

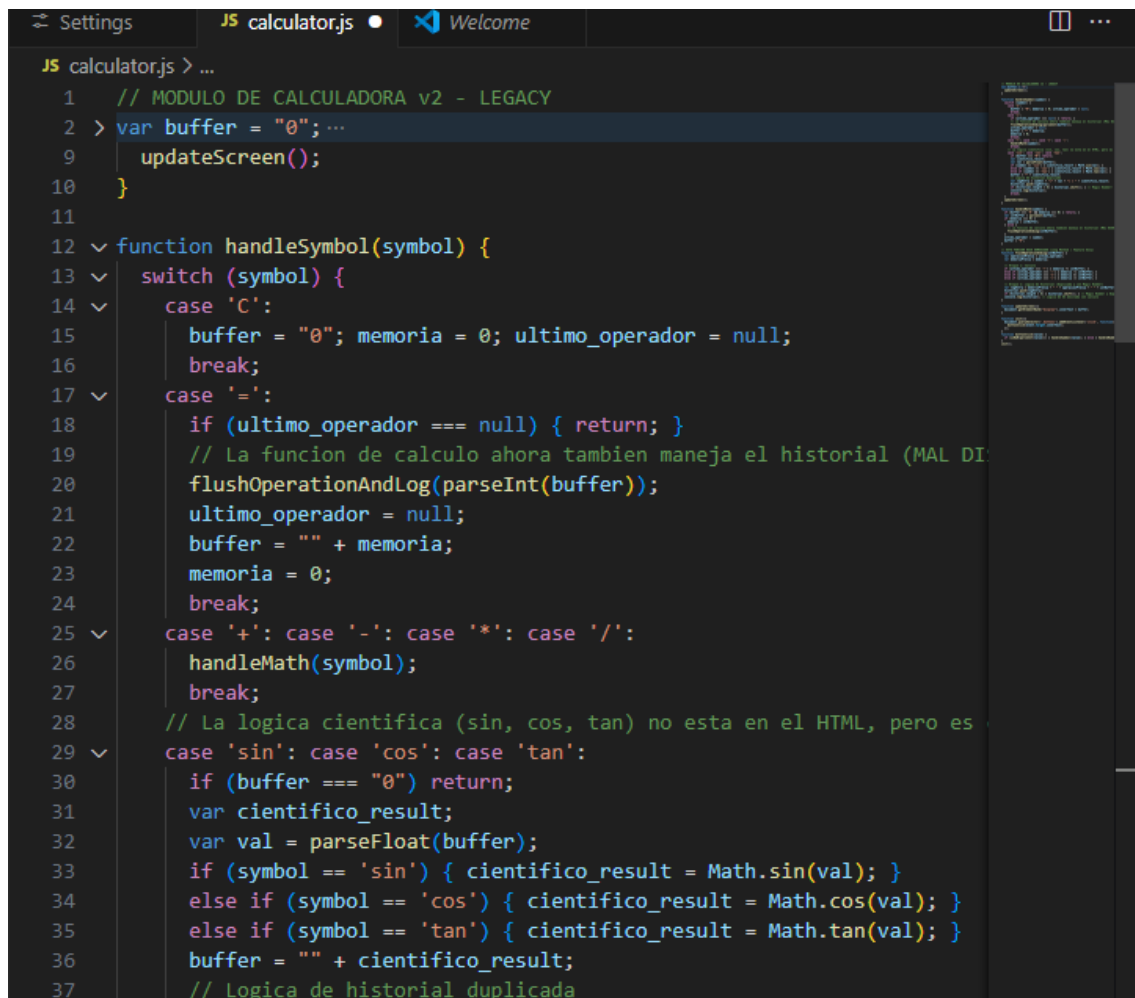
Taller de Refactorización y Métricas de Calidad

