Estándares de codificación HTML, CSS, Javascript

Carlos Garreta, Ricardo Rolín, Yliana Otero

Estándares de Codificación

Estándares de Codificación

Los estándares de codificación son reglas o técnicas para crear un código más limpio, legible y mantenible.

Además, ofrecen una forma uniforme para que los desarrolladores creen código altamente funcional. Los estándares de codificación garantizan que todos los desarrolladores sigan pautas específicas.

No son opiniones personales, sino reglas concretas que determinan el estilo, los procedimientos y los métodos de programación del código.

W3C

El consorcio de la World Wide Web, o W3C, define desde sus comienzos los estándares que hoy utilizamos para el desarrollo web. Puntos como accesibilidad, mantenimiento, escritura de código, y herramientas de validación se tratan allí y se utilizan ampliamente por toda la industria.

Recursos:

- Sección para desarrolladores – https://www.w3.org/developers/
- Cursos gratuitos de desarrollo web – https://www.edx.org/school/w3cx#programs

¿Por qué usarlos?

Garantizan consistencia y calidad en la creación de código en distintos lenguajes

Facilitan la colaboración en equipos de desarrollo

Promueven un código más limpio y eficiente

Facilita la comprensión y el aprendizaje del código

Ahorra tiempo y esfuerzo en el mantenimiento del código

Google Style Guides

Cada empresa define el estándar de codificación a seguir en el código que produce.

Google ha definido estos estándares o reglas a través de las Google Style Guides. Los temas tratados en estas guías no solo abarcan cuestiones estéticas, sino también otros tipos de convenciones de programación.

Hay diversas guias para distintos tipos de lenguajes. Por ejemplo, para JavaScript existe la Google JavaScript Style Guide, y se puede decir que un archivo JavaScript está en el estilo Google si y sólo si cumple con las reglas descritas en dicha guía.

HTML

HTML

Es el lenguaje de marcado estándar utilizado para crear y diseñar páginas web

Fue creado por Tim Berners-Lee en 1993 para el proyecto World Wide Web en el CERN y se ha convertido en el lenguaje fundamental para la construcción de sitios web en Internet

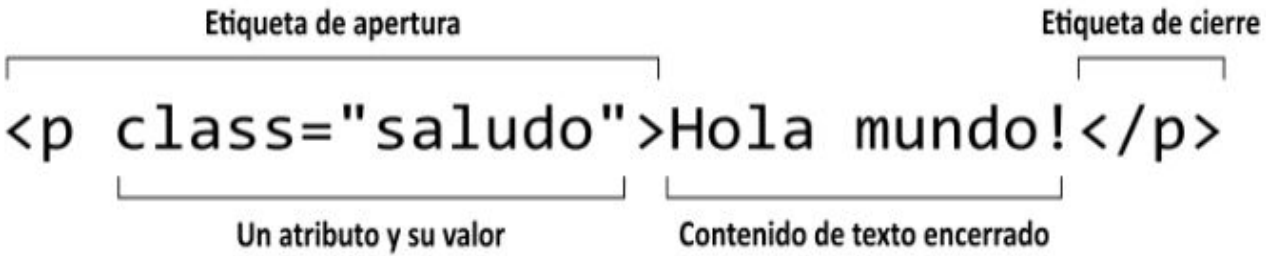
El contenido de una página web se representa a partir de una estructura jerárquica de etiquetas (tags), que los navegadores web son capaces de interpretar y representar

HTML se complementa con CSS para el diseño y estilo de los elementos de la página web, y con JavaScript para el comportamiento e interactividad de las mismas

Documentación HTML Mozilla

W3Schools HTML

Etiquetas



Siempre se debe utilizar minúsculas para: nombres, atributos, valores de atributos (si es posible)

