Taller de Diagnóstico Inicial

Melanny Yilyan Guate
Asignatura: Arquitectura y Diseño del Software
Fecha: 27 de octubre 2025
Tema: Proceso de Evaluación, Documentación y Herramientas de Calidad.

Objetivo de la Clase: Realizar un diagnóstico de calidad de software. Aprender a usar herramientas profesionales para analizar, corregir y documentar un proyecto de código, asegurando no solo su funcionamiento sino también su comprensión. Además, investiga el propósito de las herramientas que utilizas.

Plan A: Flujo de Trabajo Integrado (Si se pueden instalar extensiones)

Investigación de Herramientas / Tools Investigation

ESLint

[Español]:
ESLint es una herramienta que analiza el código JavaScript para detectar errores, malas prácticas y problemas de estilo.
Ayuda a mantener un código limpio, uniforme y sin errores lógicos o sintácticos.

[English]:
ESLint is a tool that analyzes JavaScript code to detect errors, bad practices, and style issues.
It helps keep the code clean, consistent, and free of logical or syntax errors.

Prettier

[Español]:
Prettier es un formateador automático de código que corrige la indentación, espacios, comillas y otros detalles de estilo.
Su objetivo es hacer que el código sea más legible y consistente sin cambiar su funcionalidad.

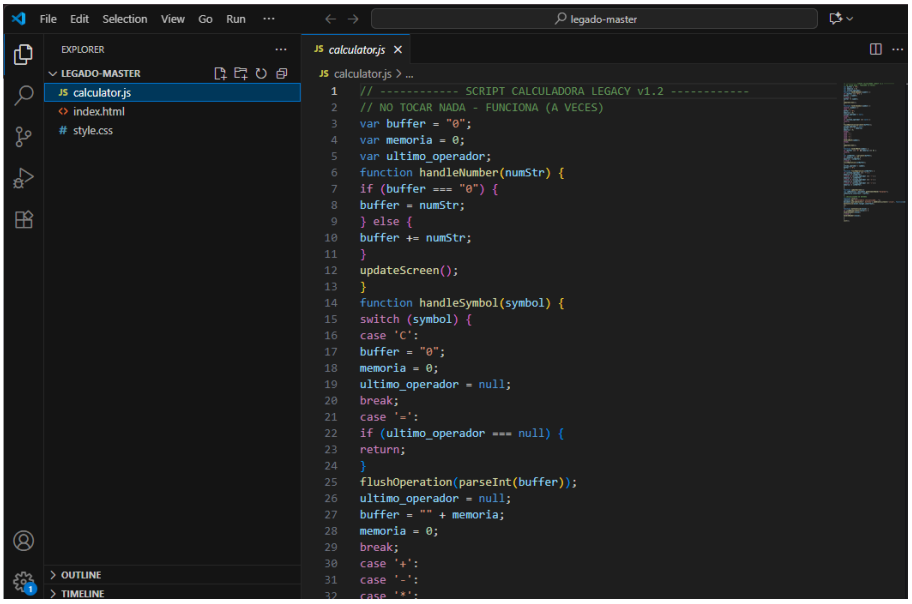
[English]:
Prettier is an automatic code formatter that adjusts indentation, spacing, quotes, and other style details.
Its goal is to make code more readable and consistent without altering its functionality.

Paso 1: Preparación del Entorno Evidencia Visual / Visual Evidence:

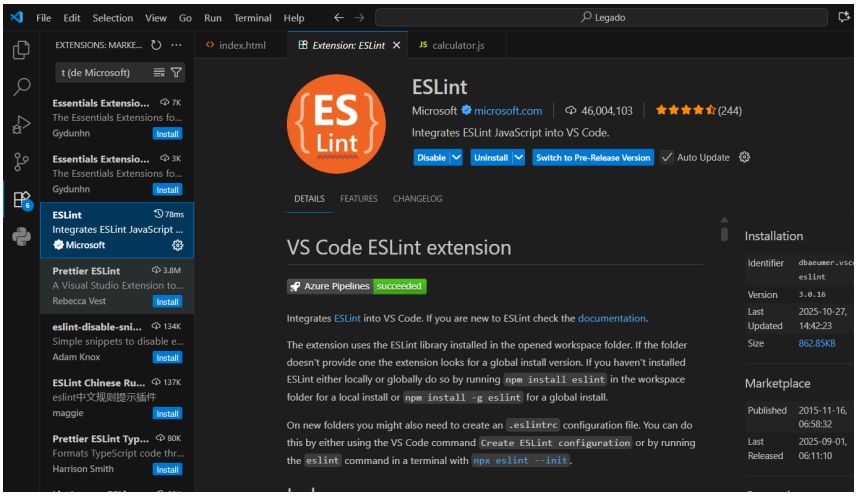
1. Clonar el Proyecto: Clona el repositorio del proyecto "Legado" desde GitHub

