Curso de

# Clean Code y Buenas Prácticas con JavaScript

Alex Camacho

**Nuestro código tiene que ser simple y directo, debería de leerse con la misma facilidad que un texto bien escrito.**

*Grady Booch entusiasta del diseño de patrones*

# Deuda técnica

## Los 4 tipos de deuda

# Imprudente y deliberada

# Imprudente e inadvertida

# Prudente y deliberada

# Prudente
# e inadvertida

# Refactorización de código

¿Qué es y cuándo hacerlo?

# ¿Qué es refactorizar?

La refactorización es un proceso que tiene como objetivo **mejorar el código** de un proyecto **sin alterar su comportamiento** para que sea más entendible y tolerante a cambios.

# ¿Cuándo refactorizar?

- El código sea de baja calidad.
- Al detectar un *code smell.*

# Las reglas del diseño simple

# Las 4 reglas del diseño simple:

1. El código pasa correctamente los test.
2. Revela la intención del diseño.
3. Respeta el principio DRY.
4. Tiene el menor número posible de elementos.

# Clean Code

¿Qué es?

# Clean code

Es un término popularizado por Robert C. Martin en su libro **Clean Code: A handbook of Agile Software Craftsmanship** en el 2008.

**El código limpio es aquel que se ha escrito con la intención de que otra persona lo entienda.**

*Anónimo*

# ¿Cómo usar var, let y const?

# Reglas de nomenclatura

# Nombres pronunciables y expresivos

# Nombres sin información técnica

# Usar lenguaje ubicuo

# Nombres según el tipo de dato

# Arrays

# Booleanos

# Números

# Funciones

# Clases

# Ámbito de las variables

# Ámbito de las variables

No todo es escribir buenos nombres, también es fundamental entender cómo funciona el ámbito o *scope* en JavaScript.

# Ámbito global

# Ámbito local o de función

# Ámbito de bloque

# Ámbito estático

# Hoisting

# Funciones

**Se sabe que se ha desarrollado código limpio cuando cada función hace exactamente lo que su nombre indica.**

*Ward Cunningham co-autor del manifiesto ágil*

# Declaración de una función

# Expresión de una función

# Argumentos de una función

# Parámetros por defecto

# Parámetro rest y operador spread

# Funciones de flecha

# Clases

# Prototype y ECMAScript

Todos los objetos de JavaScript heredan del objeto prototipo, antes de ES6 se trabajaba de forma desordenada y posterior a ES6 tiene el soporte para trabajar con POO.

# Constructores

# Métodos

# Herencia

# Tamaño reducido

Principio de responsabilidad única

# ¿Las clases se organizan?

# Organización de clases

- Variables
- Constantes
- Variables estáticas
- Métodos
- Funciones estáticas
- Getters y setters

# Los comentarios

¿Utilizarlos o no?

**No comentes código mal escrito, reescríbelo.**

*Brian Kernighan*

# Reglas para trabajar en equipo

Formato coherente

# Problemas similares, soluciones similares

# Densidad, apertura y distancia vertical

# Lo más importante siempre va primero

# Indentación

# Principio DRY

No te repitas a ti mismo

# Don’t repeat yourself

El principio DRY se asegura que cuando se detecte código duplicado, este sea extraído a una clase o función para utilizarlo en donde se necesite.

# Algoritmos eficientes

Notación Big-O

**El código limpio no es solo aquel que es fácil de leer, también debe de ser eficiente, o sea, su rendimiento es óptimo.**

# Big O notation

La notación Big-O se utiliza para describir el comportamiento de un algoritmo, tanto temporal como espacial.

# O(1) constante

# O(log n)
# logarítmica

# O(n) lineal

# O(log n)

# O(n^2) cuadrática

# O(2^n)
# exponencial