Bairon Sebastián Ardila Mendoza

https://github.com/BaironArd/legado-master/tree/entrega2legacyV2

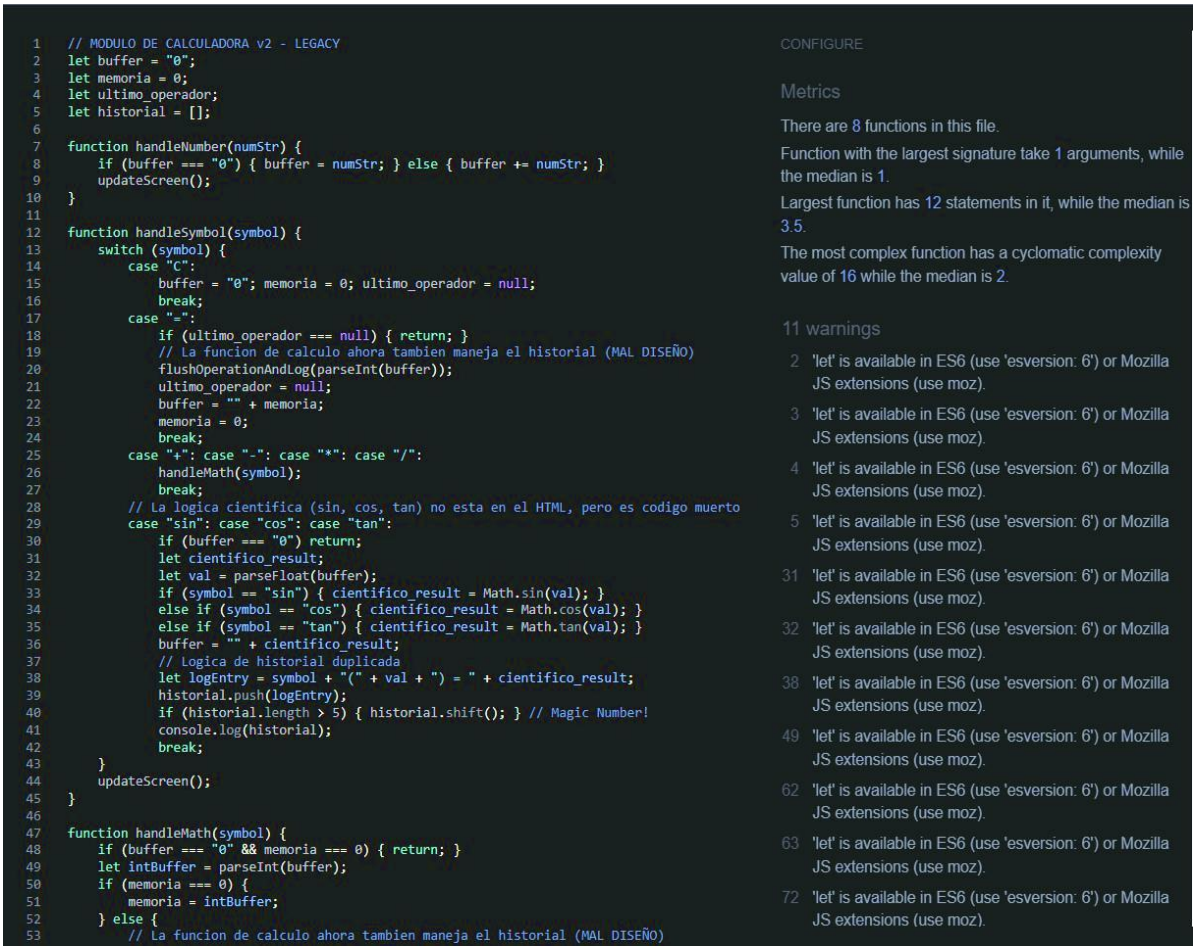
Taller de Refactorización y Métricas de Calidad.

