

# MATERIAL EXTRA

Erros em JavaScript

# ERROS EM JAVASCRIPT

Erros em JavaScript são situações que ocorrem quando algo inesperado acontece durante a execução do código. Eles podem interromper o funcionamento normal de um programa ou causar comportamentos indesejados. Existem diferentes tipos de erros em JavaScript, e é importante saber como identificá-los e tratá-los para garantir que o código funcione corretamente.

## Principais Tipos de Erros em JavaScript:

- **SyntaxError** (Erro de Sintaxe): Ocorre quando há um problema de sintaxe na estrutura do código, como esquecer uma chave ou um ponto e vírgula.

```
console.log("Olá Mundo!") // Falta a aspas de fechamento
```

- **ReferenceError** (Erro de Referência): Acontece quando se tenta acessar uma variável que não foi definida ou está fora do escopo.

```
console.log(variavelInexistente)  
// ReferenceError: variavelInexistente não está definida
```

- **TypeError** (Erro de Tipo): Ocorre quando se tenta realizar uma operação em um valor de tipo inapropriado, como chamar uma função em um valor que não é função.

```
let numero = 5  
numero() // TypeError: numero não é uma função
```

## Tratamento de Erros

Em JavaScript, erros podem ser capturados e tratados para evitar que o programa quebre ou tenha um comportamento inesperado. Para isso, usa-se o bloco try...catch.

```
try {
    // Código que pode gerar um erro
    let resultado = undefinedVariable
} catch (erro) {
    // Tratamento do erro
    console.log('Ocorreu um erro: ' + erro.message)
} finally {
    console.log('Esta mensagem sempre será mostrada')
}
```

- **try:** O código dentro deste bloco é executado normalmente. Se ocorrer um erro, a execução é interrompida e o controle passa para o bloco catch.
- **catch:** Este bloco é executado se ocorrer um erro dentro do bloco try. Aqui, o erro pode ser tratado.
- **finally:** (opcional) Um bloco que sempre será executado após try e catch, independentemente de um erro ter ocorrido ou não.

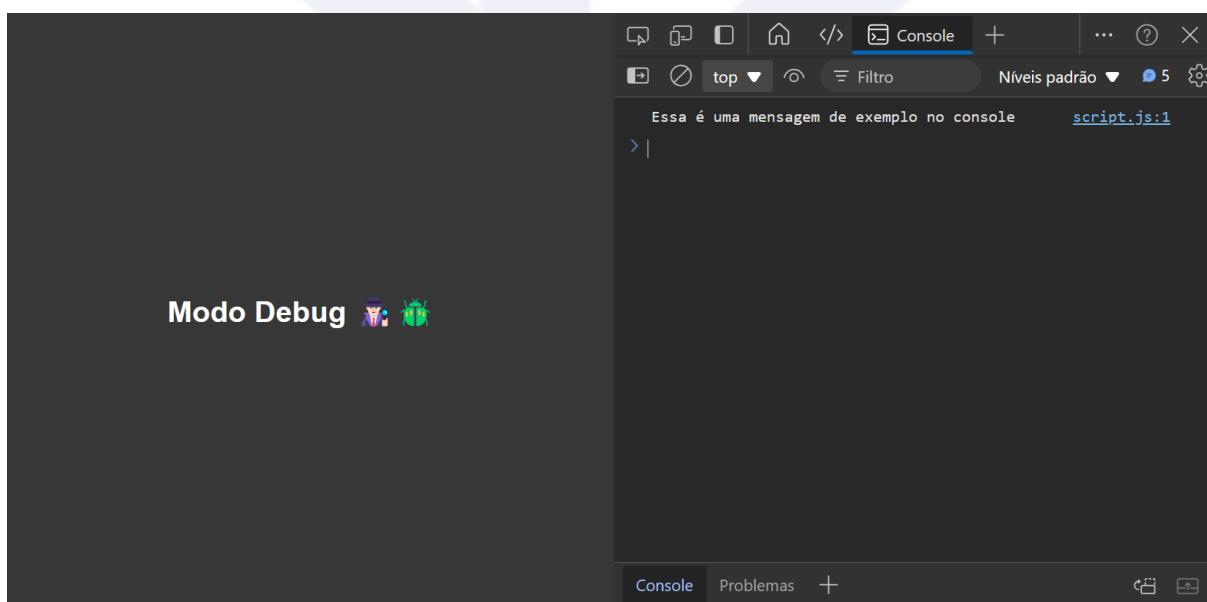
## Modo Debug no Navegador

Os navegadores modernos (como Chrome, Firefox, Edge) oferecem ferramentas de desenvolvedor que permitem depurar e inspecionar o código JavaScript. O **modo de depuração (debug)** ajuda a encontrar e corrigir erros no código.

