Objetivo del Proyecto

Desarrollo de la Calculadora Interactiva en JavaScript

El proyecto consistía en desarrollar una calculadora interactiva en JavaScript que no solo permitiera realizar operaciones básicas como suma, resta, multiplicación, división y raíz cuadrada, sino también en registrar un historial detallado de las operaciones realizadas. La calculadora debía funcionar en la consola, sin interfaz gráfica, lo que me ayudó a enfocarme en la lógica y estructura del código. El desarrollo incluyó decisiones de diseño importantes y, aunque tuve algunos desafíos, pude implementar cada elemento de manera que resultara en una calculadora estable y funcional. **Decisiones de Diseño**

1. **Estructura del Código**
   La primera decisión fue modularizar el código, dividiéndolo en varias funciones que se encargaran de tareas específicas. Esto hizo que el código fuera más limpio . Me di cuenta de que sin una organización clara, la lógica se volvería confusa, especialmente porque el programa debía gestionar diferentes tipos de operaciones y validar entradas. Funciones:
   - menuCalculadora(): Esta función controla el menú principal de la calculadora, mostrando las opciones al usuario y guiando a la siguiente selección. ○ menuOperaciones(): Se encarga de ofrecer el menú de operaciones básicas, facilitando la elección de la operación.
   - calculadora(): Esta función es el núcleo de la calculadora, encargándose de validar las entradas y de ejecutar la operación seleccionada.
   - realizarOperacion(): Aquí se ejecuta la operación matemática en sí, y decidí ubicarla fuera de calculadora() para separar la lógica de la interfaz de usuario. ○ mostrarHistorial(): Muestra el historial completo de operaciones realizadas, facilitando al usuario revisar el detalle de cada cálculo.

2. **Gestión del Historial de Operaciones**
   La idea era permitir que el usuario revisara el historial de operaciones al final de cada cálculo. Elegí un array para almacenar las operaciones, y

cada operación se guarda como un objeto que incluye el tipo de operación, los operandos, y el resultado.

Desafío: Al implementar el historial, me preocupaba que fuera difícil acceder a la información de cada operación. Tras analizar diferentes opciones, descubrí que los objetos dentro del array eran la mejor opción en términos de eficiencia y claridad.

3. **Validación de Entradas**
   Otro punto crítico era validar las entradas del usuario. Las funciones validarInput() y calculadora() se encargan de verificar que las entradas sean números válidos antes de realizar cualquier operación. Para esto, usé parseFloat() y isNaN() para convertir las entradas de texto a números y verificar su validez. También incluí una verificación específica en realizarOperacion() para evitar errores comunes, como dividir por cero.

Desafío: La validación de entradas fue uno de los problemas que más tiempo me llevó resolver. Al principio, no estaba capturando todos los casos de entradas no válidas, y esto provocaba errores en el flujo de la calculadora. Tras varios ajustes y pruebas, logré una validación que cubre la mayoría de situaciones.

4. **Uso de Try-Catch para el Manejo de Errores**
   Durante el desarrollo, observé que aún con validaciones cuidadosas, algunos errores podían surgir por entradas inesperadas o por problemas en la lógica. Para manejar esto, envolví la lógica principal de calculadora() en un bloque try-catch. Esto me permitió capturar excepciones de forma ordenada y mostrar mensajes claros sin interrumpir la ejecución del programa.

Desafío: Al principio, no me pareció necesario, pero al enfrentar errores inesperados me di cuenta de que try-catch era indispensable para asegurar la estabilidad del programa. Esta decisión fue esencial para evitar que errores menores interrumpieran la experiencia del usuario.

5. **Operación de Raíz Cuadrada**
   La raíz cuadrada fue la operación especial en el proyecto, ya que requiere solo un número, a diferencia de las otras operaciones que usan dos operandos. Decidí que en esta operación solo se usaría num1, lo cual simplificó el proceso y permitió realizar esta operación con facilidad.

Desafío: No quería hacer la raíz cuadrada tan rígida y pensé en cómo permitir más flexibilidad. Finalmente, decidí que mantenerla simple, usando solo un número, era suficiente para este caso.

Desafíos Encontrados y Soluciones

1. **Validación de Entradas de Usuario**
   Como la entrada del usuario en prompt es texto, convertirlo a número y validar su contenido fue un reto. Inicialmente, probé varias formas de validación hasta que la combinación de parseFloat() con isNaN() demostró ser la solución.

2. **División entre Cero**
   Este fue otro problema. Sin validación, cualquier división entre cero podía provocar errores o resultados indeseados. En realizarOperacion(), implementé una verificación específica para capturar este caso y mostrar un mensaje de error en lugar de intentar la división.

3. **Estructura del Historial**
   Consideré varias opciones, pero al final, almacenar cada operación como un objeto en un array resultó la mejor solución. Esto me permitió acceder a la información de cada operación de manera sencilla y eficiente.

4. **Manejo de Excepciones**

El manejo de errores fue algo que añadí al final, cuando me di cuenta de que algunos errores inesperados podrían interrumpir el programa. Implementar try-catch fue necesario para asegurar que el programa pudiera manejar situaciones inesperadas de manera controlada. **Conclusión**

Este proyecto fue un ejercicio en la implementación de una calculadora interactiva con JavaScript, con un enfoque en la modularidad, validación de datos, y manejo de errores. Logré desarrollar una calculadora que cumple con su propósito y ofrece una experiencia de usuario sencilla.