Cada etiqueta de apertura debe tener su etiqueta de cierre correspondiente

Utilizar comillas dobles para los valores de atributos

Escribir cada bloque en una nueva línea

Estructura

```
<!DOCTYPE html>
<html lang="es">
<meta charset="utf-8">
<head>
  <title>Titulo de la página</title>
</head>
<body>
  <div>
    <h1>Primer encabezado</h1>
    <p>Primer párrafo</p>
  </div>
</body>
</html>
```

Utilizar siempre html5

Utilizar el formato de codificación utf-8

Utilizar siempre las etiquetas principales de un documento html (html, head, title, body)

Tags Opcionales

Enlaces y usos de archivos externos

```html
<!DOCTYPE html>
<html>
  <meta charset="utf-8">
  <head>
    <title>Titulo de la página</title>
    <link href="https://www.google.com/css/maia.css"
      rel="stylesheet" />
    <link
      rel="apple-touch-icon" sizes="114x114"
      href="apple-icon-114.png" type="image/png" />
  </head>
  <body>
    <div>
      <!-- Contenido -->
    </div>

    <script src="javascript.js"></script>
  </body>
</html>
```

Mantener separados el contenido, el estilo y el comportamiento de una página

Cargar hojas de estilo en el head

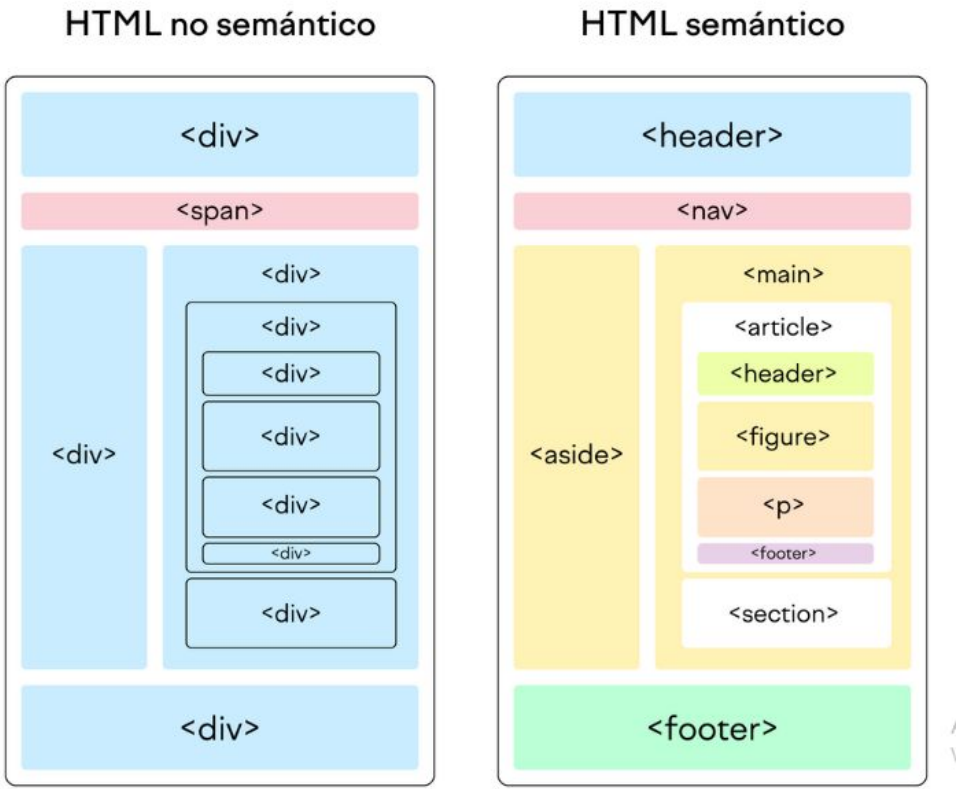
Cargar scripts al final del body

En html5, cuando se trata de una hoja de estilos .css o de un script .js, no es necesario colocar el atributo type

