

Funciones básicas

Módulo 03

Implementación SASS

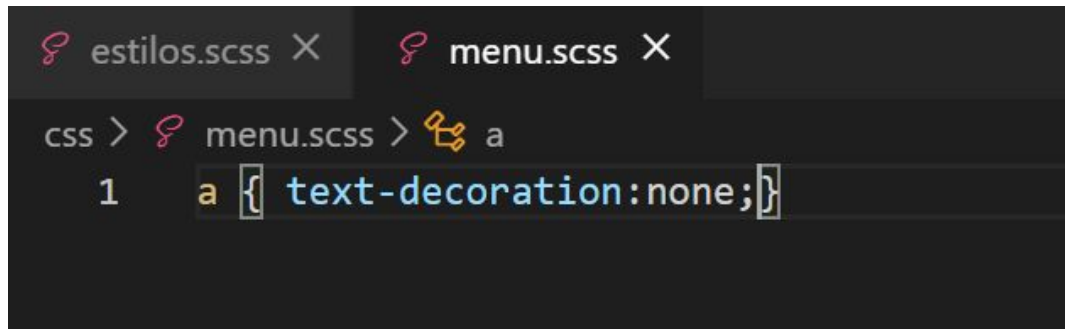
Implementación

SASS

Una de las facilidades al trabajar con sass es la modularización.

Por ejemplo ir separando nuestros **archivos en diferentes estilos dependiendo de qué elemento está siendo afectado de nuestra estructura.**

Por ejemplo generamos **dos archivos .scss**



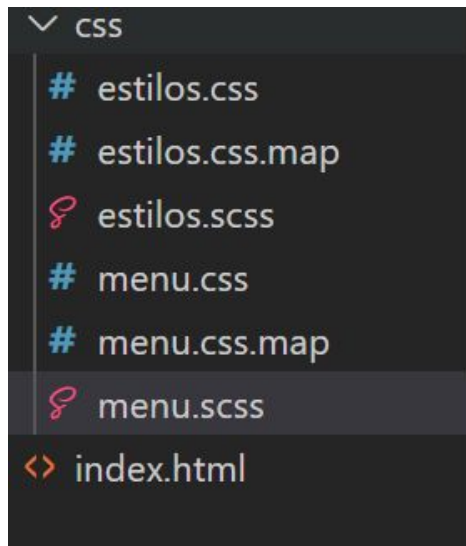
```
estilos.scss X menu.scss X
css > menu.scss > a
1 a { text-decoration: none; }
```



Implementación

SASS

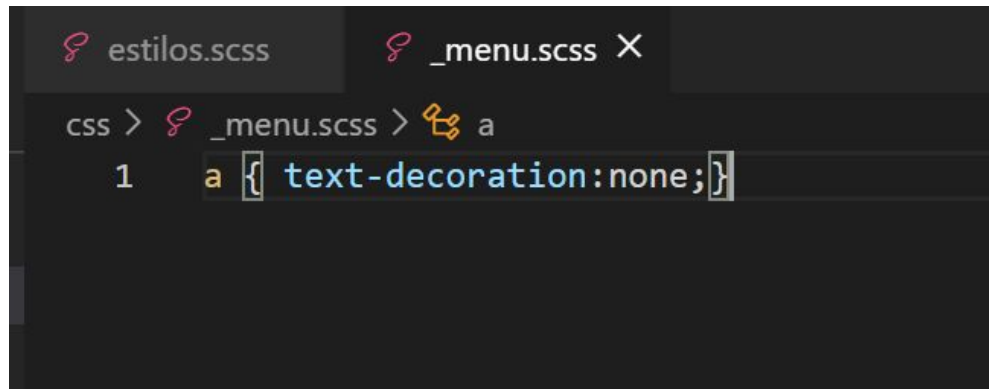
Sin embargo en el paso anterior debemos considerar qué se generará un **segundo archivo también .css**



Implementación

SASS

Por lo tanto debemos hacer un pequeño cambio pues esta no es la idea,

A screenshot of a code editor with a dark background. At the top, there are two tabs: 'estilos.scss' and '_menu.scss'. The '_menu.scss' tab is active and shows a single line of SASS code: 'a { text-decoration: none; }'. The line is numbered '1' on the left. Above the code, there is a command prompt interface showing 'css > _menu.scss > a' with a cursor pointing at the 'a'.

