

```
isVideo = ( (type == "image") || (type == "video") )  
isUrl = ( (type == "url") || (type == "image") )  
isElement = ( (type == "element") || (type == "image") )  
isObject = ( (typeof subject == "object") )  
  
// Check if boxer is already active, return early  
if ($("#boxer").length > 1 || (isImage || isVideo)  
    return;  
}  
  
// Kill event  
_killEvent(e);  
  
// Cache internal data  
data = $.extend({}, {  
    $window: $(window),  
    $body: $("body"),  
    $target: $target,  
    $object: $object,  
    visible: false,  
    resizeTimer: null,  
    touchTimer: null,  
    gallery: {  
        active: false  
    }  
});  
e.data.active,
```

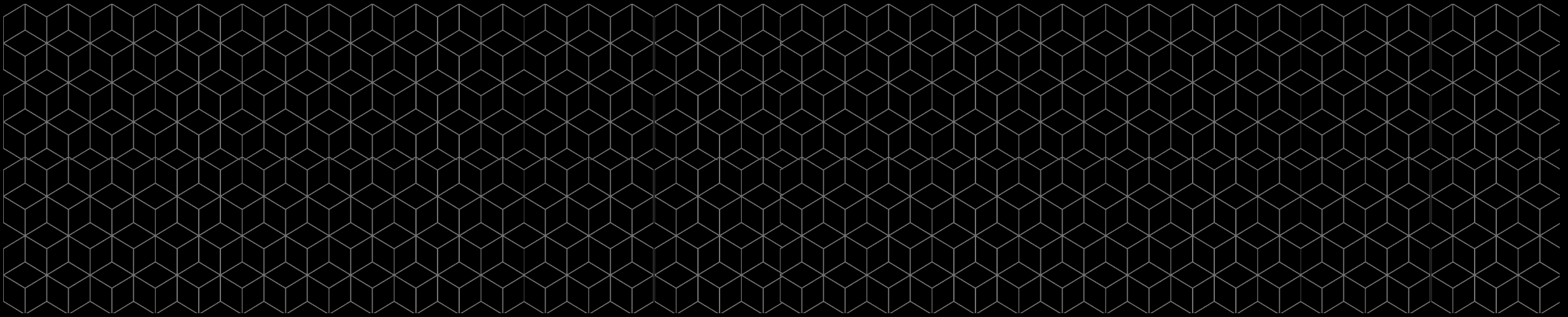
Buenas Prácticas

Marzo 12 - 2021



Agenda

- ❑ Normalize
- ❑ Comprobar Compatibilidades Web
- ❑ Nombres de Variables para Colores
- ❑ Pseudo-elementos
- ❑ Pseudo-clases
- ❑ Deuda Técnica
- ❑ Ejercicio Paso a Paso



Normalize



Normalizar estilos CSS de elemento HTML



Acceder a todos los elementos HTML (*): Está bien implementarlo para reiniciar o normalizar los estilos css de elementos HTML para que sean iguales en todos los navegadores web, se utiliza por ser rápido, pero no es la mejor forma de hacerlo.

```
1  * {  
2    padding: 0;  
3    margin: 0;  
4  }  
5  
6  *,  
7  *::after,  
8  *::before {  
9    box-sizing: border-box;  
10 }
```

Normalize.css: Es un proyecto desarrollado para ver cuáles estilos CSS son diferentes entre cada uno de los navegadores y con esto se intentan normalizar los estilos para que sean iguales en todos los navegadores. Se utiliza para procurar que el diseño se vea igual en todos los navegadores. Existen herramientas como reset CSS y este accede a todos los elementos HTML y reiniciarlos.

Con normalize.css no se van a reiniciar los estilos a 0, se va a encargar de arreglar algunos bugs y diferencias que hay entre navegadores.

Diferencia entre reset.css y normalize.css: <https://www.kodetop.com/diferencias-entre-reset-css-y-normalize-css/>

Reset.css: <https://meyerweb.com/eric/tools/css/reset/>

Normalize.css: <https://necolas.github.io/normalize.css/>



Comprobar Compatibilidades Web

Can I Use

Can I Use: Permite mostrar las compatibilidades de los estilos CSS en diferentes navegadores. <https://caniuse.com/>

Can I use

display: flex


? ⚙ Settings

3 results found

☒ Caniuse (1) ☒ MDN (2)

#

☆

CSS Flexible Box Layout Module  - CR

Usage

% of all users

Global

97.44% + 1.63% = 99.07%

unprefixed:

97.09% + 1.04% = 98.13%

Method of positioning elements in horizontal or vertical stacks. Support includes all properties prefixed with `flex`, as well as `display: flex`, `display: inline-flex`, `align-content`, `align-items`, `align-self`, `justify-content` and `order`.

Current aligned Usage relative Date relative Filtered All ⚙

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
					10-11.5											
		1 2-21	1 4-20	1 3.1-6	12.1	1 3.2-6.1										
6-9		22-27	21-28	6.1-8	15-16	7-8.4		1 2.1-4.3	12							
2 4 10	12-88	28-85	29-88	9-13.1	17-72	9-13.7		4.4-4.4.4	12.1				4-12.0			
4 11	89	86	89	14	73	14.4	all	81	62	89	86	12.12	13.0	10.4	7.12	2.5
		87-88	90-92	TP												

Notes Test on a real browser Sub-features Known issues (9) Resources (13) Feedback

Most partial support refers to supporting an [older version](#) of the specification or an [older syntax](#).

1 Only supports the [old flexbox](#) specification and does not support wrapping.

Can I use

subgrid

? ⚙ Settings

3 results found

☒ Caniuse (1)

☒ MDN (2)

#

CSS Subgrid  - WD

Usage % of all users ?
Global 3.37%

☆

Feature of the CSS Grid Layout Module Level 2 that allows a grid-item with its own grid to align in one or both dimensions with its parent grid.