Al utilizar direcciones web, utilizar el protocolo https siempre que sea posible

Semántica

Utilizar las etiquetas HTML acorde al tipo de contenido que se quiera mostrar



Identación y espaciado

```
<ul>
  <li>
    <img src="mdn-logo-sm.png" alt="MDN_logo"/>
  </li>
  <li>Segundo elemento</li>
</ul>

<button mat-icon-button color="primary"
    class="menu-button" (click)="openMenu()">
  <mat-icon>menu</mat-icon>
</button>
```

Indentación de dos espacios (no utilizar tabs)

Se puede utilizar una línea en blanco para separar bloques de código

Evitar que las líneas de código ocupen demasiado espacio horizontal (ej: 80 caracteres)

Line-Wrapping

Comentarios

```
<div>
  <!-- Comentario de linea -->

  <!--
    Comentario
    de múltiples
    lineas
  -->

  <!-- TODO: tarea por hacer -->
</div>
```

Utilizar comentarios solo cuando se consideren estrictamente necesarios para explicar una sección de código

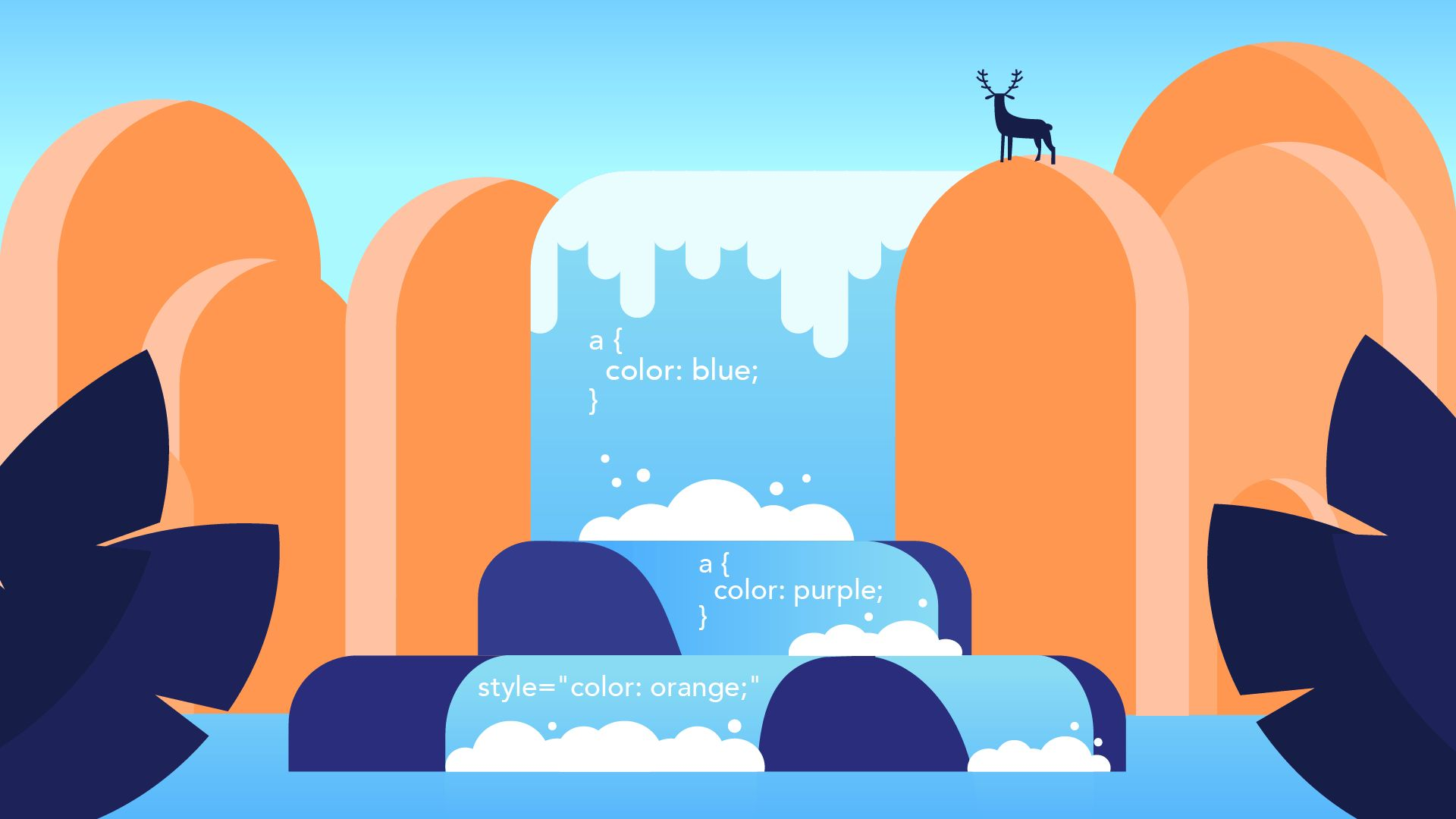
Evitar comentarios obvios

Utilizar código explicativo

CSS

Cascading Style Sheets

Tal como HTML, CSS (Cascading Style Sheets) u Hojas de estilo en cascada en español, no es realmente un lenguaje de programación. Es un lenguaje de hojas de estilo, es decir, te permite aplicar estilos de manera selectiva a elementos en documentos HTML.


a {
color: blue;
}
a {
color: purple;
}
style="color: orange;"

CSS resolvió un gran problema

En los inicios de HTML, su propósito único era darle estructura a las páginas web. Cuando HTML introdujo la capacidad de darle estilos a los elementos, surgió un gran problema de mantenimiento, ya que todas estas características se tenían que aplicar a cada uno de los elementos de la página y no se podían englobar lógicamente. CSS se creó para suplir esta demanda de los desarrolladores.

¿Cómo se usa?

En un archivo a parte, de extensión .css, se agregan las clases de estilos con sus atributos. Desde el HTML, se importa dicho archivo de estilos y en los elementos HTML se aplicarán las distintas clases definidas en el .css.

