



[Hacé click acá para dejar tu feedback sobre esta clase.](#)



[Hacé click acá completar el quiz teórico de esta lecture.](#)

## CSS Avanzado

---

En esta Lesson se verán los siguientes temas:

- Frameworks CSS
- CSS Preprocessors

### Frameworks CSS

En primer lugar un 'Framework' es un marco de referencia o marco de trabajo que nos provee distintas herramientas que se puede utilizar para facilitar el desarrollo de aplicaciones, ofreciéndonos una forma estándar y por lo general más simple para programar. En particular para el caso de un 'Framework CSS', se refiere a un conjunto de estilos predefinidos que pueden utilizarse para elaborar una interfaz de usuario atractiva sin necesidad de tener que definir a mano todas y cada una de las propiedades CSS de nuestros elementos HTML.

Existen una gran variedad de Frameworks CSS pero entre los más utilizados en la actualidad se encuentran:


- Bootstrap
- Foundation
- Bulma
- UIKit
- Semantic UI

### Bootstrap

En esta lesson nos centraremos en Bootstrap que es el más utilizado de ellos. Las ventajas que nos ofrece este Framework es que ya tiene componentes con estilos predefinidos que podemos reutilizar para ganar tiempo.

Por ejemplo supongamos que quisiéramos crear en nuestra página web un botón rojo con bordes redondeados y texto blanco:


```
<!DOCTYPE html>
<html>
<style media="screen">
  button {
    color: white;
    background-color: red;
    border-radius: 5px;
  }
</style>
<body>
<button type="button">Click</button>
</body>
</html>
```

Con ese código lo que obtendríamos es lo siguiente:  alt text

Ahora bien, vamos a intentar lo mismo utilizando Bootstrap. Para ello necesitaremos agregar a nuestro HTML una referencia a la librería de Bootstrap para poder utilizar todos sus beneficios.

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
integrity="sha384-
Vkoo8x4CGs03+Hh xv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
crossorigin="anonymous">
</head>
<style media="screen">
  .buttonComun {
    color: white;
    background-color: red;
    border-radius: 5px;
  }
</style>
<body>
<button type="button" class="buttonComun">Boton Común</button>
<button class="btn btn-danger">Boton Bootstrap</button>
</body>
</html>
```

*Veamos que en el header se agrego un link hacía Bootstrap*

Veamos ahora como quedó nuestra página:  alt text

No tuvimos que definir ninguna propiedad CSS para nuestro nuevo botón sino que simplemente le asignamos las clases `btn` y `btn-danger` y Bootstrap se encargó del resto.

Para ver que esto no es magia, lo que está pasando por detrás es que existe un archivo CSS 'enorme' con muchísimas clases definidas que ya contienen propiedades de estilos asignadas entonces por ejemplo cuando nosotros le asignamos la clase `btn-danger` es como si estuviéramos escribiendo el siguiente código en nuestro archivo CSS o en la etiqueta `<style>` de nuestro HTML:

```
.btn-danger {  
  color: #fff;  
  background-color: #dc3545;  
  border-color: #dc3545;  
}
```

*Lo mismo sucede con la clase `btn`, le aporta a nuestro elemento más propiedades CSS*

En la página de [Bootstrap](#) podrán encontrar muchos componentes que pueden reutilizar en sus páginas web.

## Responsive Design

Cuando queremos que nuestra página se vea 'linda' en cualquier dispositivo o cambie algunas características ya sea en una computadora, en un teléfono celular, en una tablet o incluso en un televisor smart, necesitamos hacer algunos ajustes a las propiedades de los elementos en función del dispositivo.

### CSS Media Queries

Para poder determinar que una propiedad solo se aplique en función del tamaño de la pantalla del dispositivo tenemos la posibilidad de usar `CSS Media Queries`.

Supongamos que queremos modificar el color de fondo de la página web:

- Negro para una pantalla de 600px o menos de ancho
- Azul para una pantalla de entre 600px a 900px de ancho
- Rojo para una pantalla de más de 900px de ancho

