Guía de Estilo CSS / Sass

> *Un enfoque más razonable para CSS y Sass*

Tabla de contenidos

Terminología

Reglas

Una "declaración de regla" es el nombre dado a un selector (o un grupo de selectores) acompañados de un grupo de propiedades. He aquí un ejemplo:

```
.listing {
  font-size: 18px;
  line-height: 1.2;
}
```

Selectores

En una declaración de regla, los "selectores" son los bits que determinan qué elementos en el árbol DOM tendrán el estilo dado por las propiedades definidas. Los selectores pueden coincidir con los elementos HTML, así como clases de elementos, ID, o cualquiera de sus atributos. He aquí algunos ejemplos de selectores:

```
.mi-clase-de-elemento {
  /* ... */
}

[aria-hidden] {
  /* ... */
}
```

Propiedades

Finalmente, las propiedades son las que dan el estilo a los elementos seleccionados de una declaración de regla. Las propiedades son pares *nombre-valor*, y una declaración de regla puede contener una o más declaraciones de propiedades. Así se ven las declaraciones de propiedades:

```
/* algún selector */ {
  background: #f1f1f1;
  color: #333;
}
```

CSS

Formato

- Usar "soft tabs" (2 espacios) de indentación
- Preferentemente usar guiones medios (-) en lugar de camelCase en nombres de clases.
    - Guiones bajos (_) y PascalCasing son válidos si está utilizando BEM (ver OOCSS y BEM abajo)
- No utilizar selectores ID
- Cuando se utilizan varios selectores en una declaración de regla, poner cada selector en una línea
- Poner un espacio antes de la llave de apertura { en declaraciones de regla
- En las propiedades, poner un espacio después, y no antes, del caracter : (dos puntos)
- Poner las llaves de cierre } de las declaraciones de regla en una nueva línea
- Poner líneas en blanco entre las declaraciones de reglas

Mal

```
.avatar{
    border-radius:50%;
    border:2px solid white; }
.no, .nope, .not_good {
    // ...
}
#lol-no {
  // ...
```

```
}
```

Bien

```
.avatar {
  border-radius: 50%;
  border: 2px solid white;
}

.un,
.selector,
.por-linea {
  // ...
}
```

Comentarios

- Preferentemente usar // (en Sass) para bloques de comentarios.
- Preferentemente escribir comentarios en una única línea para ello. Evitar los comentarios al final de una línea.
- Escribir comentarios detallados de código que no es auto-documentado:
    - Usos de z-index
    - Compatibilidad o hacks para navegadores específicos

OOCSS y BEM

Incentivamos el uso de una combinación de OOCSS y BEM por las siguientes razones:

- Ayuda a crear relaciones claras y estrictas entre CSS y HTML
- Ayuda a crear componentes reutilizables y que se pueden recomponer.
- Permite menos anidación y menor especificidad
- Ayuda a construir hojas de estilos escalables

OOCSS, o "CSS Orientado a Objetos" por sus siglas en Inglés, es un enfoque para escribir CSS que le permite pensar en sus hojas de estilo como una colección de "objetos": reutilizables, con snippets que pueden repetirse fácilmente y se pueden utilizar independientemente a través de un sitio web.

- OOCSS wiki de Nicole Sullivan
- Introduction to OOCSS de Smashing Magazine

BEM, o "Bloque-Elemento-Modificador", es una *convención de nomenclatura* para clases en HTML y CSS. Fue originalmente desarrollado por Yandex con grandes bases de código y escalabilidad en mente, y puede servir como un conjunto sólido de directrices para la aplicación de OOCSS.

- BEM 101 de CSS Trick
- Introduction to BEM de Harry Roberts

Se recomienda una variante de BEM con "bloques" en formato PascalCase, que funciona particularmente bien cuando se combina con componentes (ej. React). Guiones medios y bajos se utilizan para los modificadores e hijos.

Ejemplo

```
// ListingCard.jsx
function ListingCard() {
  return (
```

```
    <article class="ListingCard ListingCard--featured">

      <h1 class="ListingCard__title">Adorable 2BR in the sunny Mission</h1>

      <div class="ListingCard__content">
        <p>Vestibulum id ligula porta felis euismod semper.</p>
      </div>

    </article>
  );
}
```

```
/* ListingCard.css */
.ListingCard { }
.ListingCard--featured { }
.ListingCard__title { }
.ListingCard__content { }
```

- .ListingCard es el "bloque" y representa el componente de más alto nivel.
- .ListingCard__title es un "elemento" y representa un descendiente de .ListingCard que ayuda a componer el bloque en su conjunto.
- .ListingCard--featured es un "modificador" y representa un estado diferente o variación del bloque .ListingCard.