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="styles.css">
  </head>
 <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
 </body>
</html>
```

```css
body {
  background-color: powderblue;
}
h1 {
  color: blue;
}
p {
  color: red;
}
```

Archivos: Nomenclatura y encoding

Tres pautas que generalmente se extienden a todos los archivos de contenido de un proyecto web.

1. El encoding debe ser UTF-8.
2. Los nombres de archivo van en minúscula. **Ejemplo: *estilo.css***
3. Los espacios se reemplazan por guiones. **Ejemplo: *estilo-custom.css***

Recomendaciones de estilo generales

- Indentar con dos espacios. Los tabs pueden traer problemas con distintos entornos de desarrollo y repositorios.
- Usar punto y coma al finalizar cada instrucción de atributos.
- Salto de línea entre reglas de estilo.
- Llave de apertura en la misma línea del selector, separada por un espacio.
- Un espacio luego de los dos puntos al asignar el valor a un atributo.
- Utilizar cita simple (‘ ’) en vez de doble (“ ”) para los valores de los atributos.
- No usar citas en las URLs de las importaciones.

Ejemplo: Recomendaciones generales

```css
/* Recomendado */
@import url(https://www.google.com/css/maia.css);

html {
  font-family: 'open sans', arial, sans-serif;
}

head {
  background: #fff;
}

body {
  margin: auto;
  width: 50%;
}
```

Nomenclatura de clases

- Usar nombres específicos que indiquen sobre qué componente se tratan, o nombres genéricos.
- En caso de nombres específicos, usar nombres que reflejen el propósito de la regla.
- En general ser los específicos son preferidos ya que se entienden fácilmente.
- Los nombres genéricos se pueden ver como “helpers” los cuales se utilizarán en múltiples elementos.

Ejemplo: Nomenclatura de clases

```
/* Recomendados: Específicos */
.gallery {}
.login {}
.video {}

/* Recomendados: Genéricos */
.aux {}
.alt {}
```

Selectores de clases y tipos

Se debe evitar utilizar selectores de tipo como parte de los nombres de clase. Esto resulta ser más claro y deriva en una mejor performance.