Fecha: 28 de October de 2025

Nombre: Melanny Guate

1. Auditoría Inicial / Initial Audit

Antes de aplicar refactorización, el código legacy de la calculadora presentaba alta complejidad ciclomática, especialmente en las funciones `handleSymbol` (Complejidad ≈ 10) y `flushOperationAndLog` (Complejidad ≈ 12). Esto indicaba múltiples responsabilidades, condicionales anidados y duplicación de lógica.



```
JS calculator.js > ...
1 // MODULO DE CALCULADORA v2 - LEGACY
2 > var buffer = "0"; ...
9   updateScreen();
10 }
11
12 function handleSymbol(symbol) {
13   switch (symbol) {
14     case 'C':
15       buffer = "0"; memoria = 0; ultimo_operador = null;
16       break;
17     case '=':
18       if (ultimo_operador === null) { return; }
19       // La funcion de calculo ahora tambien maneja el historial (MAL DI
20       flushOperationAndLog(parseInt(buffer));
21       ultimo_operador = null;
22       buffer = "" + memoria;
23       memoria = 0;
24       break;
25     case '+': case '-': case '*': case '/':
26       handleMath(symbol);
27       break;
28     // La logica cientifica (sin, cos, tan) no esta en el HTML, pero es
29     case 'sin': case 'cos': case 'tan':
30       if (buffer === "0") return;
31       var cientifico_result;
32       var val = parseFloat(buffer);
33       if (symbol == 'sin') { cientifico_result = Math.sin(val); }
34       else if (symbol == 'cos') { cientifico_result = Math.cos(val); }
35       else if (symbol == 'tan') { cientifico_result = Math.tan(val); }
36       buffer = "" + cientifico_result;
37       // Logica de historial duplicada
```

Before refactoring, the legacy calculator code showed high cyclomatic complexity, particularly in the functions `handleSymbol` (Complexity ≈ 10) and `flushOperationAndLog` (Complexity ≈ 12). This revealed multiple responsibilities, nested conditionals, and duplicated logic.

```
JS calculator.js > logHistory
1 // MODULO DE CALCULADORA v2 - REFACTORIZADO
2 > var buffer = "0"; ...
12 '*': (a, b) => a * b,
13 '/': (a, b) => a / b
14 };
15
16 function logHistory(logEntry) {
17   historial.push(logEntry);
18   if (historial.length > MAX_HISTORY_ITEMS) {
19     historial.shift();
20   }
21   console.log(historial);
22 }
23
24 function handleNumber(numStr) {
25   buffer = buffer === "0" ? numStr : buffer + numStr;
26   updateScreen();
27 }
28
29 function handleSymbol(symbol) {
30   switch (symbol) {
31     case 'C':
32       buffer = "0"; memoria = 0; ultimo_operador = null;
33       break;
34     case '=':
35       if (!ultimo_operador) return;
36       flushOperation(parseInt(buffer));
37       const logEntryEq = memoria + " " + ultimo_operador + " " + parseInt(buffer);
38       logHistory(logEntryEq);
39       ultimo_operador = null;
40       buffer = "" + memoria;
41       memoria = 0;
```

2. Análisis de Code Smells / Code Smells Analysis

Duplicated Code (Violación de DRY):

El bloque de historial dentro de handleSymbol y flushOperationAndLog era idéntico.

Example:

```
historial.push(logEntry);
```

```
if (historial.length > 5) { historial.shift(); } // Magic Number console.log(historial);
```

Magic Number: El número 5 aparece sin contexto en dos lugares del código.

Long Method: La función flushOperationAndLog realizaba cálculo y gestión de historial, violando el principio de responsabilidad única.

English:

The history block was duplicated in both `handleSymbol` and `flushOperationAndLog`. The number 5 was a magic number without explanation, and `flushOperationAndLog` performed both arithmetic and logging, violating the Single Responsibility Principle.

3. Proceso de Refactorización / Refactoring Process

Se aplicaron tres patrones principales de refactorización: Extract Constant, Extract Method y Replace Conditional with Strategy.

