CSS Coding Standards

Consenso acerca de nuestros estándares de codificación para:

- Sintaxis y formato de CSS.
- Convenciones para nombrar.
- Arquitectura de nuestro CSS, incluyendo objetivos, errores a evitar y buenas prácticas.

Tabla de Contenidos

Sintaxis y formato

A manera general queremos:

- Indentar con **2** espacios, no tabs.
- Uso significativo de espacio en blanco.
- Uso de minúsculas y nombres en inglés para selectores por clase.
- Cambio de palabra separado por un guión (-), ejem. `.menu-item`
- Columnas con ancho no mayor a 80 caracteres; configure su IDE o editor de texto.
- Cuide el orden y especificidad, evite darle *estilo* a IDs y sobrecalificar selectores.
- Evite el uso del * wildcard selector.
- Como regla general, evite anidación innecesaria, y nunca anidar más de tres niveles. Si no puede evitarlo, de un paso atrás y reconsidere su estrategia.
- Evite anidar selectores: selectores descendientes son los selectores más lentos.
- Evite especificar unidades cuando el valor es 0 (cero), ejemplo,
  - `margin: 0;` en lugar de `margin: 0px;`
  - `border: 0;` en lugar de `border: none;`
  - `background-position: 0 0;` en lugar de `background-position: 0% 0%;`
- Siempre que aplique, utilice shorthand, `margin: 20px 0;` en lugar de `margin: 20px 0 20px 0;`

- Pero evite utilziar shorthand cuando es innecesario, `margin-bottom: 20px;` en lugar de `margin: 0 0 20px;`
- Use shorthand para valores hex cuando sea posible, ejemplo, `#fff` en lugar de `#ffffff`
- Use minúsculas para todos los valores hexadecimales, ejem., `#fff` .
- Todas las declaraciones deben terminar con un punto y coma (;).
- Propiedades que tienen valores separados por coma, deben incluir un espacio despues de cada coma (ejem. rgb(0, 0, 0)).
- Todo archivos de extensión **.css** debe estar incluido en el folder /css del proyecto.

Anatomía de una regla:

```
selector {
    propiedad: valor;
    [<--declaración--->]
}
[<--Regla o ruleset--->]
```

Convenciones para nombrar

- Principios
- Namespaces
- Abreviaciones

Principios:

Cualquier nombre que se utiliza directamente en el HTML (class/id) deben tener en cuenta estos principios:

- **Claridad** - comportamiento esperado / estilo debe ser inmediatamente obvio.
- **Semántica** - lo que un objeto es importa más que como se ve.
- **Genérico** - el nombre debe ser aplicar para la mayoría de los sitios. Nombres demasiado específicos reducen el número de casos de uso o causa que clases semánticas sean utilizados en una forma no semántica.
- **Brevedad** - cada byte cuenta, así que mantenga nombres tan cortos como sea posible, pero siempre y cuando sea necesario. Nombres más de 5-7 caracteres deben ser abreviados.

Siempre usamos minúscula, y el cambio de palabra separado por guión (-). Esto aplica para: selectores por clase ó id:

```
// Mal
.main_top {
    color: #FFC20E;
}

// Bien
.main-top {
    color: #ffc20e;
}
```

Namespaces:

- **js-** cuando seleccionamos elementos en el DOM con Javascript lo ideal es hacerlo via: ID's (preferiblemente) y, cuando son multiples elementos, usamos clasess con el prefijo js-, por ejemplo `js-submenu` . Estas clases **nunca** deben ser utilizadas para *styling*.
- Usamos BEM como convencion para nombrar.

Abreviaciones:

Cada byte cuenta, así que, mantenga nombres tan cortos como sea posible. Como regla general, nombres de más de 5-7 caracteres deben ser abreviados, sin embargo, no sacrificamos la **claridad** por brevedad. Algunas ideas de abreviaciones:

- configuration => config
- introduction => intro

- subcategory => subcat
- category => cat

Orden y especificidad

Las reglas se deben escribir de tal manera que, utilicemos la Cascada y la Herencia, y siempre tratar de que la Especificidad este a un nivel similar (en la medida de lo posible).