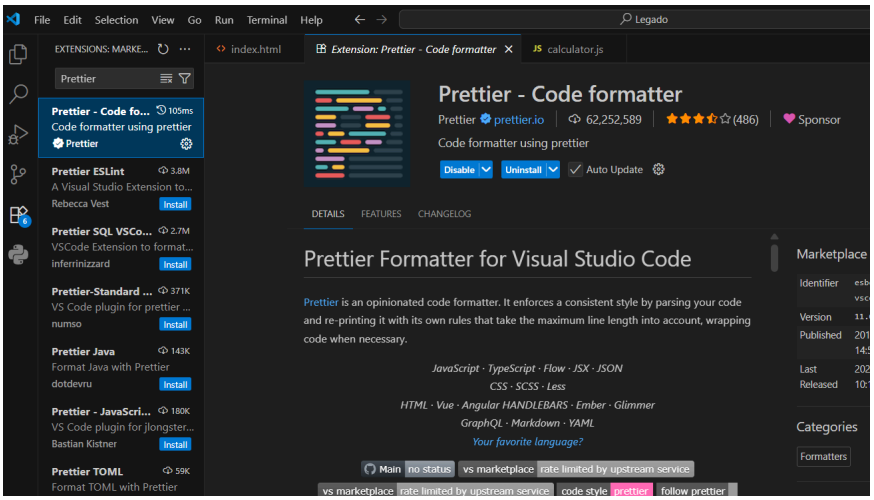
```
C:\Users\yilgr\OneDrive\Desktop>git clone https://github.com/yilgr19/Legado.git Legado-Clonado
Cloning into 'Legado-Clonado'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 4 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (4/4), done.

C:\Users\yilgr\OneDrive\Desktop>cd Legado-Clonado

C:\Users\yilgr\OneDrive\Desktop\Legado-Clonado>code .

C:\Users\yilgr\OneDrive\Desktop\Legado-Clonado>
```

2. Abrir en VS Code y Navegar: Abre la carpeta en VS Code y explora los archivos.



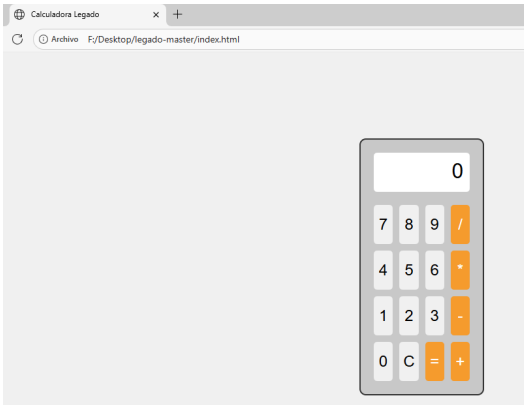
3. Instalar Extensiones: Instalar ESLint y Prettier - Code formatter.

