectar un tipo de dispositivo
- También podemos agregar condicionales para detectar distintos tipos de dispositivos
- Pueden leer más en el [sitio de MDN](#)

Ejemplo:

```
@media (max-width: 600px) {
  p {
    color: red;
  }
}
```

- Se pueden agregar más condicionales utilizando la palabra reservada **and**

Ejemplo:

```
@media (min-width: 200px) and (max-width: 600px) {
  .rojo {
    color: red;
  }
}
```

- Otra de las opciones es detectar la posición del dispositivo para saber si esta en formato vertical u horizontal
- Utilizamos la propiedad orientation y los valores **landscape** o **portrait**
- Pueden leer más sobre esto en el [sitio de MDN\(inglés\)](#)

Ejemplo:

```
@media (max-width: 600px) and (orientation: portrait) {
  .rojo {
    color: red;
  }
}
```

- Para trabajar con celulares u otros dispositivos es posible que tengamos que agregar el siguiente encabezado
- Pueden leer más sobre este tema en el [siguiente sitio \(en inglés\)](#)

Ejemplo:

```
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
</head>
```

- Con las [developers tools](#) podemos probar diferentes dispositivos

Práctica

[Ejercicio 49](#)

Flexbox

- Próxima sección de Flexbox

Práctica Especial Sitio 02

[Sitio 02](#)