Sección 1: Auditoría Inicial / Initial Audit

Captura de Métricas "Antes" (mostrando la Complejidad Ciclomática Alta) / "Before" Metrics Screenshot (showing High Cyclomatic Complexity)

- **[Español]**: En esta sección se muestra la captura de la herramienta de métricas (JSHint) antes de realizar la refactorización del código. Al analizar el código "Legacy v2" de la calculadora, observamos que las funciones handleSymbol y flushOperationAndLog tienen una **complejidad ciclomática alta**. Esta alta complejidad indica que el código es difícil de probar y mantener, lo que es una señal clara de que se necesita refactorización.

- **[English]**: This section shows the screenshot of the metrics tool (JSHint) before refactoring the code. Upon analyzing the "Legacy v2" calculator code, we observed that the functions handleSymbol and flushOperationAndLog have **high cyclomatic complexity**. This high complexity indicates that the code is hard to test and maintain, which is a clear signal that refactoring is needed.



```
// MODULO DE CALCULADORA v2 - LEGACY
let buffer = "0";
let memoria = 0;
let ultimo_operador;
let historial = [];

function handleNumber(numStr) {
    if (buffer === "0") { buffer = numStr; } else { buffer += numStr; }
    updateScreen();
}

function handleSymbol(symbol) {
    switch (symbol) {
        case "C":
            buffer = "0"; memoria = 0; ultimo_operador = null;
            break;
        case "=":
            if (ultimo_operador === null) { return; }
            // La funcion de calculo ahora tambien maneja el historial (MAL DISEÑO)
            flushOperationAndLog(parseInt(buffer));
            ultimo_operador = null;
            buffer = "" + memoria;
            memoria = 0;
            break;
        case "+": case "-": case "*": case "/":
            handleMath(symbol);
            break;
        // La logica cientifica (sin, cos, tan) no esta en el HTML, pero es codigo muerto
        case "sin": case "cos": case "tan":
            if (buffer === "0") return;
            let cientifico_result;
            let val = parseFloat(buffer);
            if (symbol == "sin") { cientifico_result = Math.sin(val); }
            else if (symbol == "cos") { cientifico_result = Math.cos(val); }
            else if (symbol == "tan") { cientifico_result = Math.tan(val); }
            buffer = "" + cientifico_result;
            // Logica de historial duplicada
            let logEntry = symbol + "(" + val + ") = " + cientifico_result;
            historial.push(logEntry);
            if (historial.length > 5) { historial.shift(); } // Magic Number!
            console.log(historial);
            break;
    }
    updateScreen();
}

function handleMath(symbol) {
    if (buffer === "0" && memoria === 0) { return; }
    let intBuffer = parseInt(buffer);
    if (memoria === 0) {
        memoria = intBuffer;
    } else {
        // La funcion de calculo ahora tambien maneja el historial (MAL DISEÑO)
```

CONFIGURE

Metrics

There are 8 functions in this file.

Function with the largest signature take 1 arguments, while the median is 1.

Largest function has 12 statements in it, while the median is 3.5.

The most complex function has a cyclomatic complexity value of 16 while the median is 2.

11 warnings

2 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

3 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

4 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

5 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

31 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

32 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

38 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

49 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

62 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

63 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

72 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

Sección 2: Análisis de "Code Smells" / "Code Smells" Analysis

Código Duplicado / Duplicated Code (Violation of DRY)

[Español]: El código contiene una violación del principio **DRY** (Don't Repeat Yourself), especialmente en el "Bloque 2: Lógica de Historial" dentro de la función flushOperationAndLog. Esta lógica se repite en el caso de handleSymbol, donde también se manipula el historial de manera similar. Este es un ejemplo claro de **Código Duplicado**.

Código Original (Duplicado en flushOperationAndLog y handleSymbol):

```
let logEntry = symbol + "(" + val + ") = " + cientifico_result;
            historial.push(logEntry);
            if (historial.length > 5) { historial.shift(); } // Magic Number!
            console.log(historial);
```

[English]: The code contains a violation of the **DRY** (Don't Repeat Yourself) principle, particularly in **Block 2: History Logic** inside the flushOperationAndLog function. This logic is repeated in handleSymbol, where history is also manipulated in a similar way. This is a clear example of **Duplicated Code**.