```
/* No recomendado */
ul.example {}
div.error {}
```

```
/* Recomendado */
.example {}
.error {}
```

Estilo para el nombre de clases

Se recomienda usar el nombre más corto posible siempre que describa claramente de qué se trata.

```
/* No recomendado */
.navigation {}
.atr {}
```

```
/* Recomendado */
.nav {}
.author {}
```

Delimitadores de nombres de clase

No se debería concatenar palabras y abreviaciones en los selectores con otra cosa que no sea un guión.

```
/*
No recomendado: No se separan las palabras
"demo" e "image" con  un guión
*/
.demoimage {}

/* No recomendado: Se usa un guión bajo en vez del guión. */
.error_status {}
```

```
/* Recomendado */
.video-id {}
.ads-sample {}
```

Selectores por ID

En general, es una mala práctica aplicar estilos específicamente a un elemento por su identificador. Los selectores por clase deberían ser la forma preferida.

```
/* No recomendado */
#example {}
```

```
/* Recomendado */
.example {}
```

Abreviaturas de propiedades

Se recomienda utilizarlas, cuando sea posible, para compactar la cantidad de líneas escritas en el documento de estilos. Las abreviaturas aportan a la eficiencia en lectura y mantenimiento.

Ejemplo: Abreviaturas de propiedades

```css
/* No recomendado */
border-top-style: none;
font-family: palatino, georgia, serif;
font-size: 100%;
line-height: 1.6;
padding-bottom: 2em;
padding-left: lem;
padding-right: lem;
padding-top: 0;
```

```css
/* Recomendado */
border-top: 0;
font: 100%/1.6 palatino, georgia, serif;
padding: 0 lem 2em;
```

Unidades en propiedades

A no ser que sea requerido, se debe omitir la unidad en los valores “0” de los atributos. Además, se debe poner los ceros antes del decimal en los valores entre -1 y 1.

```
flex: 0px; /* El componente flex requiere la unidad "px". */
/* No es necesario indicar la unidad */
margin: 0;
padding: 0;
/* No se omite el cero al principio */
font-size: 0.8em;
```

Notación hexadecimal

Usar la notación de 3 caracteres siempre que sea posible. La notación hexadecimal de 3 caracteres es más corta y concisa, si el color lo permite.

```
/* No recomendado */
color: #eebbcc;
/* Recomendado */
color: #ebc;
```

Declaraciones !important (o no tanto)

¡Evitarlo! Estas declaraciones rompen con la herencia por “cascada” del CSS, lo que dificulta el entendimiento de cómo se aplican y componen los estilos sobre los elementos. Se sugiere utilizar algún selector específico (cómo el de ID) para aplicar un estilo muy particular al elemento y sobre-escribir el estilo por herencia.

Ejemplo: Declaraciones !important

```
/* No recomendado */
.example {
  font-weight: bold !important;
}
```

```
/* Recomendado */
.example {
  font-weight: bold;
}
```

Validación

Una de las herramientas interesantes, gratuita, que provee W3C, es la de validación de CSS. La misma verificará el archivo CSS según la gramática, propiedades y valores definidos en la especificación de CSS 2.1. Esto puede ser útil para reducir el código sin efecto o valores ilegales. Validar provee una base de calidad en nuestro código y mejora su manutención.

- CSS Validator – https://jigsaw.w3.org/css-validator/

Otros estándares de estilo para CSS

- WordPress: https://developer.wordpress.org/coding-standards/wordpress-coding-standards/css/
- Bootstrap: https://codeguide.co/#css
- Mozilla: https://firefox-source-docs.mozilla.org/code-quality/coding-style/css_guidelines.html

JavaScript

JavaScript

“JavaScript es un robusto lenguaje de programación que se puede aplicar a un documento HTML y usarse para crear interactividad dinámica en los sitios web. Fue inventado por Brendan Eich, cofundador del proyecto Mozilla.”

- Mozilla Developer Guide