Current aligned Usage relative Date relative Filtered All ⚙

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Opera Mobile	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
		2-70														
6-10	12-88	71-85	4-88	3.1-13.1	10-72	3.2-13.7		2.1-4.4.4	12-12.1				4-12.0			
11	89	86	89	14	73	14.4	all	81	62	89	86	12.12	13.0	10.4	7.12	2.5
		87-88	90-92	TP												

Notes

Test on a real browser

Known issues (0)

Resources (7)

Feedback



Nombres de Variables para Colores


Podemos crear el nombre de las variables para los colores fácilmente . <https://chir.ag/projects/name-that-color>

Ejemplo: \$miApp-silver

chir.ag

[about](#) [blog](#) [contact](#) [pics](#) [projects](#) [read](#) [hangouts](#) [www's](#)

search




found: chir.ag / art

projects > name that color


Name that Color

Click & drag over the Color Wheel to make a color



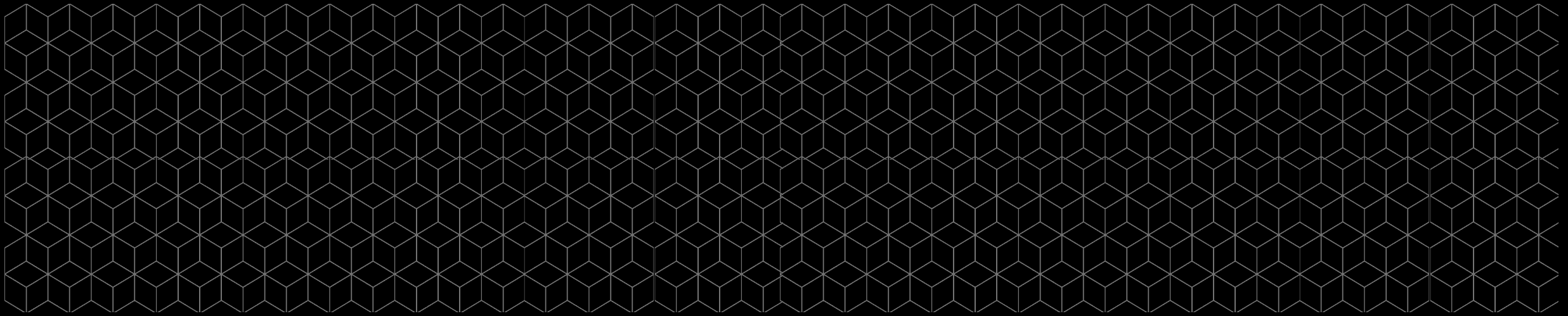
Silver approx.

Select a Color: ▼



Your Color: RGB: 194, 194, 194

Outer box: Your selected color, Inner box: Actual color of closest match.



Pseudo-elementos

Los pseudo-elementos se añaden a los selectores y permiten añadir estilos a una parte concreta del documento o elemento HTML.

<https://developer.mozilla.org/es/docs/Web/CSS/Pseudo-elements>

El pseudo-elemento `:first-line` permite seleccionar la primera línea de texto de un elemento.

Ejemplo: `div:first-line { color: red; }`

El pseudo-elemento `:first-letter` permite seleccionar la primera letra de la primera línea de texto de un elemento.

Ejemplo: `p:first-letter { text-transform: uppercase; }`

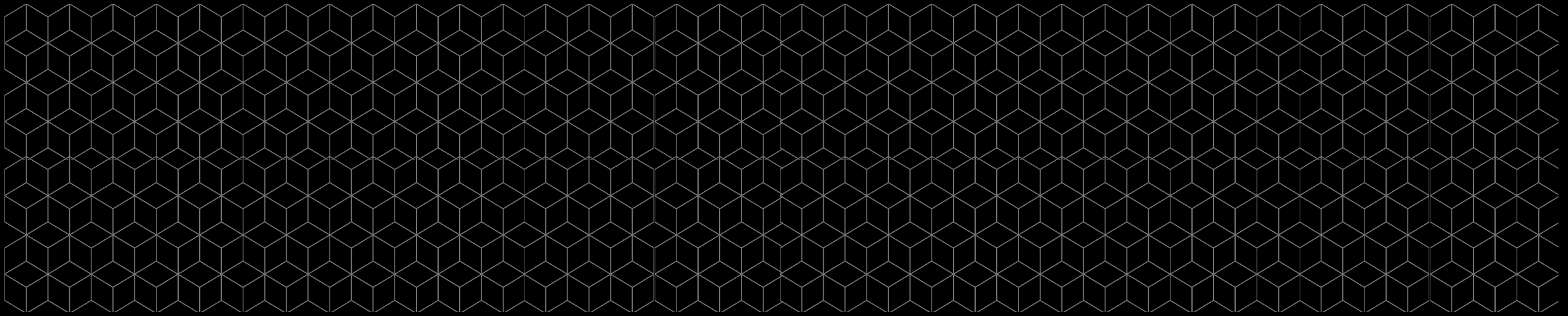
Los pseudo-elementos `:before` y `:after` se utilizan en combinación con la propiedad `content` de CSS para añadir contenidos antes o después del contenido original de un elemento.

El contenido insertado mediante los pseudo-elementos `:before` y `:after` se tiene en cuenta en los otros pseudo-elementos `:first-line` y `:first-letter`

Ejemplo:

`h1:before { content: "Capítulo - "; }`

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



Pseudo-classes

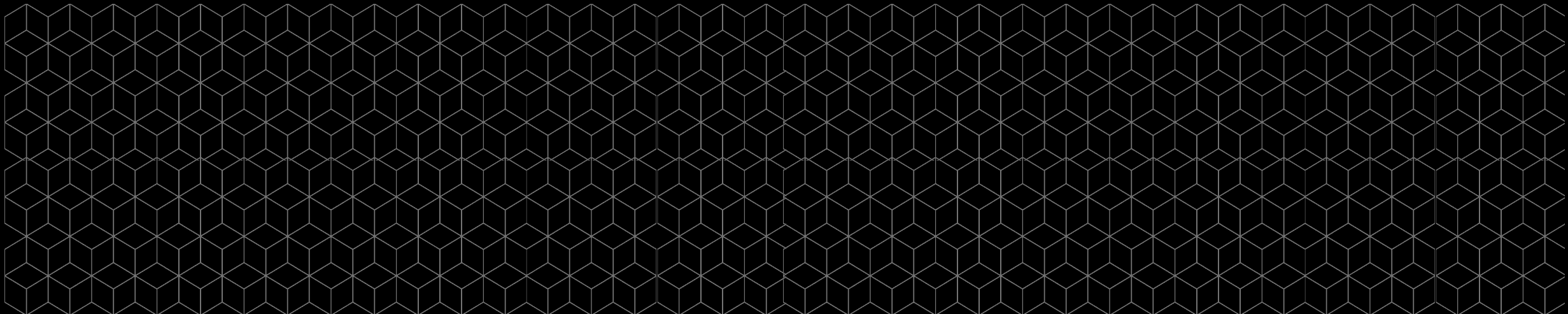
Una **pseudo clase CSS** es una palabra clave que se añade a los selectores y que especifica un estado especial del elemento seleccionado. Por ejemplo, `:hover` aplicará un estilo cuando el usuario haga hover sobre el elemento especificado por el selector.