Magic Number

[Español]: En el código, encontramos el uso del número **5** en la condición historial.length > 5, lo cual es un **número mágico**. Este valor no tiene ningún significado claro en el código, lo que hace que sea más difícil de entender y mantener. Es importante extraer este número a una constante que tenga un nombre descriptivo.

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

[English]: In the code, we found the use of the number **5** in the condition historial.length > 5, which is a **magic number**. This value has no clear meaning in the code, making it harder to understand and maintain. It is important to extract this number into a constant with a descriptive name.

Método Largo / Long Method

[Español]: La función flushOperationAndLog está violando el **Principio de Responsabilidad Única** al hacer demasiadas cosas: tanto el cálculo de la operación como la gestión del historial. Esto hace que la función sea difícil de entender y mantener.

Código Original (Método Largo):

```
function flushOperationAndLog(intBuffer) {
    let operacionPrevia = ultimo_operador;
    let memoriaPrevia = memoria;

    // Bloque 1: Calculo
    if (ultimo_operador === "+") { memoria += intBuffer; }
    else if (ultimo_operador === "-") { memoria -= intBuffer; }
    else if (ultimo_operador === "*") { memoria *= intBuffer; }
    else if (ultimo_operador === "/") { memoria /= intBuffer; }

    // Bloque 2: Logica de Historial (Duplicada y con Magic Number)
    let logEntry = memoriaPrevia + " " + operacionPrevia + " " + intBuffer + " = " + memoria;
    historial.push(logEntry);
    if (historial.length > 5) { historial.shift(); } // Magic Number y Duplicacion!
    console.log(historial); // Logica de UI mezclada con calculo
}
```

[English]: The flushOperationAndLog function is violating the **Single Responsibility Principle** by doing too many things: both performing the operation calculation and managing the history. This makes the function hard to understand and maintain.

Sección 3: Proceso de Refactorización / Refactoring Process

Refactorización de Código Duplicado y Magic Number / Refactoring Duplicated Code and Magic Number