```
function setUserName() {
  const myName = prompt("Please enter your name.");
  if (!myName) {
    setUserName();
  } else {
    localStorage.setItem("name", myName);
    myHeading.textContent = `Mozilla is cool, ${myName}`;
  }
}
```

Nomenclatura

La guía de estilo de Google para JavaScript tiene un conjunto de reglas que aplican a todos los identificadores.

El nombre de un identificador debe:

- Ser lo más descriptivo posible.
- Estar compuesto por letras y, opcionalmente, también números y guiones bajos.

Nomenclatura

También hay reglas específicas para ciertos identificadores, por ejemplo:

- Los nombres de los paquetes, métodos y variables se escriben con camel case minúscula.
- Los nombres de los métodos o variables privados deben tener un guión bajo al final.
- Los nombres de las clases se escriben con camel case mayúscula.
- Las constantes del programa se escriben con todas las letras en mayúsculas, con palabras separadas por guiones bajos.

Nomenclatura

“Los identificadores utilizan solo letras y dígitos ASCII y, en un pequeño número de casos señalados a continuación, guiones bajos y muy raramente (cuando es necesario según frameworks como Angular) signos de dólar.

Proporciona un nombre lo más descriptivo posible, dentro de lo razonable. No te preocupes por ahorrar espacio horizontal, ya que es mucho más importante **que tu código sea comprensible de inmediato** para un nuevo lector. **No utilices abreviaturas que sean ambiguas o desconocidas** para lectores fuera de tu proyecto, y no abrevies eliminando letras dentro de una palabra.”

- Google JavaScript Style Guide, punto 6.1

Ejemplos

Correcto

```
// Identificador normalmente se abrevia como Id.
// Es una constante del programa y está en mayúsculas.
const CUSTOMER_ID = 1234;

// No usa abreviaturas.
let errorCount = 1;

// DNS es un acrónimo bien conocido.
let dnsConnectionIndex = 0;

// La nomenclatura ayuda a entender que, dado un ancho
// y un alto, se devuelve el area del rectangulo.
function getRectangleArea(width, height) {
  return width * height;
}
```

Incorrecto

```
// No tiene un significado claro o especifico.
let n = 3;

// Abreviatura no es clara.
let nCompConns = 7;

// Elimina letras internas a la palabra customer.
let cstmrId = 3456;

// No se entiende qué es lo que hace la función.
function myFunction() {
  document.getElementById("demo").innerHTML = "Hello World!";
}
```

Declaración de variables locales

- Declarar todas las variables locales con **const** o **let**.
  - Usar const por defecto, a menos que una variable necesite ser reasignada.
  - No usar la keyword var. El scope de las variables declaradas con var es el cuerpo inmediato de la función, mientras que las variables declaradas con let tienen como scope el bloque inmediato que se indica con { }.

- Declarar las variables cuando se necesitan, e inicializarlas tan pronto como sea posible. Las variables locales se deben declarar cerca del punto en el que se van a utilizar por primera vez, para minimizar su alcance.

- Una única variable por declaración. Evitar cosas como `let a = 1, b = 2;`

Comentarios

“Los comentarios de bloque se sangran al mismo nivel que el código circundante. Pueden estar en estilo /* … */ o //-style. Para comentarios multi-línea en /* … */, las líneas subsiguientes deben comenzar con * alineado con el * en la línea anterior, para hacer los comentarios evidentes sin contexto adicional.”

- Google JavaScript Style Guide, punto 4.8.1

Más allá del formato a seguir, es importante recalcar que los comentarios deben ser significativos. Idealmente, el código debería ser lo más descriptivo posible, evitando la necesidad de agregar comentarios. Si un comentario no aporta a la claridad del programa, es mejor no agregarlo.

Ejemplos

Correcto

```
/*
 * Esto está
 * bien.
 */

// Y esto también
// lo está.