<https://developer.mozilla.org/es/docs/Web/CSS/Pseudo-classes>

```
div:hover {  
  background-color: #F89B4D;  
}
```

Índice de las pseudo-clases estándar

- `:active`
- `:checked`
- `:default`
- `:dir()`
- `:disabled`
- `:empty`
- `:enabled`
- `:first`
- `:first-child`
- `:first-of-type`
- `:fullscreen`
- `:focus`
- `:hover`
- `:indeterminate`
- `:in-range`
- `:invalid`
- `:lang()`
- `:last-child`
- `:last-of-type`
- `:left`
- `:link`
- `:not()`
- `:nth-child()`
- `:nth-last-child()`
- `:nth-last-of-type()`
- `:nth-of-type()`
- `:only-child`
- `:only-of-type`
- `:optional`
- `:out-of-range`
- `:read-only`
- `:read-write`
- `:required`
- `:right`
- `:root`
- `:scope (en-US)`
- `:target`
- `:valid`
- `:visited`



Deuda Técnica

La deuda técnica es una metáfora que trata de explicar, que la falta de calidad en el código de un proyecto software genera una deuda que repercutirá en sobrecostos futuros, dichos sobrecostos están directamente relacionados con la tolerancia al cambio de nuestro proyecto.

¿QUÉ ES LA DEUDA TÉCNICA?



WARD CUNNINGHAM

- ✓ ES UNA METÁFORA DE LA DEUDA FINANCIERA
- ✓ REPERCUTE EN SOBRECOSTES FUTUROS
- ✓ SOFTWARE POCO TOLERANTE AL CAMBIO
- ✓ CONCEPTO INTRODUCIDO POR WARD CUNNINGHAM EN LA OOPSLA DE 1992



TIPOS DE DEUDA TÉCNICA



MARTIN FOWLER



Se puede dividir en cuatro tipos dependiendo de donde provenga.
Cuadrante de la deuda técnica

- ❑ **Imprudente y deliberado:** El desarrollador actúa conscientemente de forma imprudente y deliberada, esto suele derivar en un proyecto de mala calidad y poco tolerante al cambio.
- ❑ **Imprudente e inadvertido:** Es probable que este sea el tipo de deuda más peligrosa, ya que es generada por el desconocimiento y falta de experiencia, normalmente suele ser desarrollada por un programador con un perfil Junior o lo que es peor, un falso Senior.
- ❑ **Prudente y deliberado:** Dice W.C, que un poco de deuda técnica puede ser buena para acelerar el desarrollo de un proyecto, siempre y cuando esta se pague lo antes posible, el peligro viene con la deuda que no se paga, cuanto más tiempo se pasa con un código incorrecto, más se incrementan los intereses, este tipo de deuda se suele usar para crear un prototipo ya sea de un proyecto, una funcionalidad o incluso un MVP en el que apremie entregar valor cuanto antes.
- ❑ **Prudente e inadvertido:** Se da en la mayoría de los proyectos ya que está relacionado con los conocimientos adquiridos por el propio programador a lo largo del desarrollo del producto, llega un momento en el que el programador se da cuenta que pudo haber optado por un diseño de código mejor, llegado ese momento se debe evaluar si la deuda técnica se debe pagar o se debe posponer.

LAS DEUDAS SE PAGAN



- ✓ LA DEUDA TÉCNICA SE ACABA PAGANDO
- ✓ EVITAR EL DEFAULT EN NUESTRO PROYECTO
- ✓ ES IDEAL PAGAR LA DEUDA CUANTO ANTES
- ✓ SE PAGA A TRAVÉS DE LA REFACTORIZACIÓN

La deuda técnica tarde o temprano se acaba pagando, al igual que la deuda financiera sino se va amortizando a tiempo, los intereses se van acumulando en forma de código poco tolerante al cambio, incluso si se acumula demasiado va a llegar un momento en el que se vuelve inviable mantener el proyecto, los desarrolladores deben evitar entrar al default (suspensión de pago, término utilizado en finanzas, no se puede hacer frente al pago de la deuda) del proyecto, este escenario quiere decir que es tan costoso introducir cambios en el sistema que no vale la pena.

Incurrir en deuda técnica a menudo es inevitable, en este caso se debe ser consciente de las implicaciones y tratar de pagar la deuda técnica tan pronto como sea posible. ¿Cómo paga un desarrollador la deuda técnica? Generalmente:

- ☐ Refactorización
- ☐ Cambios en la arquitectura
- ☐ Rediseño en general

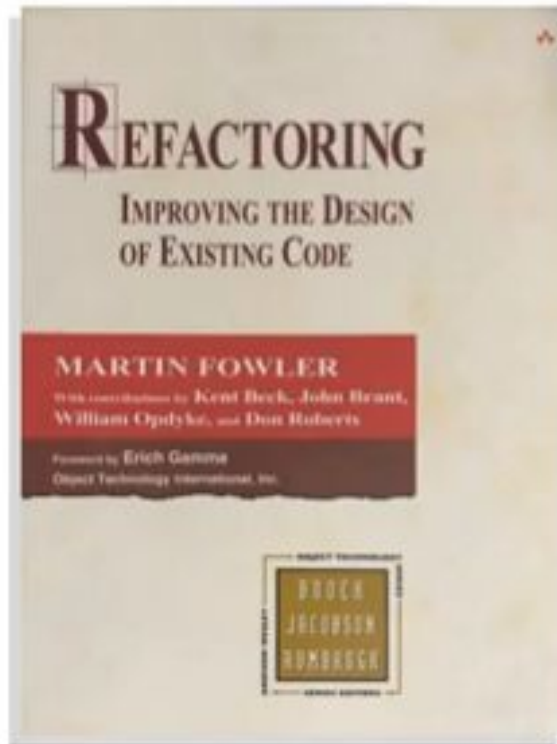
REFACTORING



"EL REFACTORING ES UN PROCESO QUE TIENE COMO OBJETIVO MEJORAR EL CÓDIGO DE UN PROYECTO SIN ALTERAR SU COMPORTAMIENTO PARA QUE DICHO CÓDIGO SEA MÁS EXPRESIVO Y TOLERANTE A CAMBIOS."

El desarrollo de software no es de naturaleza estática, el cliente solicita nuevas funcionalidades y se deben implementar, durante este proceso se van a encontrar módulos que no funcionan según lo esperado o funciones que han crecido demasiado, esto se debe mejorar y para esto utilizamos la refactorización de código.

¿CUÁNDO REFACTORIZAR?



- ✓ CUANDO VEAS CÓDIGO COMENTADO O EXCESO DE COMENTARIOS
- ✓ MALOS NOMBRES EN VARIABLES, FUNCIONES Y/O CLASES
- ✓ DUPLICIDAD EN EL CÓDIGO
- ✓ FUNCIONES QUE NO HACEN UNA SOLA COSA
- ✓ CLASES CON MÚLTIPLES RESPONSABILIDADES

¿CÓMO REFACTORIZAR?



- ✓ EL CÓDIGO DEBE TENER TESTS AUTOMÁTICOS
- ✓ NO INTENTES AÑADIR UNA FUNCIONALIDAD MIENTRAS REFACTORIZAS
- ✓ HACERLO EN PEQUEÑOS PASOS
- ✓ APÓYATE EN EL CONTROL DE VERSIONES

La refactorización implica la realización de muchos cambios pequeños y al final generan como resultado un cambio de mayor tamaño.