De esta forma le anunciamos al **compilador** que no efectúe una **compilación sobre este archivo específico**,



Implementación

SASS

Luego del [paso anterior](#), importamos nuestro archivo **menu.scss** a **estilos.scss**,



```
estilos.scss X # estilos.css _menu.scss
css > estilos.scss > ...
1  @import 'menu';
2
3  $color : blue;
4  $ancho1 : 300px;
5  $mitad: $ancho1 / 2;
6
7
```

Implementación

SASS

Es importante aclarar que la última versión de SASS se actualizó la regla permitiendo trabajar con **@use**. Las diferencias son varias, pero la más importante es que la compilación se realiza una sola vez con @use no importando cuantas veces se la importe en realidad. De todas formas utilizaremos en este curso @import ya que aún sigue siendo válido y es la regla más común.

Otra de las ventajas de **@use** es que permite trabajar con reglas privadas para no hacerlas extensivas al momento de la importación. Para poder saber más sobre actualizaciones recomendamos la lectura de la página oficial de SASS, <https://sass-lang.com/>.

Para saber sobre ese punto específico,
<https://sass-lang.com/documentation/at-rules/use>



Implementación

SASS

Por último comprobamos **que en nuestro estilos.css, efectivamente se añadió nuestra regla nueva en este caso referida al selector a**

```
estilos.scss  # estilos.css  _menu.scss
css > # estilos.css > a
1  a {
2    text-decoration: none;
3  }
4
5  body {
6    background-color: blue;
7  }
8
```



Implementación

SASS

Por otro lado **vamos a suponer que encontramos lo siguiente en nuestros estilos,**

```
h1 { font-size: 2em; text-transform: uppercase; color: green;}  
h2 { font-size: 0.5em;text-transform: uppercase; color: green;}  
h3 { font-size: 0.2em;text-transform: uppercase; color: green;}
```

Si bien una solución sería un selector grupal, dado **que los tres selectores poseen dos propiedades con los mismos valores,** vamos a [trabajar con mixins.](#)



Implementación

SASS

El [código anterior podemos trabajarlo](#) desde **sass** de la **siguiente forma**,

```
✓ @mixin fuente {  
  
}
```

Al momento de trabajar con un **mixin siempre debemos asignarle un nombre** en nuestro caso hemos **decidido darle el nombre de fuente**,



Implementación

¿Cómo empezar a trabajar?

Luego, integraremos las **propiedades iguales de la siguiente manera, y las borramos de los selectores iniciales,**

```
✓ @mixin fuente {  
    text-transform: uppercase; color: ■green;  
}
```

```
h1 { font-size: 2em; }  
h2 { font-size: 0.5em;}  
h3 { font-size: 0.2em;}
```



Implementación

¿Cómo empezar a trabajar?

El [paso anterior](#) nos permite iniciar el mixin.
Ahora vamos a llamarlo,

```
@mixin fuente {  
    text-transform: uppercase; color: ■green;  
}
```

```
h1 { font-size: 2em; @include fuente(); }  
h2 { font-size: 0.5em; @include fuente();}  
h3 { font-size: 0.2em; @include fuente();}
```



Implementación

sass

El resultado será,
luego de compilado
en nuestros **estilos.css**
como vemos en la
imagen de la derecha,

```
estilos.scss  # estilos.css  _menu.scss
css > # estilos.css > a
1  a {
2    text-decoration: none;
3  }
4
5  h1 {
6    font-size: 2em;
7    text-transform: uppercase;
8    color: green;
9  }
10
11 h2 {
12   font-size: 0.5em;
13   text-transform: uppercase;
14   color: green;
15 }
16
17 h3 {
18   font-size: 0.2em;
19   text-transform: uppercase;
20   color: green;
21 }
22
```



Implementación

sass

Podemos avanzar mucho más trabajando con parámetros. Como han visto, este trabajo con **mixin es similar a las funciones de javascript.**

Por esa razón el uso de parámetros **no debe sorprendernos,**

```
@mixin fuente($color) {  
    text-transform: uppercase; color: $color;  
}
```



Implementación

sass

Como vimos [en el paso anterior](#), hemos utilizado un **parámetro** llamado **\$color**, vamos entonces a utilizarlo,



```
✓ @mixin fuente($color) {  
    text-transform: uppercase; color: $color;  
}  
  
h1 { font-size: 2em; @include fuente(■ red); }  
h2 { font-size: 0.5em; @include fuente(■ green); }  
h3 { font-size: 0.2em; @include fuente(■ blue); }
```

Implementación

sass

Como vimos en la [diapositiva anterior](#), hemos trabajado con un **parametro de color**, si compilamos el resultado **en el .css será el siguiente**,

```
h1 {  
  font-size: 2em;  
  text-transform: uppercase;  
  color: ■ red;  
}  
  
h2 {  
  font-size: 0.5em;  
  text-transform: uppercase;  
  color: ■ green;  
}  
  
h3 {  
  font-size: 0.2em;  
  text-transform: uppercase;  
  color: ■ blue;  
}
```



Implementación

sass

Empecemos a trabajar!