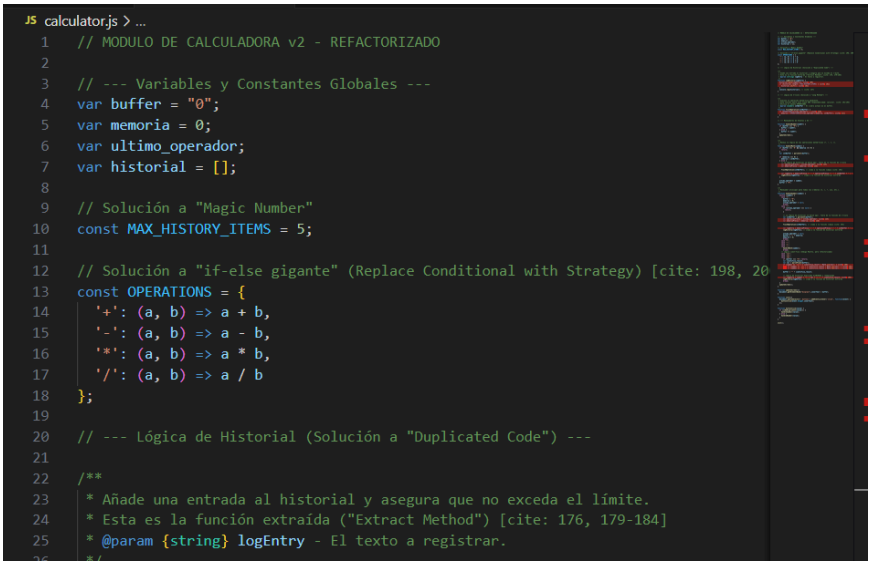




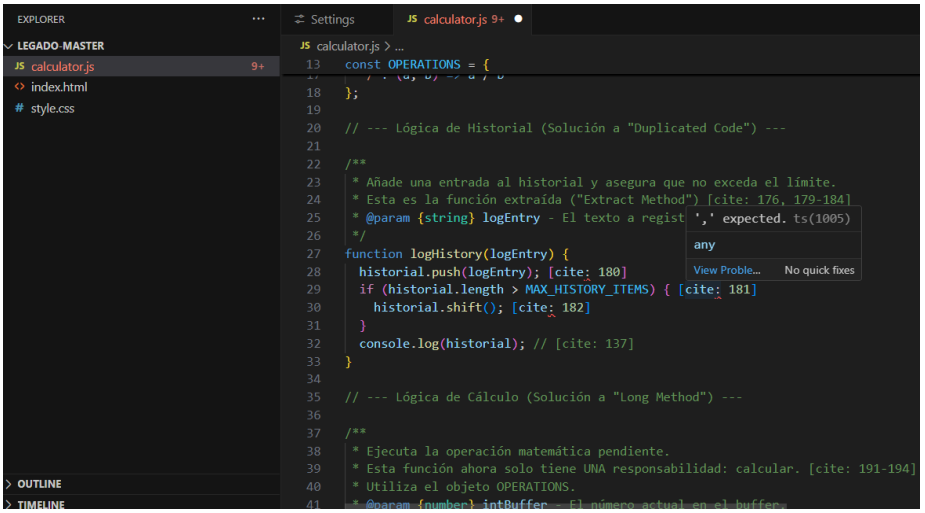
Paso 2: Análisis y Diagnóstico

1. Ejecutar y Documentar "Antes": Abre index.html en el navegador. Toma una captura de pantalla del archivo calculator.js mostrando su estado original desordenado.