MEJOR PREVENIR QUE CURAR. LAS 4 REGLAS DEL DISEÑO SIMPLE

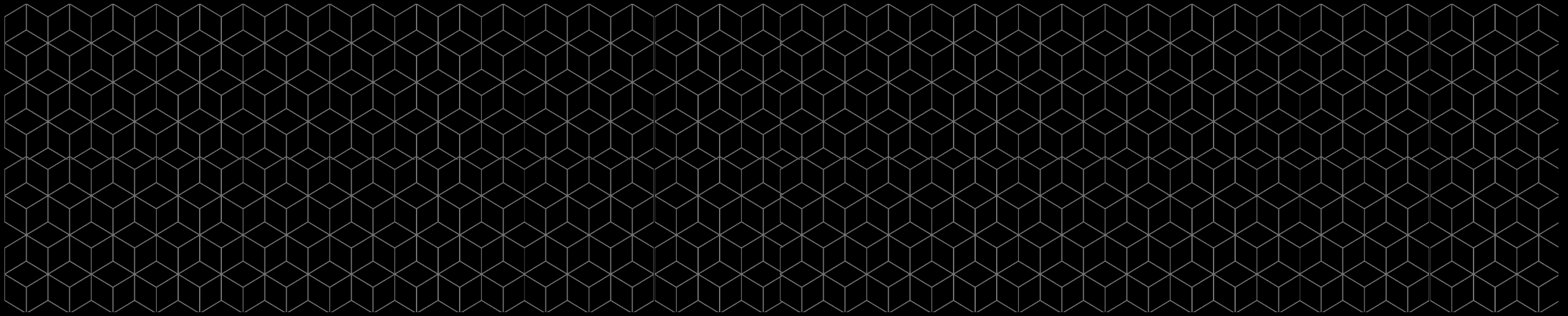


KENT BECK

- ✓ EL CÓDIGO PASA CORRECTAMENTE LAS PRUEBAS
- ✓ REVELA LA INTENCIÓN DEL DISEÑO
- ✓ EVITA LAS DUPLICIDADES (PRINCIPIO DRY)
- ✓ TIENE EL MENOR NÚMERO DE ELEMENTOS POSIBLES

El principio “No Te Repitas” (Don’t Repeat Yourself), es uno de los principios en el desarrollo de software cuyo principal objetivo es evitar la duplicación de código.

La mala calidad del software siempre la termina pagando o asumiendo alguien, ya sea el cliente, el proveedor con recursos o el propio desarrollador dedicando tiempo a refactorizar o malgastando tiempo sobre un sistema frágil. Un punto de partida para prevenir la deuda técnica, o por lo menos la más problemática es intentar valorar las cuatro reglas de diseño simple inventadas por Kent Beck.

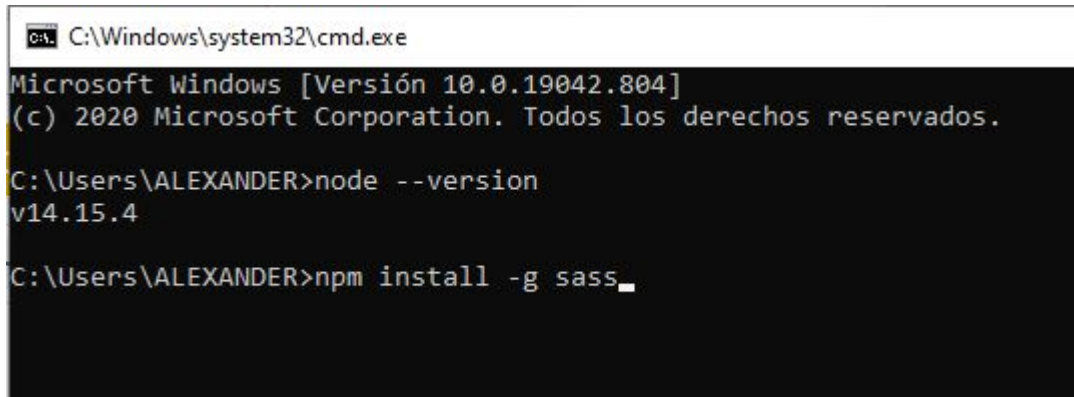
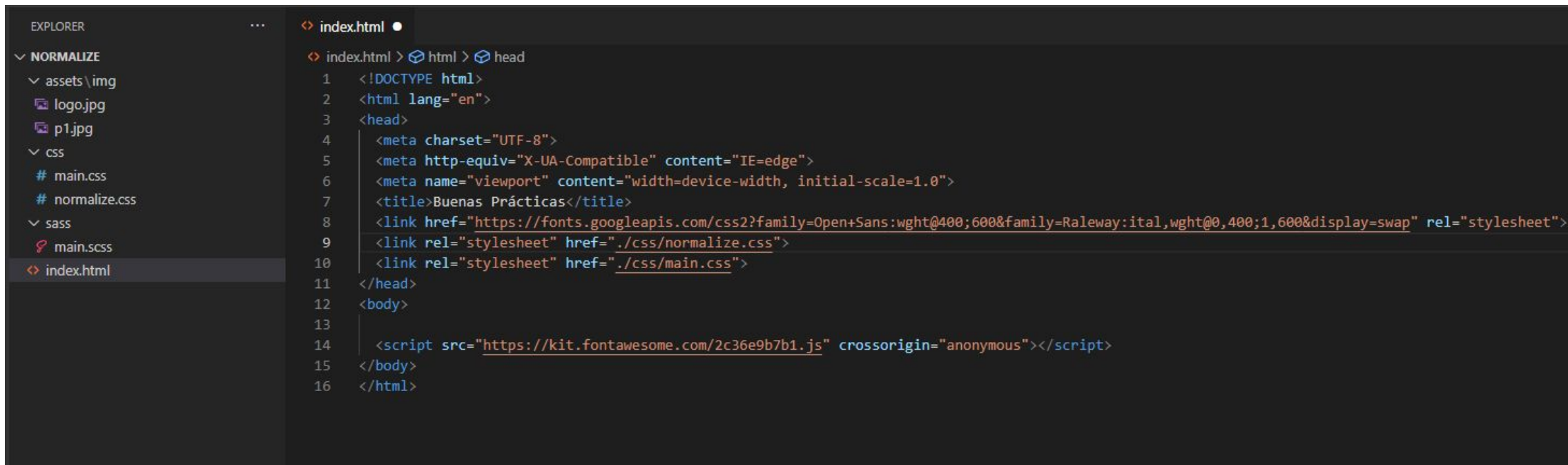


Ejercicio Paso a Paso

Paso 1:

Cree la estructura del proyecto y la estructura HTML5 en el index

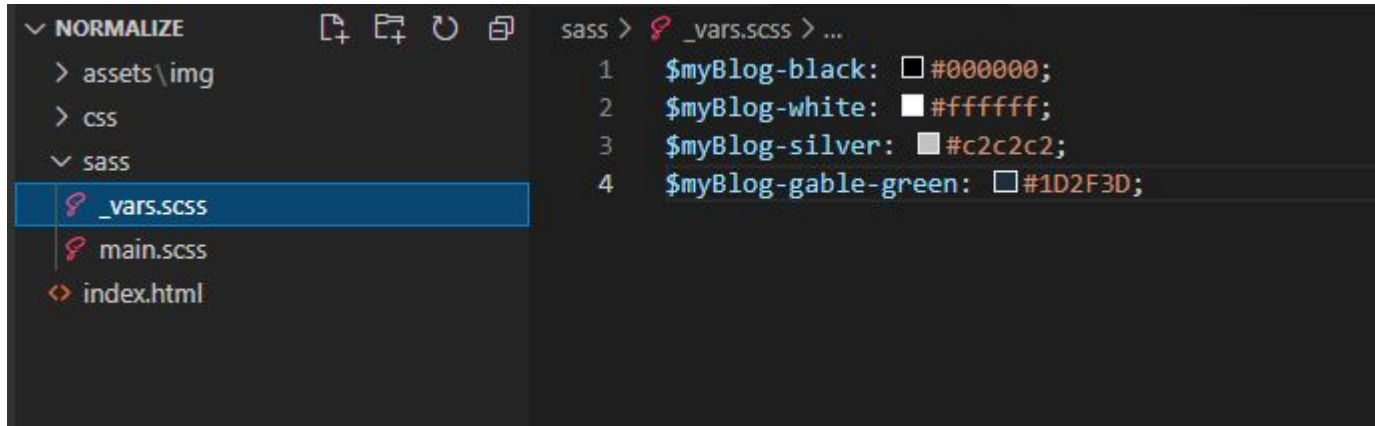
```
<link href="https://fonts.googleapis.com/css2?family=Open+Sans:wght@400;600&family=Raleway:ital,wght@0,400;1,600&display=swap" rel="stylesheet">  
<script src="https://kit.fontawesome.com/2c36e9b7b1.js" crossorigin="anonymous"></script>
```



Si no tiene instalado Sass, se instala de la siguiente manera

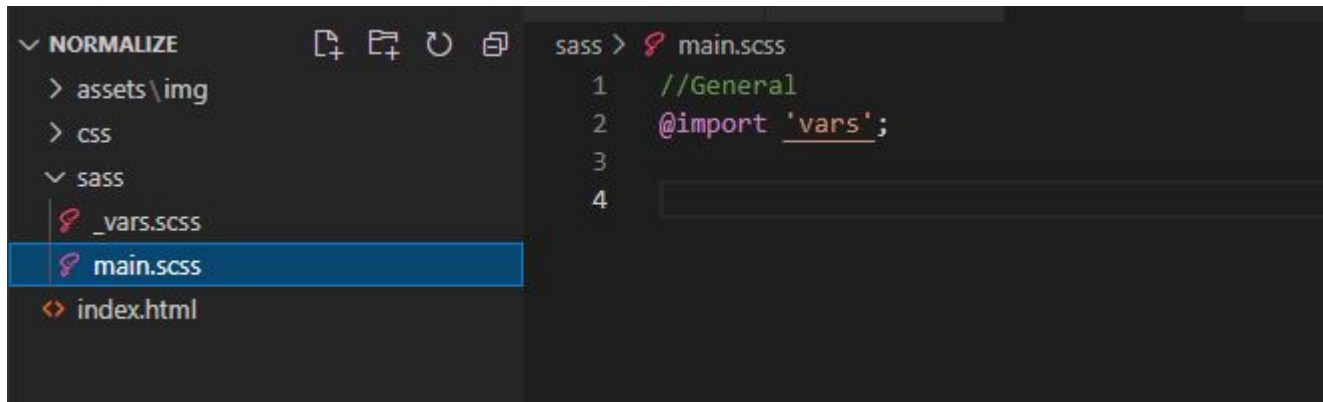
Paso 2:

Cree un archivo .scss denominado _vars.scss, en este archivo se guardan las variables que se van a utilizar en el sitio web, luego importe el archivo en el main.scss



This screenshot shows the Visual Studio Code editor with the file explorer on the left displaying a project structure under a 'sass' folder: 'assets\img', 'css', '_vars.scss' (selected), 'main.scss', and 'index.html'. The editor window shows the content of '_vars.scss' with four lines of SCSS variables, each preceded by a line number:

```
sass > _vars.scss > ...  
1 $myBlog-black: #000000;  
2 $myBlog-white: #ffffff;  
3 $myBlog-silver: #c2c2c2;  
4 $myBlog-gable-green: #1D2F3D;
```



This screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the same project structure, but now 'main.scss' is selected. The editor window shows the content of 'main.scss' with two lines of SCSS code, each preceded by a line number:

```
sass > main.scss  
1 //General  
2 @import 'vars';  
3  
4
```

Paso 3:

Cree los elementos HTML para el header

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Buenas Prácticas</title>
8      <link href="https://fonts.googleapis.com/css2?family=Open+Sans:wght@400;600&family=Raleway:ital,wght@0,400;1,600&display=swap" rel="stylesheet">
9      <link rel="stylesheet" href="./css/normalize.css">
10     <link rel="stylesheet" href="./css/main.css">
11 </head>
12 <body>
13     <header class="myBlog-header">
14         <nav class="myBlog-nav">
15             <a href="#" class="myBlog-link">Inicio</a>
16             <a href="#" class="myBlog-link">Servicios</a>
17             <a href="#" class="myBlog-link">Precios</a>
18             <a href="#" class="myBlog-link">Compañía</a>
19             <a href="#" class="myBlog-link">Soporte</a>
20         </nav>
21         <div class="myBlog-contentHeader">
22             
23         </div>
24     </header>
25
26     <script src="https://kit.fontawesome.com/2c36e9b7b1.js" crossorigin="anonymous"></script>
27 </body>
28 </html>
```

Paso 4:

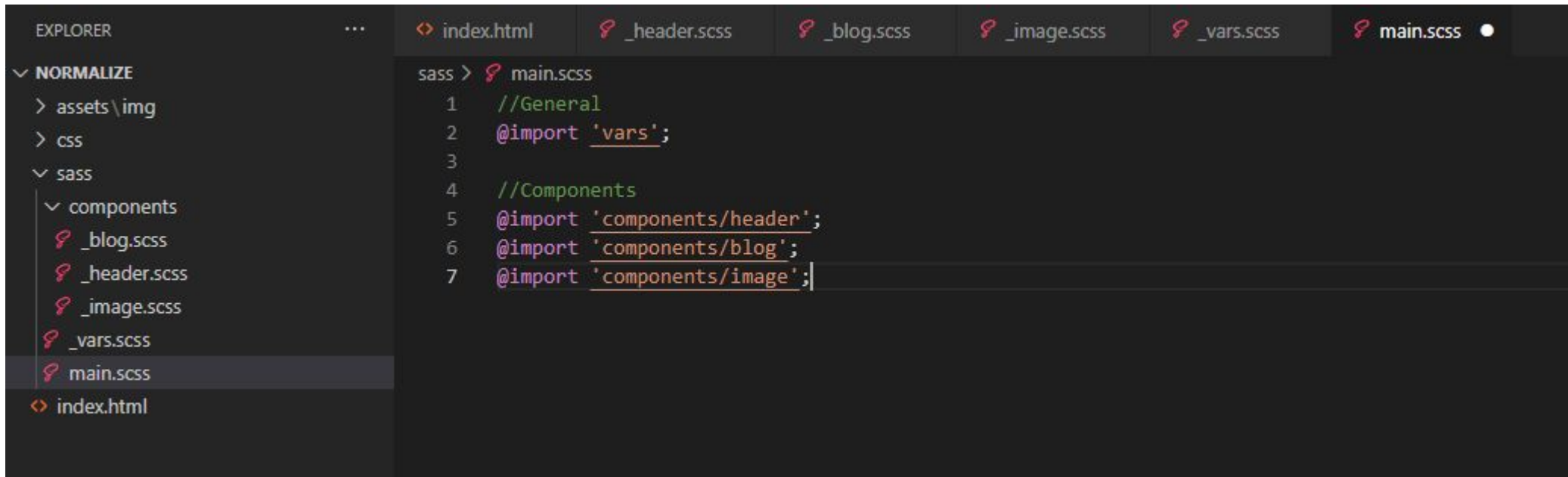
Cree los elementos HTML para el blog. copie el div de post cuatro veces

```
25 <section class="myBlog-section" id="myBlog">
26   <div class="myBlog-sectionContainer">
27     <aside class="myBlog-aside">
28       <h3 class="myBlog-asideTitle">Temas</h3>
29       <nav class="myBlog-asideNav">
30         <a href="#" class="myBlog-asideLink">Artículo #1</a>
31         <a href="#" class="myBlog-asideLink">Artículo #2</a>
32         <a href="#" class="myBlog-asideLink">Artículo #3</a>
33         <a href="#" class="myBlog-asideLink">Artículo #4</a>
34       </nav>
35     </aside>
36     <div class="myBlog-content">
37       <div class="myBlog-post">
38         <h1 class="myBlog-postTitle">Artículo #1</h1>
39         <p class="myBlog-postDate">Publicado el 16 de marzo de 2021</p>
40         <p class="myBlog-postDescription">
41           Lorem ipsum dolor sit amet consectetur adipisicing elit. Harum quos dolore fugit necessitatibus blanditiis consequatur ipsum reprehenderit
42           Iure placeat sunt saepe itaque. Velit officia perspiciatis a nobis commodi voluptatem sequi ea sed ipsum dicta enim, necessitatibus veniam
43         </p>
44       </div>
45       <div class="myBlog-post">
46         <h1 class="myBlog-postTitle">Artículo #2</h1>
47         <p class="myBlog-postDate">Publicado el 16 de marzo de 2021</p>
48         <p class="myBlog-postDescription">
49           Lorem ipsum dolor sit amet consectetur adipisicing elit. Harum quos dolore fugit necessitatibus blanditiis consequatur ipsum reprehenderit
50           Iure placeat sunt saepe itaque. Velit officia perspiciatis a nobis commodi voluptatem sequi ea sed ipsum dicta enim, necessitatibus veniam
51         </p>
52       </div>
53     </div>
54   </div>
55 </section>
56
```

Paso 5:

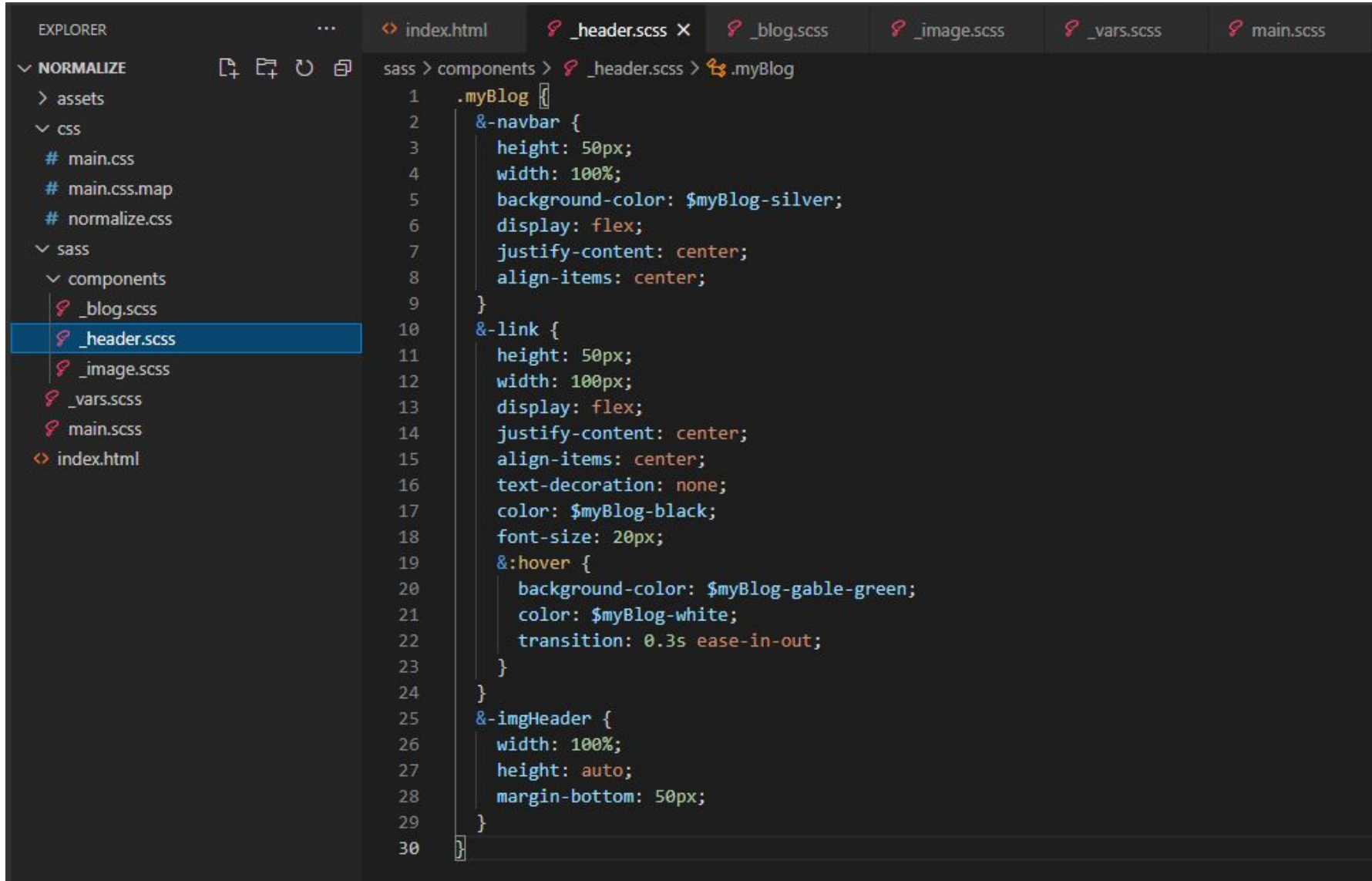
Cree los elementos HTML para el componente imagen.
Posterior a esto cree una carpeta components dentro de la carpeta sass y cree los archivos .scss relacionados a cada componente.