### Acessando o Modo Debug

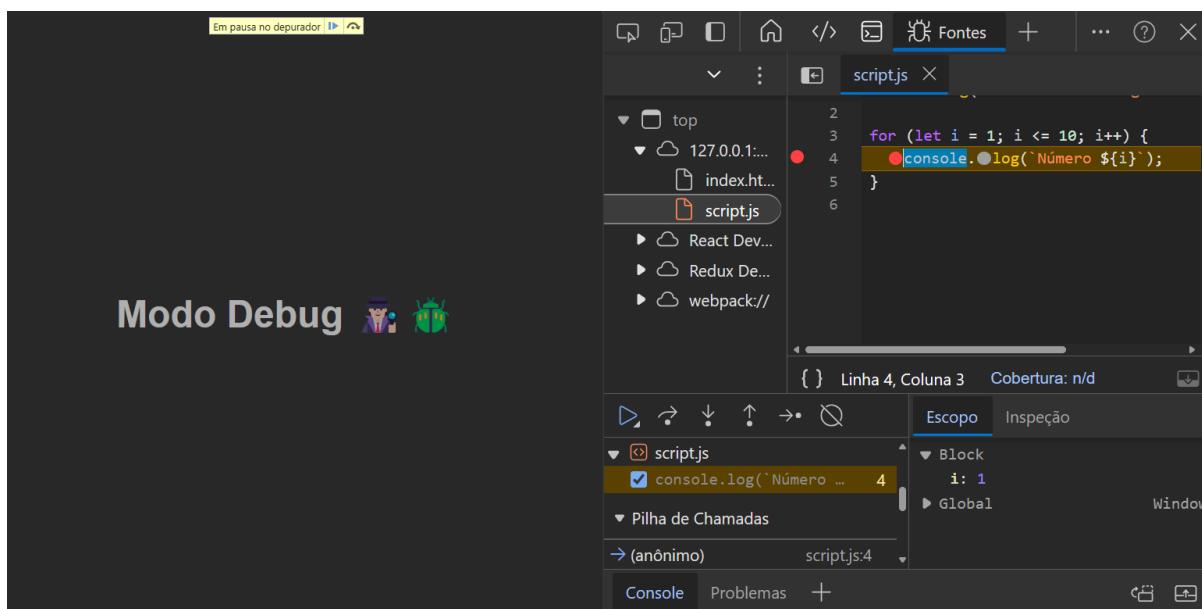
#### 1. Abrindo as Ferramentas de Desenvolvedor:

- Pressione F12 ou Ctrl + Shift + I (no Windows) ou Cmd + Option + I (no Mac) para abrir as ferramentas de desenvolvedor no navegador.
- Na aba **Console**, você pode ver mensagens de erro, avisos e resultados de console.log, além de executar comandos diretamente.



## 2. Usando Breakpoints:

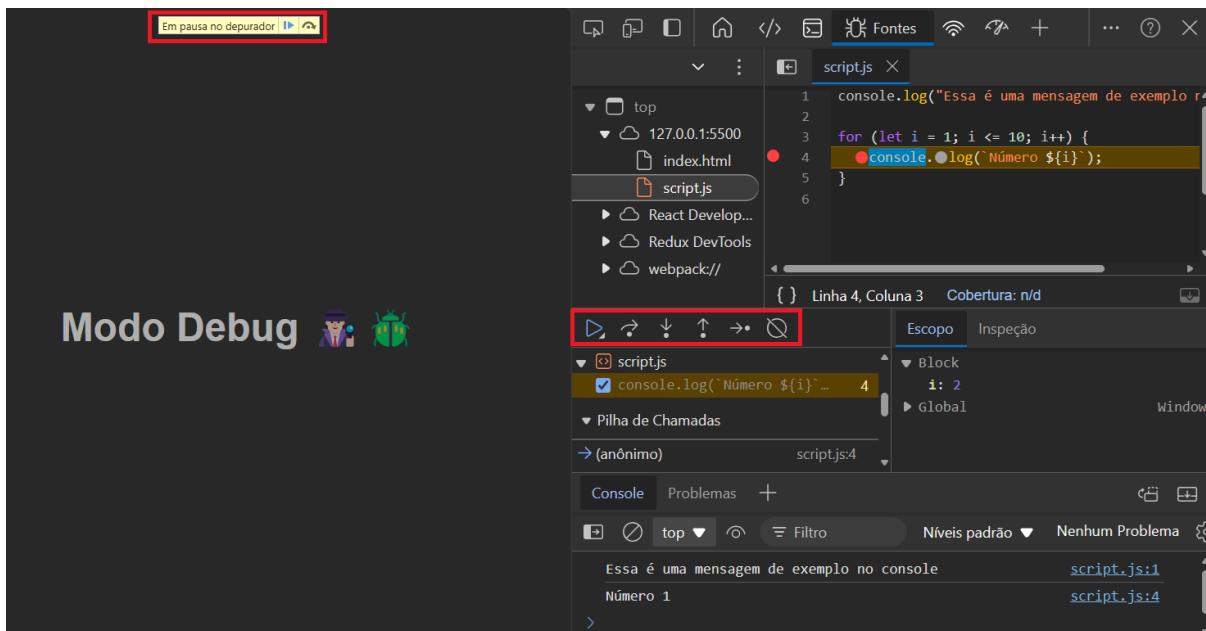
- Na aba **Sources** (ou **Fontes** no Chrome), você pode adicionar **breakpoints** clicando no número da linha do código.
- Quando o código chegar a essa linha, ele será interrompido, permitindo que você examine variáveis, o fluxo do programa e execute o código passo a passo.