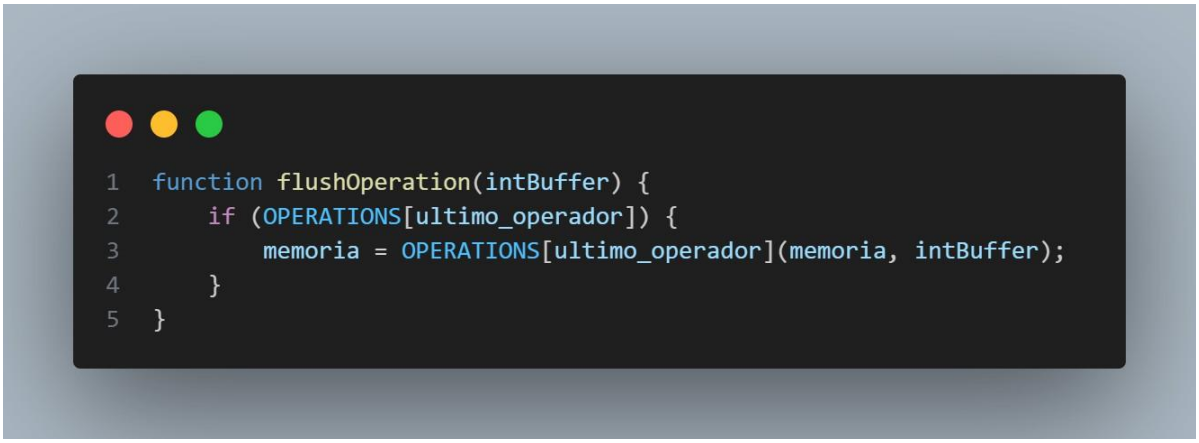
[Español]: Para solucionar el problema de **Código Duplicado**, creamos una nueva función logHistory, que ahora maneja la lógica de agregar entradas al historial y verificar si el tamaño del historial excede el límite de 5. También extraímos el **número mágico** a una constante llamada MAX_HISTORY_ITEMS.

```
// [Español]: Constante que define el número máximo de elementos en el historial.
// [English]: Constant defining the maximum number of items in the history.
const MAX_HISTORY_ITEMS = 5;

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

[English]: To solve the **Duplicated Code** problem, we created a new function logHistory which now handles the logic of adding entries to the history and checking if the history size exceeds the limit of 5. We also extracted the **magic number** into a constant called MAX_HISTORY_ITEMS.

Refactorización del Método Largo / Refactoring Long Method

[Español]: Refactorizamos la función flushOperationAndLog para que solo realice el cálculo (Bloque 1). La lógica de crear la entrada de historial y llamar a logHistory se movió a las funciones handleMath y handleSymbol, respectivamente.



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

[English]: We refactored the flushOperationAndLog function to only perform the calculation (Block 1). The logic of creating the history entry and calling logHistory was moved to the handleMath and handleSymbol functions, respectively.

Sección 4: Auditoría Final / Final Audit

Captura de Métricas "Después" / "After" Metrics Screenshot

[Español]: Después de aplicar las refactorizaciones, la complejidad ciclómica de las funciones flushOperation y handleSymbol ha disminuido drásticamente, lo que indica que el código ahora es más limpio, fácil de probar y mantener.

[English]: After applying the refactorings, the cyclomatic complexity of the flushOperation and handleSymbol functions has decreased drastically, indicating that the code is now cleaner, easier to test, and maintain.

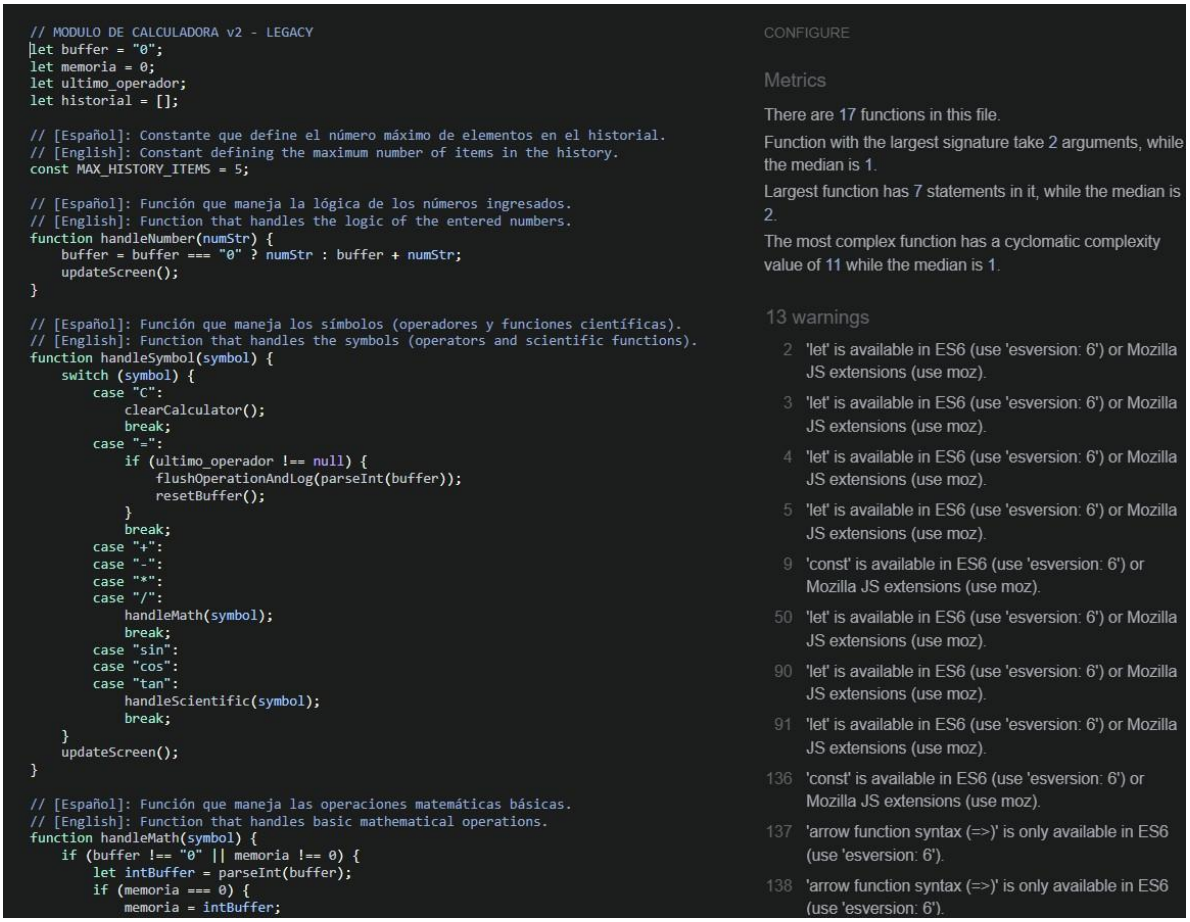
```javascript
// MODULO DE CALCULADORA v2 - LEGACY
let buffer = "0";
let memoria = 0;
let ultimo_operador;
let historial = [];