```
71 </section>
72 <div class="myBlog-image">
73   
74 </div>
75
76 <script src="https://kit.fontawesome.com/2c36e9b7b1.js" crossorigin="anonymous"></script>
77 </body>
78 </html>
```



Paso 6:

Cree los estilos para el componente _header



The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar displays a file tree with the following structure:

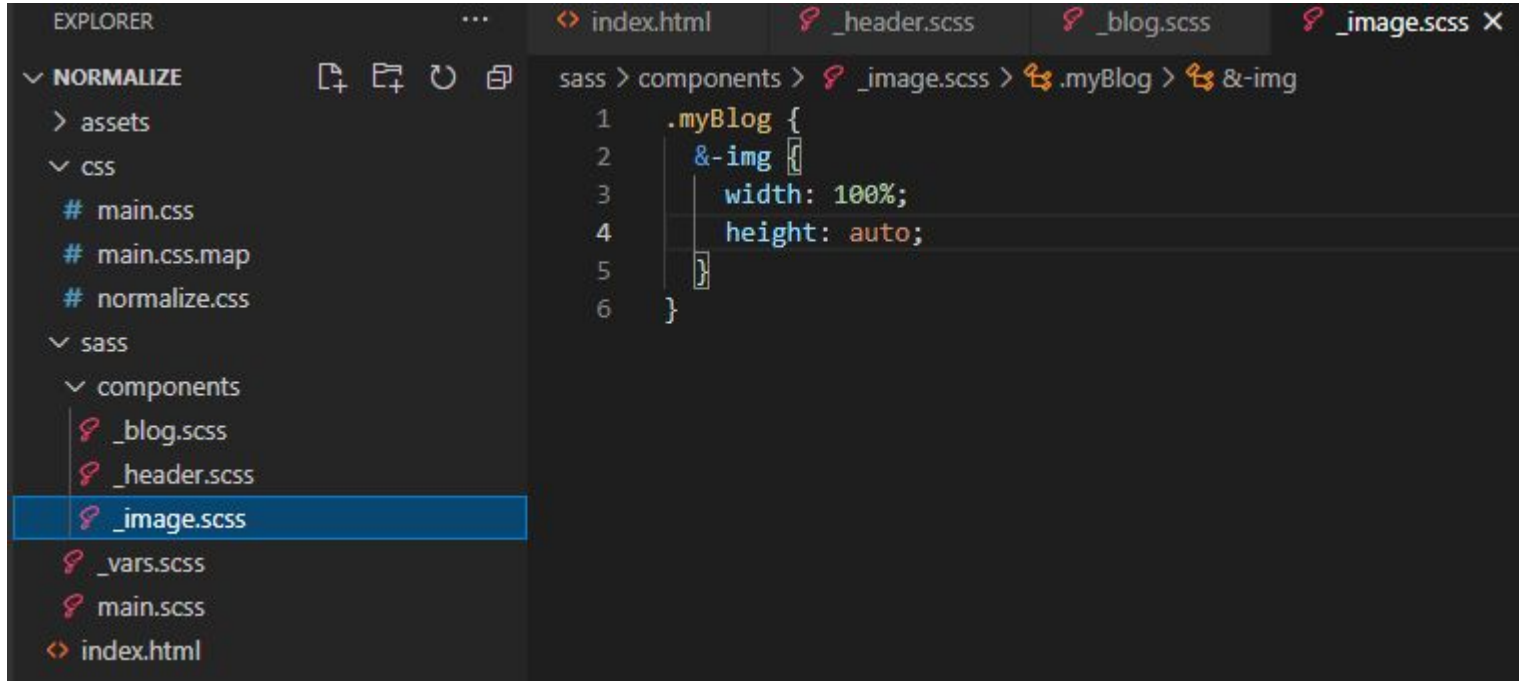
- EXPLORE
- ▼ NORMALIZE
 - > assets
 - ▼ css
 - # main.css
 - # main.css.map
 - # normalize.css
 - ▼ sass
 - ▼ components
 - _blog.scss
 - _header.scss**
 - _image.scss
 - _vars.scss
 - main.scss
 - <> index.html

The main editor area shows the content of `_header.scss` with the following SCSS code:

```
sass > components > _header.scss > .myBlog
1  .myBlog {
2    &-navbar {
3      height: 50px;
4      width: 100%;
5      background-color: $myBlog-silver;
6      display: flex;
7      justify-content: center;
8      align-items: center;
9    }
10   &-link {
11     height: 50px;
12     width: 100px;
13     display: flex;
14     justify-content: center;
15     align-items: center;
16     text-decoration: none;
17     color: $myBlog-black;
18     font-size: 20px;
19     &:hover {
20       background-color: $myBlog-gable-green;
21       color: $myBlog-white;
22       transition: 0.3s ease-in-out;
23     }
24   }
25   &-imgHeader {
26     width: 100%;
27     height: auto;
28     margin-bottom: 50px;
29   }
30 }
```

Paso 7:

Cree los estilos para el componente `_image`



```
EXPLORER
  NORMALIZE
    assets
    css
      # main.css
      # main.css.map
      # normalize.css
    sass
      components
        _blog.scss
        _header.scss
        _image.scss
      _vars.scss
      main.scss
      index.html