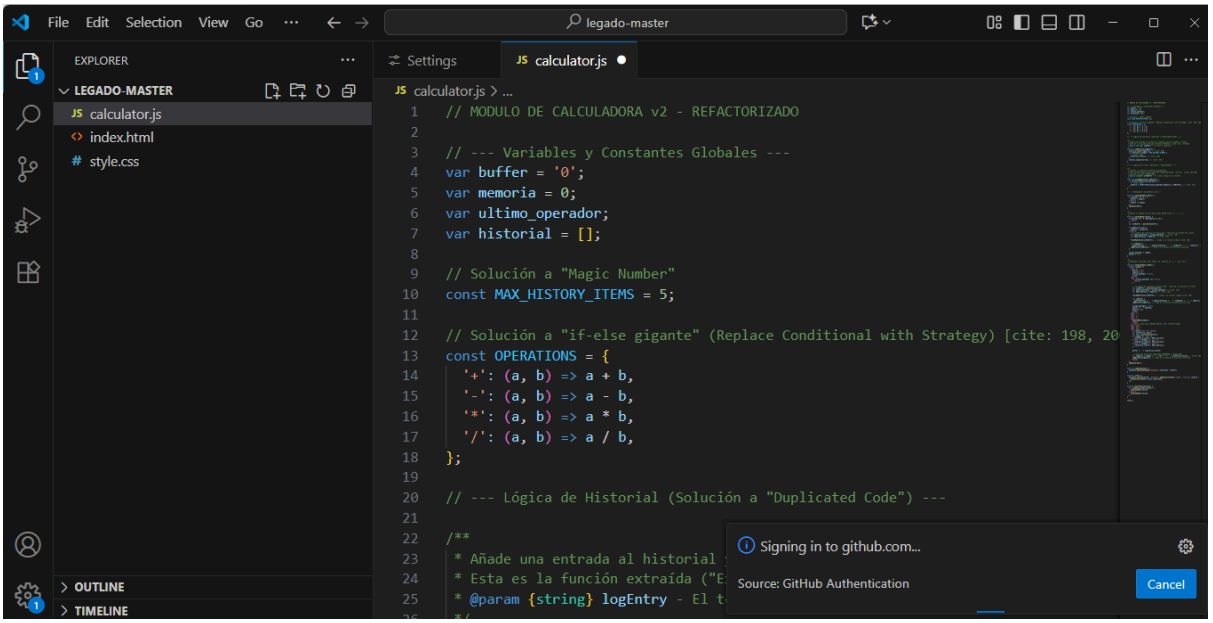


2. Diagnóstico con ESLint: Configura ESLint. Dedica tiempo a analizar las advertencias que aparecen. Pasa el mouse sobre ellas para entender los problemas reportados.

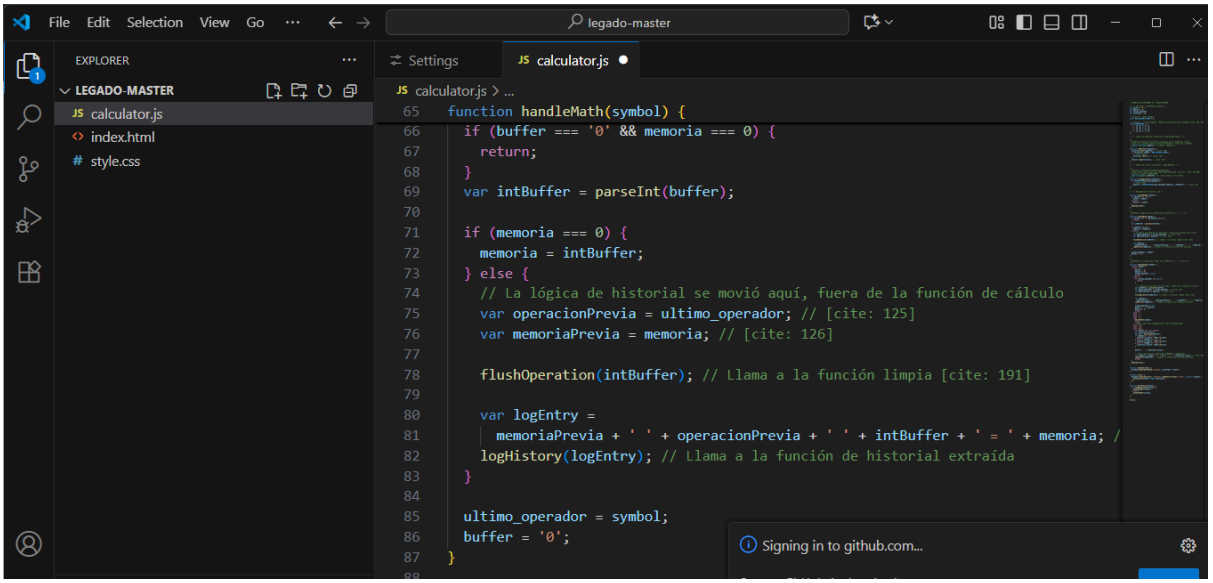


Paso 3: Corrección Automática

1. Formateo con Prettier: En el archivo calculator.js, presiona Alt + Shift + F para corregir automáticamente el estilo del código