// [Español]: Constante que define el número máximo de elementos en el historial.
// [English]: Constant defining the maximum number of items in the history.
const MAX_HISTORY_ITEMS = 5;

// [Español]: Función que maneja la lógica de los números ingresados.
// [English]: Function that handles the logic of the entered numbers.
function handleNumber(numStr) {
    if (buffer === "0") { buffer = numStr; } else { buffer += numStr; }
    updateScreen();
}

// [Español]: Función que maneja los símbolos (operadores y funciones científicas).
// [English]: Function that handles the symbols (operators and scientific functions).
function handleSymbol(symbol) {
    switch (symbol) {
        case "c":
            buffer = "0"; memoria = 0; ultimo_operador = null;
            break;
        case "=":
            if (ultimo_operador === null) { return; }
            // [Español]: La función de cálculo ahora maneja la lógica del historial de forma centralizada.
            // [English]: The calculation function now handles the history logic in a centralized manner.
            flushOperationAndLog(parseInt(buffer));
            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;
            {
                let cientifico_result;
                let val = parseFloat(buffer);

                if (symbol == "sin") {
                    cientifico_result = Math.sin(val);
                }
                else if (symbol == "cos") {
                    cientifico_result = Math.cos(val);
                }
                else if (symbol == "tan") {
                    cientifico_result = Math.tan(val);
                }

                buffer = "" + cientifico_result;
                logHistory(symbol + "(" + val + ") = " + cientifico_result);
            }
            break;
    }
    updateScreen();
}

// [Español]: Función que maneja las operaciones matemáticas básicas.
// [English]: Function that handles basic mathematical operations.
function handleMath(symbol) {
    if (buffer === "0" && memoria === 0) { return; }
    let intBuffer = parseInt(buffer);
    if (memoria === 0) {
        memoria = intBuffer;
    } else {
        flushOperationAndLog(intBuffer);
    }
    ultimo_operador = symbol;
    buffer = "0";
}

// [Español]: Función que realiza la operación de cálculo y maneja el historial.
// [English]: Function that performs the calculation operation and handles the history.
function flushOperationAndLog(intBuffer) {
    flushOperation(intBuffer);
    // [Español]: Solo realiza el cálculo.
    // [English]: Just performs the calculation.
    let logEntry = memoria + " " + ultimo_operador + " " + intBuffer + " = " + memoria;
    logHistory(logEntry);
    // [Español]: Llama a logHistory para manejar el historial.
    // [English]: Calls logHistory to handle the history.
}