Especificidad

- Escribir los bloques de reglas en un orden específico: reset, third party, elementos, patrones, objetos, componentes, etc **No** escribir las reglas según el orden en que los elementos se muestran en la página.
- En la medida de lo posible, las reglas subsecuentes deben heredar, en la medida de lo posible, no sobreescribir.
- Selector por Etiqueta - sólo darle *estilo* a etiquetas globales.
- Selector por Clase - siempre es preferible darle *estilo* a selectores de tipo clase (.nombre-selector).
- IDs - evitar darle *estilo* a IDs. Incluso si el ID ya existe en la página, es mejor añadir una clase y darle *estilo*.
- Evitar sobrecalificar selectores, esto incrementa la especificidad, limita la reutilización y son menos eficientes (más trabajo para el browser)
- Evitar anidar innecesariamente. `.header .menu .item a {}`
- Evitar *encadenar* selectores. `.nav-item.expanded`

```
// Evitar sobrecalificar selectores

// Mal
p.intro {

}

// Bien
.intro {

}


// Evitar anidar innecesariamente

// Mal
.slide .panel-slide {

}

// Bien
.panel-slide {

}


// Evitar encadenar selectores

// Mal
.message.error{

}

// Bien
.msj-error {

}
```

Orden

Ordenar de forma alfanumérica, a excepción de donde se puede romper la Cascada.

- En la hoja de estilos: pseudo-selectores (:after, :active, :before, :hover, :link, :visited), etiquetas antes que clases y antes que ids - orden natural de la especificidad.

Etiquetas y clases:

```
// Mal
.product-list {

}

p {

}

a:hover {

}

ul {

}

// Bien
a:hover {

}

p {

}

ul {

}

.product-list {

}
```

Uso de whitespace

Indentación

Indentamos con **2 espacios**, no tabs. Puede configurar su editor para que lo realice de manera automática. Los espacios son la única forma de garantizar que el código *renderea* igual en el entorno de cualquier persona.

```
// Mal
selector {
    propiedad: valor;
}

// Bien
selector {
  propiedad: valor;
}
```

espacios

Una declaración (propiedad y valor) por línea.

```
// Mal
```

```
selector {
  propiedad: valor; propiedad: valor; propiedad: valor;
}

// Bien
selector {
  propiedad: valor;
  propiedad: valor;
  propiedad: valor;
}
```

Un espacio antes **{** (*bracket que abre*).

```
// Mal
selector{

/* Bien */
selector {
```

Cierre en su propia línea **}** (*bracket que cierra*).

```
// Mal
selector { }

// Bien
selector {

}
```

Espacio después de **:** (dos puntos), **;** (punto y coma) y **,** (comas).

```
// Mal
selector {
    color:red;font-family:Helvetica,Arial,sans-serif;
}

// Bien
selector {
  color: red; font-family: Helvetica, Arial, sans-serif;
}
```

nueva línea

Después de cada regla incluimos nueva línea

```
// Mal
selector {
    propiedad: valor;
}
selector {
    propiedad: valor;
}

// Bien
selector {
  propiedad: valor;
}

selector {
  propiedad: valor;
}
```

Cuando agrupamos selectores, cada selector en su propia línea

```
// Mal
selector, selector2, selector3 {
    propiedad: valor;
}

// Bien
selector,
selector2,
selector3 {
  propiedad: valor;
}
```

strings comillas dobles (" ")

```
// Mal
selector {
    font-family: 'Goudy Bookletter 1911', sans-serif;
}

// Bien
selector {
  font-family: "Goudy Bookletter 1911", sans-serif;
}
```

Comentarios

El código es escrito y mantenido por personas. Asegúrese de que su código sea descriptivo, este bien comentado, y sea accesible por otros. Un código bien comentado transmite contexto o propósito. No se limite a reiterar un nombre de componente o clase.

Asegúrese de escribir en oraciones completas para comentarios más grandes y frases concisas para las notas generales.

Siempre dejamos un espacio antes del comentario.

```
// Mal
selector {
    width: 100% !important; /* Sobreescribo */
}

// Bien
selector {
  width: 100% !important; /* Comentario explicando porque el uso de !import */
}
```

Debido a que, vamos a eliminar los comentarios de las hojas de estilo en la versión de producción, no se preocupe por la longitud de lo que escribe. Al mismo tiempo, nuestro objetivo debe ser: código debe autoexplicarse, las classes nos sirven para eso, y mantener al mínimo la cantidad de comentarios, para que el propósito/uso de un documento sea claro y que el mantenimiento no se convierta en algo tedioso.

Fuentes

- CSS Style Guides - CSS-Tricks
- Writing efficient CSS - MDN
- Code Guide by @mdo
- BEM