1. Extract Constant:

```
const MAX_HISTORY_ITEMS = 5
```

1. Extract Method:

```
function logHistory(logEntry) { historial.push(logEntry);  
if (historial.length > MAX_HISTORY_ITEMS) { historial.shift(); } console.log(historial);  
}
```

2. Replace Conditional with Strategy:

```
const OPERATIONS = { '+': (a,b)=>a+b, '-':(a,b)=>a-b, '*':(a,b)=>a*b, '/':(a,b)=>a/b };  
function flushOperation(intBuffer){ if(OPERATIONS[ultimo_operador]) memoria  
= OPERATIONS[ultimo_operador](memoria,intBuffer); }
```

English:

Applied Extract Constant, Extract Method, and Replace Conditional with Strategy patterns to reduce duplication, improve readability, and isolate responsibilities.

2. Auditoría Final / Final Audit

Después de la refactorización, las funciones principales redujeron drásticamente su complejidad ciclomática: `flushOperation` (Complejidad ≈ 2) y `handleSymbol` (Complejidad ≈ 4). El código es más legible, mantenible y fácil de probar.

Conclusión / Conclusion:

La reducción en la complejidad y eliminación de duplicaciones demuestra un código más limpio y alineado con los principios SOLID. This shows a cleaner, maintainable, and professional-grade software design.

CODIGO

```
// MODULO DE CALCULADORA v2 - REFACTORIZADO
```

```
var buffer = "0";
```

```
var memoria = 0;
```

```
var ultimo_operador;
```

```
var historial = [];
```

```
const MAX_HISTORY_ITEMS = 5;
```

```
const OPERATIONS = {
```

```
  '+': (a, b) => a + b,
```

```
  '-': (a, b) => a - b,
```

```
  '*': (a, b) => a * b,
```

```
  '/': (a, b) => a / b
```

```
};
```

```
function logHistory(logEntry) {
```

```
  historial.push(logEntry);
```

```
  if (historial.length > MAX_HISTORY_ITEMS) {
```

```
    historial.shift();
```

```
  }
```

```
  console.log(historial);
```

```
}
```

```
function handleNumber(numStr) {  
    buffer = buffer === "0" ? numStr : buffer + numStr;  
    updateScreen();  
}
```

```
function handleSymbol(symbol) {  
    switch (symbol) {  
        case 'C':  
            buffer = "0"; memoria = 0; ultimo_operador = null;  
            break;  
        case '=':  
            if (!ultimo_operador) return;  
            flushOperation(parseInt(buffer));  
            const logEntryEq = memoria + " " + ultimo_operador + " " + parseInt(buffer) + " = " +  
memoria;  
            logHistory(logEntryEq);  
            ultimo_operador = null;  
            buffer = "" + memoria;  
            memoria = 0;  
            break;  
        case '+': case '-': case '*': case '/':  
            handleMath(symbol);  
            break;  
        case 'sin': case 'cos': case 'tan':  
            if (buffer === "0") return;  
            const val = parseFloat(buffer);  
            const result = Math[symbol](val);
```

```
    buffer = "" + result;

    logHistory(symbol + "(" + val + ") = " + result);

    break;
}

updateScreen();
}
```

```
function handleMath(symbol) {
    if (buffer === '0' && memoria === 0) return;

    const intBuffer = parseInt(buffer);

    if (memoria === 0) {
        memoria = intBuffer;
    } else {
        flushOperation(intBuffer);

        const logEntry = memoria + " " + ultimo_operador + " " + intBuffer + " = " + memoria;
        logHistory(logEntry);
    }

    ultimo_operador = symbol;
    buffer = "0";
}
```

```
function flushOperation(intBuffer) {
    if (OPERATIONS[ultimo_operador]) {
        memoria = OPERATIONS[ultimo_operador](memoria, intBuffer);
    }
}
```

```
function updateScreen() {
```

```
document.getElementById("display").innerText = buffer;  
}
```

```
function init() {  
  document.querySelector('.buttons').addEventListener('click', (event) => {  
    buttonClick(event.target.innerText);  
  });  
}
```

```
function buttonClick(value) {  
  isNaN(parseInt(value)) ? handleSymbol(value) : handleNumber(value);  
}
```

```
init();
```