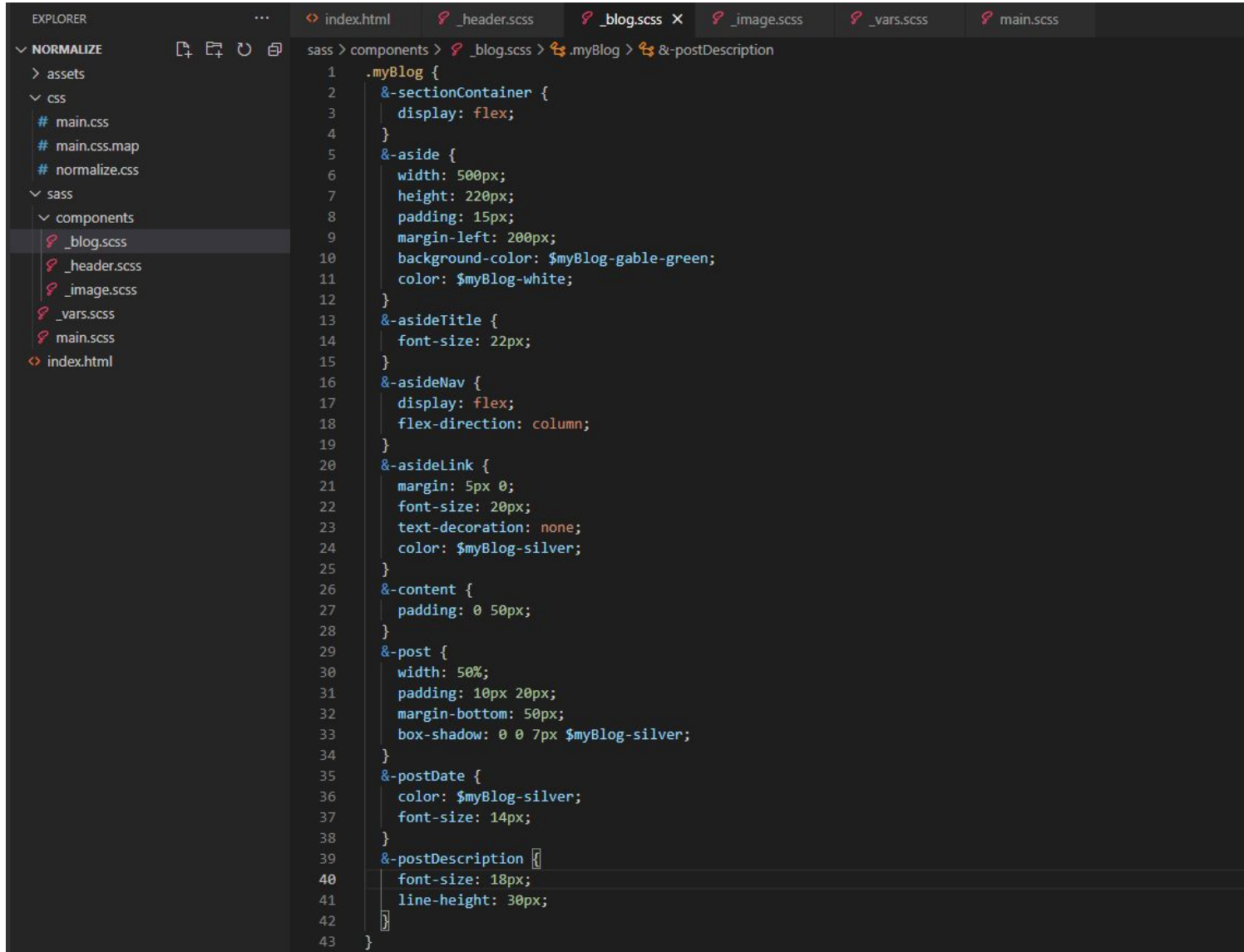
sass > components > _image.scss > .myBlog > &-img
1  .myBlog {
2    &-img {
3      width: 100%;
4      height: auto;
5    }
6  }
```

Recuerde utilizar el comando para compilar los estilos CSS

```
sass sass/main.scss css/main.css
```

Paso 8:

Cree los estilos para el componente _blog

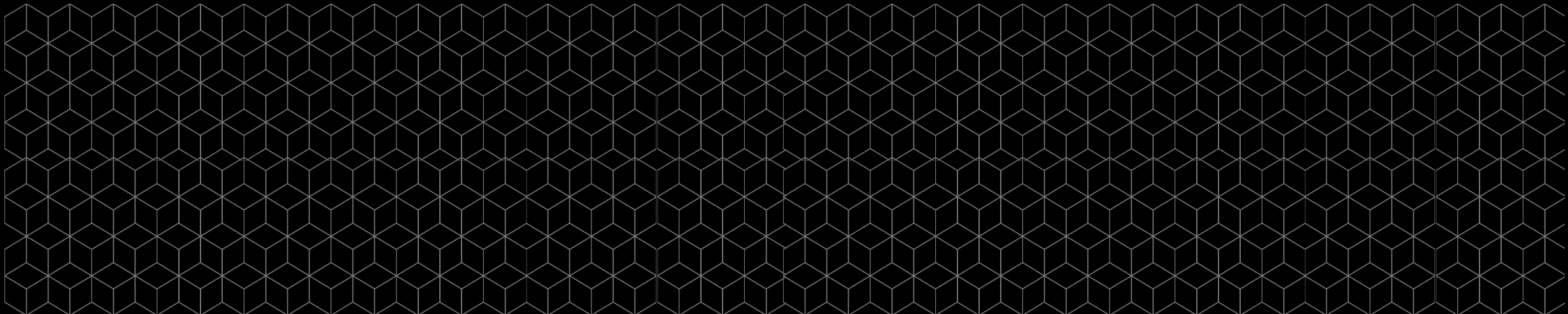


The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar displays a file tree with the following structure:

- NORMALIZE
 - assets
 - css
 - main.css
 - main.css.map
 - normalize.css
 - sass
 - components
 - _blog.scss**
 - _header.scss
 - _image.scss
 - _vars.scss
 - main.scss
- index.html

The main editor area shows the content of `_blog.scss` with the following SCSS code:

```
1 .myBlog {
2   &-sectionContainer {
3     display: flex;
4   }
5   &-aside {
6     width: 500px;
7     height: 220px;
8     padding: 15px;
9     margin-left: 200px;
10    background-color: $myBlog-gable-green;
11    color: $myBlog-white;
12  }
13  &-asideTitle {
14    font-size: 22px;
15  }
16  &-asideNav {
17    display: flex;
18    flex-direction: column;
19  }
20  &-asideLink {
21    margin: 5px 0;
22    font-size: 20px;
23    text-decoration: none;
24    color: $myBlog-silver;
25  }
26  &-content {
27    padding: 0 50px;
28  }
29  &-post {
30    width: 50%;
31    padding: 10px 20px;
32    margin-bottom: 50px;
33    box-shadow: 0 0 7px $myBlog-silver;
34  }
35  &-postDate {
36    color: $myBlog-silver;
37    font-size: 14px;
38  }
39  &-postDescription {
40    font-size: 18px;
41    line-height: 30px;
42  }
43 }
```

GRACIAS !