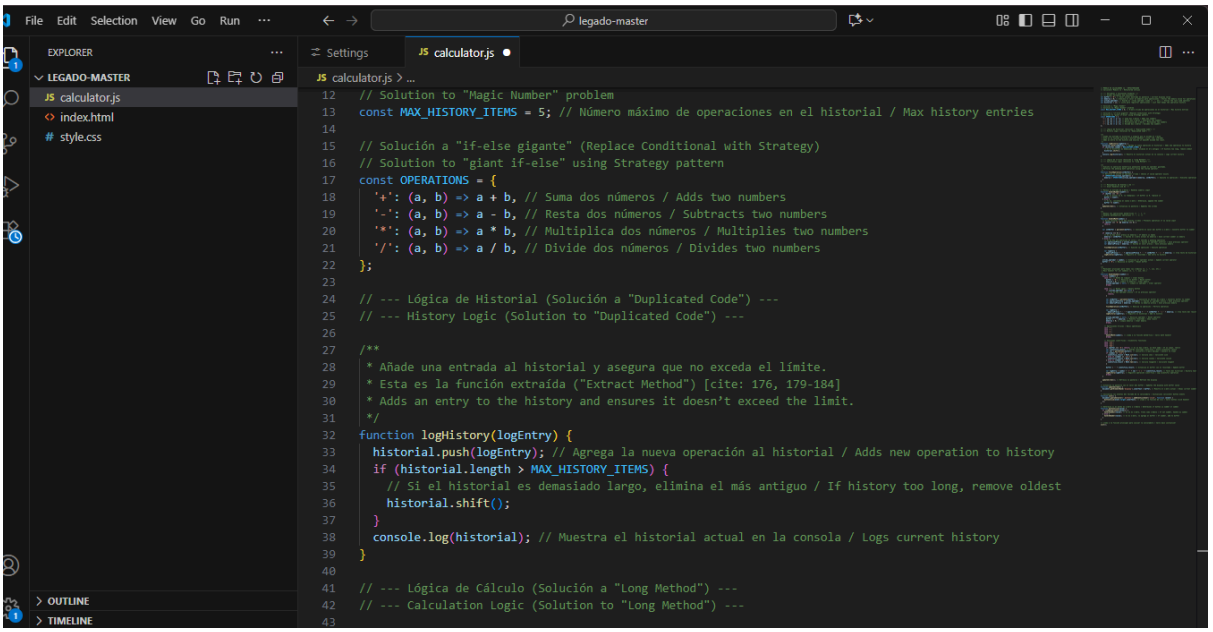


2. Documentar el "Después": Toma una captura del mismo archivo, ahora limpio y formateado.



Paso 4: Documentación Bilingüe del Código

1. Añadir comentarios: Ahora que el código está funcional y limpio, tu tarea es documentarlo. En el archivo calculator.js, añade un comentario línea por línea explicando qué hace.



Análisis de Hallazgos / Findings Analysis

Nº	Aspecto Analizado	Descripción (Español)	Description (English)
1	**Estructura modular**	El código está dividido en secciones lógicas: variables globales, operaciones, historial y UI. Esto mejora la legibilidad y el mantenimiento.	The code is organized into logical sections: globals, operations, history, and UI, improving readability and maintainability.
2	**Uso de constantes**	Se define `MAX_HISTORY_ITEMS` para evitar números mágicos. Buen ejemplo de refactorización.	Defines `MAX_HISTORY_ITEMS` to avoid magic numbers — a good refactoring practice.
3	**Patrón Strategy aplicado**	El objeto `OPERATIONS` reemplaza condicionales múltiples para las operaciones básicas.	The `OPERATIONS` object replaces multiple conditional statements, implementing the Strategy pattern.
4	**Control del historial**	La función `logHistory()` evita duplicación de código y controla el tamaño máximo del historial.	The `logHistory()` function prevents duplicated logic and limits the history length.
5	**Responsabilidad única**	Cada función realiza una tarea específica (ej. `flushOperation()` solo calcula). Cumple con el principio SRP.	Each function has a single purpose (e.g., `flushOperation()` only performs calculations), following SRP (Single Responsibility Principle).
6	**Interacción con la UI**	`updateScreen()` y `init()` manejan el DOM sin mezclar la lógica de negocio.	`updateScreen()` and `init()` handle DOM operations separately from business logic.
7	**Manejo de eventos**	El evento click en `.buttons` delega correctamente el manejo según el tipo de entrada (número o símbolo).	Button click events are properly delegated depending on input type (number or symbol).
8	**Código reutilizable**	Las funciones científicas (`sin`, `cos`, `tan`) fueron integradas de forma clara con una sola función de registro.	Scientific functions are integrated cleanly with a single history logging mechanism.
9	**Legibilidad y comentarios**	Excelente uso de comentarios bilingües y consistentes, ideal para trabajo colaborativo o educativo.	Excellent bilingual commenting, useful for collaboration and teaching.

10	**Posibles mejoras**	Se podría mejorar el manejo de errores (división por 0, valores no válidos) y usar `let/const` en lugar de `var` para modernizar el código.	Could improve error handling (e.g., division by zero) and replace `var` with `let/const` for modern JavaScript standards.

Repositorio de GitHub / GitHub Repository

[Español]:
El proyecto final se encuentra alojado en el siguiente enlace:
https://github.com/RobinsonGo09/legado