/* Esto también está bien. */
```

Incorrecto

```
/*
Esto no está
bien.*/

/*   Y esto tampoco
/*lo está.
```

Indentación

“Cada vez que se abre un nuevo bloque o una construcción similar a un bloque, la sangría aumenta en dos espacios. Cuando el bloque termina, la sangría vuelve al nivel de sangría anterior. El nivel de sangría se aplica tanto al código como a los comentarios en todo el bloque.”

- Google JavaScript Style Guide, punto 4.2

Los caracteres de tabulación no se utilizan para la sangría, únicamente se utilizan espacios.

Ejemplos

Correcto

```
function setUserName() {
  const myName = prompt("Please enter your name.");
  if (!myName) {
    setUserName();
  } else {
    localStorage.setItem("name", myName);
    myHeading.textContent = `Mozilla is cool, ${myName}`;
  }
}
```

Incorrecto

```
function setUserName() {
const myName = prompt("Please enter your name.");
if (!myName) {
setUserName();
} else {
localStorage.setItem("name", myName);
myHeading.textContent = `Mozilla is cool, ${myName}`;
}
}
```

Uso de llaves

Se requieren llaves para todas las estructuras de control (por ejemplo, if, else, for, do, while, así como cualquier otra), incluso si el cuerpo contiene solo una declaración. La primera declaración de un bloque no vacío debe comenzar en su propia línea.

Excepción: Una declaración if simple que puede encajar completamente en una sola línea sin envolver (y que no tiene un else) puede mantenerse en una sola línea sin llaves cuando mejora la legibilidad. Este es el único caso en el que una estructura de control puede omitir llaves y nuevas líneas.

Ejemplos

Correcto

```
if (someVeryLongCondition()) {
  doSomething();
}

for (let i = 0; i < foo.length; i++) {
  bar(foo[i])
};

if (shortCondition()) foo();

function doNothing() {}
```

Incorrecto

```
if (someVeryLongCondition())
  doSomething();

for (let i = 0; i < foo.length; i++) bar(foo[i]);
```

DRY: Don’t Repeat Yourself

El principio "Don't Repeat Yourself" (“No Te Repitas”) busca minimizar las repeticiones de código. Esto no solo mejora la calidad del código, sino que también facilita la incorporación de nuevas funciones y la realización de modificaciones en el futuro.

Evitar la duplicación de datos, lógica o funciones en el código no solo previene la extensión innecesaria del código, sino que también minimiza el tiempo dedicado a mantenimiento, depuración y modificaciones. La necesidad de efectuar ajustes en el código implica, en este caso, intervenciones en múltiples ubicaciones.

Ejemplos

Correcto

```
let drinks = ['lemonade', 'soda', 'tea', 'water'];
let food = ['beans', 'chicken', 'rice'];

function logItems (array) {
  for (let i = 0; i < array.length; i++) {
    console.log(array[i]);
  }
}

logItems(drinks);
logItems(food);
```

Incorrecto

```
let drinks = ['lemonade', 'soda', 'tea', 'water'];
let food = ['beans', 'chicken', 'rice'];

console.log(drinks[0]);
console.log(drinks[1]);
console.log(drinks[2]);
console.log(drinks[3]);

console.log(food[0]);
console.log(food[1]);
console.log(food[2]);
```

Referencias

- Google HTML/CSS Style Guide (google.github.io)
- HTML: HyperText Markup Language Documentation (developer.mozilla.org)
- HTML: Living Standard (html.spec.whatwg.org)
- Coding standards: what are they, and why do you need them (blog.codacy.com)
- Google JavaScript Style Guide (google.github.io)
- JavaScript Basics (developer.mozilla.org)
- What is DRY Code (codinglead.co)
- El principio DRY (codeyourapps.com)