// [Español]: Función que maneja el cálculo de las operaciones aritméticas.
// [English]: Function that handles the calculation of arithmetic operations.
function flushOperation(intBuffer) {
    if (OPERATIONS[ultimo_operador]) {
        memoria = OPERATIONS[ultimo_operador](memoria, intBuffer);
    }
}

// [Español]: Función que maneja el historial y se asegura de que no exceda el límite.
// [English]: Function that handles the history and ensures it does not exceed the limit.
function logHistory(logEntry) {
    historial.push(logEntry);
    if (historial.length > MAX_HISTORY_ITEMS) {
        historial.shift();
    }
    console.log(historial);
}

function updateScreen() {
    document.getElementById("display").innerText = buffer;
}

function init() {
    document.querySelector(".buttons").addEventListener("click", function (event) {
        buttonClick(event.target.innerText);
    });
}

function buttonClick(value) {
    if (isNaN(parseInt(value))) { handleSymbol(value); } else { handleNumber(value); }
}

init();

// [Español]: Objeto que almacena las operaciones matemáticas.
// [English]: Object that stores mathematical operations.
const OPERATIONS = {
    "+": (a, b) => a + b,
    "-": (a, b) => a - b,
    "*": (a, b) => a * b,
    "/": (a, b) => a / b
};
```

Sección 4: Auditoría Final / Final Audit

Captura de Métricas "Después" / "After" Metrics Screenshot

[Español]: Después de aplicar las refactorizaciones, la complejidad ciclómica de las funciones flushOperation y handleSymbol ha disminuido drásticamente, lo que indica que el código ahora es más limpio, fácil de probar y mantener.

[English]: After applying the refactorings, the cyclomatic complexity of the flushOperation and handleSymbol functions has decreased drastically, indicating that the code is now cleaner, easier to test, and maintain.



```
// MODULO DE CALCULADORA v2 - LEGACY
let buffer = "0";
let memoria = 0;
let ultimo_operador;
let historial = [];

// [Español]: Constante que define el número máximo de elementos en el historial.
// [English]: Constant defining the maximum number of items in the history.
const MAX_HISTORY_ITEMS = 5;

// [Español]: Función que maneja la lógica de los números ingresados.
// [English]: Function that handles the logic of the entered numbers.
function handleNumber(numStr) {
    buffer = buffer === "0" ? numStr : buffer + numStr;
    updateScreen();
}

// [Español]: Función que maneja los símbolos (operadores y funciones científicas).
// [English]: Function that handles the symbols (operators and scientific functions).
function handleSymbol(symbol) {
    switch (symbol) {
        case "C":
            clearCalculator();
            break;
        case "=":
            if (ultimo_operador !== null) {
                flushOperationAndLog(parseInt(buffer));
                resetBuffer();
            }
            break;
        case "+":
        case "-":
        case "*":
        case "/":
            handleMath(symbol);
            break;
        case "sin":
        case "cos":
        case "tan":
            handleScientific(symbol);
            break;
    }
    updateScreen();
}

// [Español]: Función que maneja las operaciones matemáticas básicas.
// [English]: Function that handles basic mathematical operations.
function handleMath(symbol) {
    if (buffer !== "0" || memoria !== 0) {
        let intBuffer = parseInt(buffer);
        if (memoria === 0) {
            memoria = intBuffer;
```

CONFIGURE

Metrics

There are 17 functions in this file.

Function with the largest signature take 2 arguments, while the median is 1.

Largest function has 7 statements in it, while the median is 2.

The most complex function has a cyclomatic complexity value of 11 while the median is 1.

13 warnings

2 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

3 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

4 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

5 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

9 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

50 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

90 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

91 'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

136 'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).

137 'arrow function syntax (=>)' is only available in ES6 (use 'esversion: 6').

138 'arrow function syntax (=>)' is only available in ES6 (use 'esversion: 6').