## 3. Depuração Passo a Passo:

- Após parar em um **breakpoint**, você pode usar as opções:
  - **Step Over (F10)**: Executa a linha atual e passa para a próxima.
  - **Step Into (F11)**: Entra em funções chamadas na linha atual.
  - **Step Out (Shift + F11)**: Sai da função atual e retorna ao código que a chamou.
  - **Continue (F8)**: Continua a execução até o próximo breakpoint.

**Modo Debug** 🎩 🐛



```
script.js
1 console.log("Essa é uma mensagem de exemplo r
2
3 for (let i = 1; i <= 10; i++) {
4     console.log(`Número ${i}`);
5 }
```

Em pausa no depurador ⏪ ⏹

Linha 4, Coluna 3 Cobertura: n/d

script.js:4

Escopo Inspeção

Block i: 2 Global Window

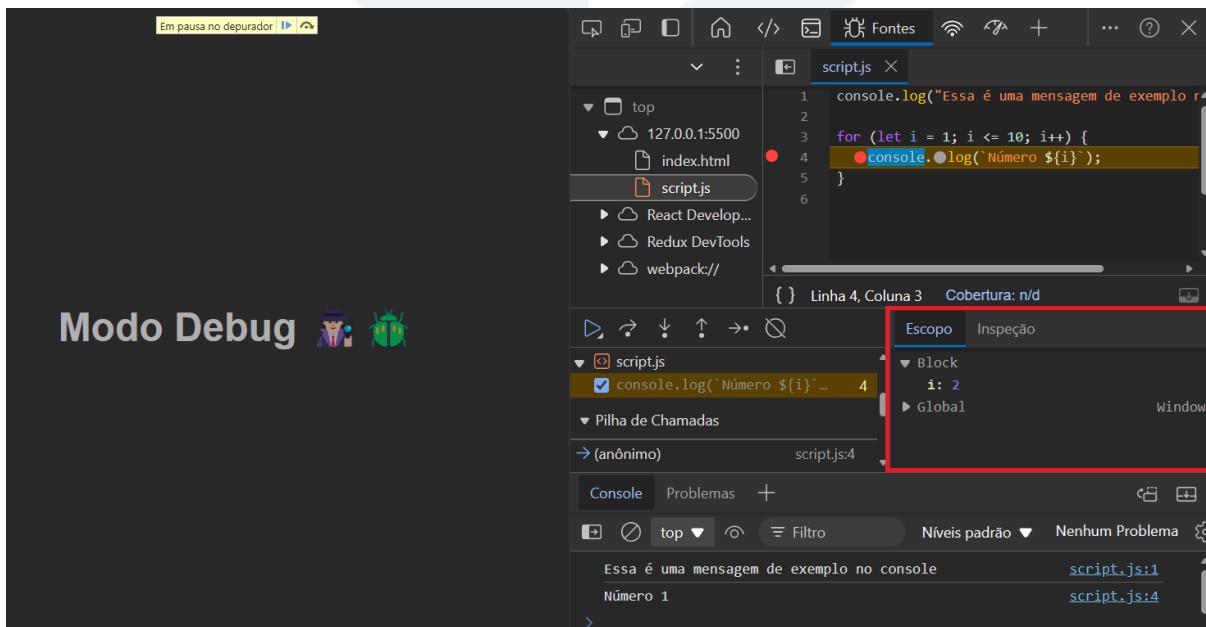
Console Problemas +

Script.js:1 Número 1 Script.js:4

#### 4. Inspecionando Variáveis:

- Na aba **Scope**, você pode visualizar o valor de todas as variáveis no contexto atual. Isso é útil para entender o estado do programa no momento da execução.