```
body {  
  background-color: red;  
}  
  
/* Pantallas de menos de 992px de ancho */  
@media screen and (max-width: 992px) {  
  body {  
    background-color: blue;  
  }  
}  
  
/* Pantallas de menos de 600px de ancho */  
@media screen and (max-width: 600px) {  
  body {
```

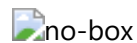
```
background-color: black;  
}  
}
```

El resultado obtenido sería el siguiente:



## Bootstrap

Supongamos ahora que queremos cambiar la cantidad de columnas que se muestren en función de la pantalla para que nos queden cuatro columnas en pantallas grandes, dos en medianas y una en pequeñas:



Podríamos realizarlo con CSS Media Queries similar al ejemplo anterior. Así que si quieren pueden intentarlo (Es un buen ejercicio para practicar lo que ya saben de CSS con esta nueva herramienta).

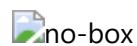
Pero ahora vamos a ver como solucionar esto utilizando el Framework que explicamos más arriba **Bootstrap**.

### Grid System

Bootstrap ya tiene integrado un sistema de grillas implementado a partir de flexbox que nos va a facilitar la tarea. Para ello utiliza cinco clases ya definidas:

- .col- (extra small devices - menos de 576px)
- .col-sm- (small devices - mayor o igual a 576px)
- .col-md- (medium devices - mayor o igual a 768px)
- .col-lg- (large devices - mayor o igual a 992px)
- .col-xl- (xlarge devices - mayor o igual a 1200px)

El sistema de grilla de Bootstrap permite colocar hasta una suma de 12 'espacios' por fila distribuyéndolos de la forma que se quiera, ya sea colocando 12 columnas de 1 'espacio', 2 columnas de 6 'espacios' o cualquier variante de combinaciones:



*También existe la opción de dejar que Bootstrap identifique la cantidad de columnas que hay y a partir de ello le asigne el mismo ancho a cada una hasta completar la totalidad de la fila (Siempre recordando que el máximo es de 12). Para ello se utiliza simplemente la clase **.col** en cada columna*

Utilizando simplemente esas clases podemos crear múltiples tipos de grillas que se adapten a nuestras pantallas.



En el gif de arriba podemos ver como en función del ancho de la pantalla va cambiando la cantidad de columnas

```
<div class="row">
  <div class="col-12 col-sm-6 col-md-3" style="background-color:black;
color:black;">.</div>
  <div class="col-12 col-sm-6 col-md-3" style="background-color:orange;
color:orange;">.</div>
  <div class="col-12 col-sm-6 col-md-3" style="background-color:yellow;
color:yellow;">.</div>
  <div class="col-12 col-sm-6 col-md-3" style="background-color:green;
color:green;">.</div>
</div>
```

La documentación completa la pueden encontrar [acá](#)

## CSS Preprocessors

Un preprocesador CSS es un programa que te permite generar CSS a partir de la syntax única del preprocesador. Existen varios preprocesadores CSS de los cuales escoger, sin embargo la mayoría de preprocesadores CSS añadiran algunas características que no existen en CSS puro, como variable, mixins, selectores anidados, entre otros. Estas características hacen la estructura de CSS más legible y fácil de mantener.

Estos son algunos de lo preprocesadores CSS más populares:

- SASS
- LESS
- Stylus
- PostCSS

Si quieren jugar un poco con distintos preprocesadores [codepen](#) nos brinda un entorno de fácil configuración. En la configuración del panel de CSS podemos seleccionar el que queramos

### LESS (Leaner Style Sheets)

En esta lesson nos centraremos en LESS ya que es uno de los más utilizados y tiene un tipo de sintaxis muy similar al código CSS pero con ciertos agregados por lo que va a ser más sencillo entenderlo.

### Variables

LESS nos permite utilizar variables dentro de nuestro archivo de estilos para evitar la repetición innecesaria de definiciones de propiedades, por ejemplo no nos resulta tan fácil recordar un código de un color en formato hexadecimal en cambio si pudiéramos definirlo una única vez y asignárselo a una variable con un nombre representativo, nos sería mucho más sencillo.

Ejemplo de código SCSS:

```
@color-fondo: #F55;
@width: 10px;
@height: @width + 10px; /* También es posible realizar operaciones sobre las
variables */

h1 {
    background-color: @color-fondo;
    width: @width;
    height: @height;
}
```

*En este ejemplo estamos creando variables con un color y medidas determinadas que van a poder ser reutilizadas en distintos componentes y clases las veces que queramos*

Luego este código va a ser compilado en a un archivo CSS para que pueda ser interpretado por los navegadores por lo que el ejemplo anterior quedaría así:

```
h1 {
    background-color: #F55;
    width: 10px;
    height: 20px;
}
```

Al igual que en otros lenguajes de programación, las variables tienen un scope determinado, primero se analiza si en el contexto actual se encuentra definida dicha variable y si no la encuentra la buscará en el scope padre.

```
@var: red;

#page {
    @var: white;
    color: @var; // white
}
```

La variable de la primer línea podrá utilizarse dentro del resto de las definiciones pero la que se encuentra dentro de `#page` sólo será válida allí. Por otra parte, el valor correspondiente a la propiedad `color` va a ser `white` ya que en dicho contexto si se encuentra definido el valor de `@var`. En cambio si tuvieramos algo como lo siguiente:

```
@var: red;

#page {
    color: @var; // red
}
```

En este caso el valor de `@var` sería `red` ya que en su contexto no está definida la variable pero en el contexto global sí por lo que toma su valor de allí.

También es posible utilizar variables dentro de los nombres de los selectores, de las propiedades e incluso en URL's.

## Selectores

```
@my-selector: banner;

.#{@my-selector} {
  font-weight: bold;
  line-height: 40px;
  margin: 0 auto;
}
```

## Propiedades

```
@property: color;

.widget {
  @property: #0ee;
  background-@property: #999;
}
```

## URLs

```
@images: "../img";

body {
  color: #444;
  background: url("@images/white-sand.png");
}
```

## Lazy evaluation

No es necesario declarar las variables antes de usarlas, por lo que el siguiente código sería válido:

```
.lazy-eval {
  width: @var;
}

@var: 200px;
```

## Funciones

LESS nos provee de ciertas funciones que nos permiten transformar colores, manipular strings y realizar cálculos matemáticos.

Ejemplo de utilización:

```
@base: #f04615;
@width: 0.5;

.class {
  width: percentage(@width);
  color: saturate(@base, 5%);
  background-color: spin(lighten(@base, 25%), 8);
}
```

En este caso por un lado con la función `percentage` estamos convirtiendo el valor `0.5` en `5%` y por otro lado, con la función `saturate` estamos incrementando la saturación del color base en un 5%.

Para ver la documentación completa de las funciones disponibles ingresar [aquí](#)

## Anidado

LESS también nos permite anidar definiciones de estilos CSS similar a como es una estructura HTML:

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }
  li {
    display: inline-block;
  }
  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

En este caso estamos asignándole propiedades a los elementos `ul`, `li` y `a` que se encuentren dentro de un `nav`

Los mismo puede realizarse con la directiva `@media`:

El siguiente código:



```
.component {
  width: 300px;
  @media (min-width: 768px) {
    width: 600px;
    @media (min-resolution: 192dpi) {
      background-image: url(/img/retina2x.png);
    }
  }
  @media (min-width: 1280px) {
    width: 800px;
  }
}
```

Se traduciría en:

```
.component {
  width: 300px;
}
@media (min-width: 768px) {
  .component {
    width: 600px;
  }
}
@media (min-width: 768px) and (min-resolution: 192dpi) {
  .component {
    background-image: url(/img/retina2x.png);
  }
}
@media (min-width: 1280px) {
  .component {
    width: 800px;
  }
}
```

## Importación

La directiva `@import` nos permite incluir el contenido de otros archivos en el actual. Supongamos que tenemos un archivo less llamado "general.less" como el siguiente:

```
html,
body,
ul,
ol {
  margin: 0;
  padding: 0;
}
```

Podríamos importar dichas definiciones de atributos en otro less de la siguiente forma:

```
@import "general";

body {
  font-family: Helvetica, sans-serif;
  font-size: 18px;
  color: red;
}
```

De esta forma en nuestro último archivo de estilos también vamos a poder contener las definiciones de "general.less".

Observen que no es necesario aclarar la extensión del archivo *general*, LESS automáticamente asume que es un archivo de estilos válido

## Mixins

Los mixins nos permiten incluir un set de propiedades ya definido dentro de otro.

```
.important-text {
  color: black;
  font-size: 25px;
  font-weight: bold;
}
```

Ahí estamos creando el mixin llamado *important-text* que luego podemos utilizar de la siguiente forma:

```
.danger {
  .important-text();
  background-color: red;
}

.success {
  .important-text();
  background-color: green;
}
```

Esto se va a traducir a código CSS equivalente a:

```
.danger {
  color: red;
  font-size: 25px;
  font-weight: bold;
  border: 1px solid blue;
}
```

```
background-color: green;
}
```

Es decir lo que sucedió es que se inyectaron todas las propiedades definidas en el mixin dentro de la clase *danger* y *success*. También es posible utilizar ids como mixins (*#b()*;) )

## Parametros

Los mixin pueden recibir parámetros:

```
.bordered(@color; @width) {
  border: @width solid @color;
}

.myArticle {
  .bordered(blue; 1px);
}

// Es posible indicar el nombre del parámetro al invocar el mixin
// para evitar tener que respetar un orden en particular
.myArticle-2 {
  .bordered(@width: 20px; @color: #33acfe);
}
```

Aquí lo que estamos haciendo es definir un mixin que recibe dos parámetros (color y width) que luego van a ser utilizados para definir el borde del elemento. Con ello podemos reutilizar el mixin simplemente llamándolo con diferentes colores o anchos como en el ejemplo que se le está dando un color azul y un borde de un pixel a los elementos con la clase *myArticle*

## Valores por defecto

Adicionalmente se puede setear un valor por defecto para dichos parámetros para que, en el caso de que no se les indique un valor, tomen el por defecto:

```
.bordered(@color: blue; @width: 1px) {
  border: @width solid @color;
}

.myArticle-default {
  .bordered();
}
```

## Variable @arguments

La variable *@arguments* dentro de un mixin contiene todos los argumentos que le fueron suministrados a dicho mixin.

```
.box-shadow(@x: 0; @y: 0; @blur: 1px; @color: #000) {  
  box-shadow: @arguments;  
}  
.big-block {  
  .box-shadow(2px; 5px);  
}
```

Esto resultaría en:

```
.big-block {  
  box-shadow: 2px 5px 1px #000;  
}
```

## Herencia

Por último también es posible, heredar/compartir las propiedades de un selector en otro. Esto es útil para aquellos casos en los que entre dos selectores comparten la mayor parte de los atributos pero tienen una o algunas pequeñas diferencias.

```
.button-basic {  
  border: none;  
  padding: 15px 30px;  
  text-align: center;  
  font-size: 16px;  
  cursor: pointer;  
}  
.button-report {  
  &:extend(.button-basic);  
  background-color: red;  
}  
.button-submit {  
  &:extend(.button-basic);  
  background-color: green;  
  color: white;  
}
```

*En este caso el botón de report y de submit extienden las propiedades del botón básico manteniendo todas sus propiedades pero agregándole algunas más que son propias de ellas*

Con esto cubrimos la mayor parte de las funcionalidades agregadas por LESS pero existen otras que para aquel que le interese indagar aun más sobre este tema puede acceder a la documentación oficial [aquí](#)

## Homework

Completa la tarea descrita en el archivo [README](#)