Ahora generemos una regla de estilo cómo usualmente hacemos,



```
<> index.html X  estilos.scss X
C: > Users > sacab > OneDrive > Escritorio > cosas > practicaSass > css > estilos.scss > body
1  body { background-color: blue; }
```

Implementación

sass

También podemos pasar **más de un parámetro**,

```
✓ @mixin fuente($color,$transformacion) {  
    text-transform: $transformacion; color: $color;  
}
```

De esta manera podemos [flexibilizar toda nuestra hoja de estilo de la siguiente manera.](#)



Implementación

sass

Empecemos a trabajar!

Luego de generar nuestro segundo parámetro en el mixin, vamos a utilizarlo en nuestro scss.



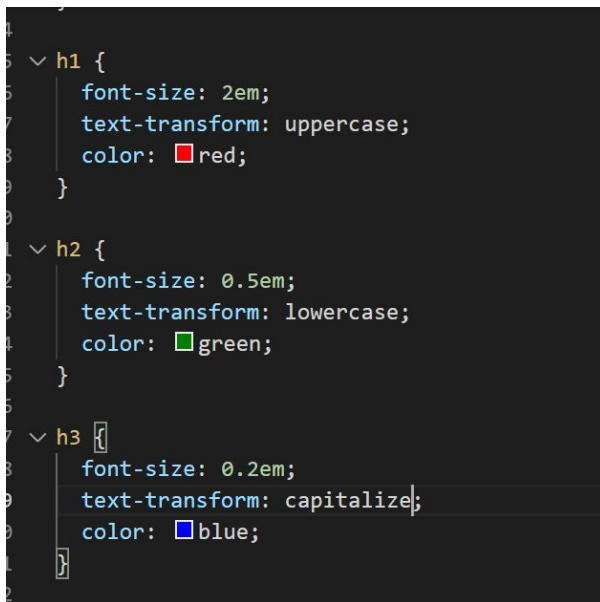
```
h1 { font-size: 2em; @include fuente(■red, uppercase); }  
h2 { font-size: 0.5em; @include fuente(■green, lowercase); }  
h3 { font-size: 0.2em; @include fuente(■blue, capitalize); }
```

Si compilamos el resultado en el .css será.

Implementación

sass

El resultado del código anterior trabajado en el **scss. será,**



Implementación

sass

Empecemos a trabajar!

También podemos partir de un **ejemplo en .css**,

```
main {  
  width: 900px;  
}  
  
#seccion1 {  
  width: 400px;  
}  
  
#seccion2 {  
  width: 500px;  
}
```



Implementación

sass

Empecemos a trabajar!

El [código anterior funciona](#), en una estructura pero no de forma responsiva. Entonces vamos a **suponer que cambiamos el width el contenedor principal**,

```

    main {
        width: 100%;
    }

    #seccion1 {
        width: 400px;
    }

    #seccion2 {
        width: 500px;
    }

```



Implementación

sass

En el caso anterior, debemos entonces **calcular a cuánto sería equivalente cada una de las columnas** para poder trabajarlas porcentualmente. Sin embargo **sass** nos permite hacer el cálculo directamente en nuestro **.scss**,

```
}  
  
#seccion1 {  
  width: 400px/900px * 100%;  
}  
  
#seccion2 {  
  width: 500px/900px * 100%;  
}
```



Implementación

sass

El resultado del [código anterior](#) será luego de compilado,

```
main {  
  width: 100%;  
}  
  
#seccion1 {  
  width: 44.44444%;  
}  
  
#seccion2 {  
  width: 55.55556%;  
}
```



Revisión

- Repasar los conceptos vistos de **sass**.
- Trabajar con el **bonus track** debajo de todo para practicar los **elementos individualmente**.
- Ver todos los videos y materiales necesarios antes de continuar



¡Muchas gracias!

¡Sigamos trabajando!