**Modo Debug** 🎩 🐛



```
script.js
1 console.log("Essa é uma mensagem de exemplo r
2
3 for (let i = 1; i <= 10; i++) {
4     console.log(`Número ${i}`);
5 }
```

Em pausa no depurador ⏪ ⏹

Linha 4, Coluna 3 Cobertura: n/d

script.js:4

Escopo Inspeção

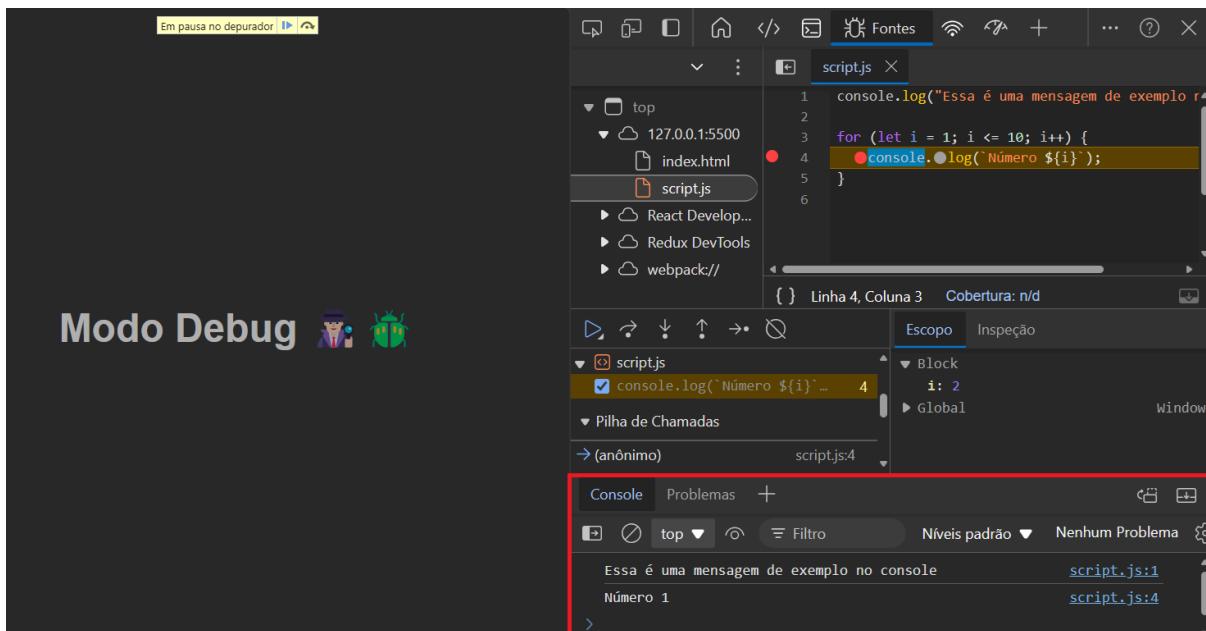
Block i: 2 Global Window

Console Problemas +

Script.js:1 Número 1 Script.js:4

## 5. Usando console.log():

- O uso de console.log no código ajuda a depurar sem parar a execução. Ele imprime valores ou mensagens no console, o que é útil para verificar o fluxo do código ou o estado de variáveis.



## Dicas para Debug:

- Sempre verifique o console:** O console do navegador frequentemente exibe mensagens de erro detalhadas que ajudam a identificar o problema.
- Use breakpoints estrategicamente:** Coloque breakpoints em partes cruciais do código para acompanhar o fluxo e valores.
- Teste com diferentes navegadores:** Às vezes, um erro pode ocorrer apenas em um navegador específico.

## Conclusão

Erros são uma parte inevitável do desenvolvimento em JavaScript, mas com o uso de ferramentas adequadas de depuração e tratamento de erros, é possível identificar e corrigir esses problemas de forma eficaz. O **modo debug** do navegador é uma ferramenta valiosa para examinar o código em tempo de execução, encontrar erros e otimizá-lo.

# BONS ESTUDOS