Selectores ID

Si bien es posible seleccionar elementos por ID en CSS, en general se debe considerar un anti-patrón. Los selectores ID introducen innecesariamente un alto nivel de especificidad a sus declaraciones de regla, y no son reutilizables.

Por más información sobre este tema, lea el siguiente artículo de CSS Wizardry sobre el control de la especificidad.

JavaScript hooks

Evitar la vinculación de la misma clase en tu CSS y JavaScript. Combinar los dos a menudo tiene consecuencias negativas, como mínimo, tiempo perdido durante la refactorización cuando un desarrollador debe hacer una referencia cruzada de cada clase que está cambiando, y peor aún, los dessarrolladores temen hacer cambios por miedo a romper la funcionalidad.

Recomendamos crear clases específicas para JavaScript para vincular con CSS, con el prefijo .js-:

```
<button class="btn btn-primary js-request-to-book">Request to Book</button>
```

Border

Utilizar 0 en lugar de none para especificar que un estilo no tiene borde.

Mal

```
.foo {
  border: none;
}
```

Bien

```
.foo {
```

```
  border: 0;
}
```

Sass

Sintaxis

- Utilizar la sintaxis .scss, nunca la sintaxis .sass original
- Ordenar el CSS regular y las declaraciones @include lógicamente (ver a continuación)

Orden de las declaraciones de propiedades

1. Declaraciones de Propiedades

   Listar todas las declaraciones de propiedades estándar, todo lo que no sea un @include o un selector anidado.

   ```
   .btn-green {
     background: green;
     font-weight: bold;
     // ...
   }
   ```

2. Declaraciones @include

   Agrupar @includes al final hace que sea más fácil de leer el selector entero.

   ```
   .btn-green {
     background: green;
     font-weight: bold;
     @include transition(background 0.5s ease);
     // ...
   }
   ```

3. Selectores anidados

   Los selectores anidados, *si es necesario*, van al final, y nada va después de ellos. Añadir espacio en blanco entre las declaraciones y selectores de reglas anidadas, así como entre los selectores adyacentes anidados. Aplicar las mismas directrices anteriores a los selectores anidados.

   ```
   .btn {
     background: green;
     font-weight: bold;
     @include transition(background 0.5s ease);

     .icon {
       margin-right: 10px;
     }
   }
   ```

Variables

Preferentemente utilizar nombres de variable con guiones medios (ej. $mi-variable) en lugar de escritura de camello (camelCase) o guiones bajos (snake_case). Es aceptable utilizar como prefijo para nombres de variables que están pensados para ser utilizados solo dentro del mismo archivo con guión bajo (ej., $_mi-variable).

Mixins

Se deben utilizar Mixins para no repetir código (principio DRY), agregar claridad o abstraer de complejidad de la misma manera que las funciones. Los Mixins que no aceptan argumentos pueden ser útiles para ello, pero notar que si no está comprimiendo su carga (ej. gzip), esto puede contribuir a la duplicación innecesaria de código en los estilos resultantes.

Extend

@extend debe evitarse ya que tiene un comportamiento poco intuitivo y potencialmente peligroso, específicamente cuando se utiliza con selectores anidados. Incluso extender los selectores placeholder de nivel superior puede causar problemas si el orden de los selectores termina cambiando más tarde (ej. si se encuentran en otros archivos y el orden de los archivos cambia). Gzipping debe manejar la mayor parte de los ahorros que se habría obtenido mediante el uso de @extend, y se puede evitar la repetición de código [1] en las hojas de estilos muy bien con mixins.

[1] *Se utiliza el verbo "DRY up" haciendo referencia al principio DRY (Don't Repeat Yourself).*

Anidación

¡No anidar selectores más de 3 niveles de profundidad!

```
.page-container {
  .content {
    .profile {
      // PARAR!
    }
  }
}
```

Cuando los selectores se vuelven muy largos, probablemente se está escribiendo CSS que es:

- Fuertemente acoplado al HTML (frágil) —*O*—
- Excesivamente específico (poderoso) —*O*—
- No reutilizable

Nuevamente: **¡nunca anidar selectores ID!**

Si tiene que usar un selector ID en primer lugar (debería intentar no hacerlo), nunca deberían anidarse. Si se encuentra haciendo esto, neceista revisar su marcado HTML, o averiguar por qué necesita tanta especificidad. Si está escribiendo HTML y CSS bien formado, **nunca** debería hacer esto.