📌 JavaScript – Erros

❗ O que são erros?

Erros são problemas que acontecem durante a execução do código e podem impedir o programa de funcionar corretamente.

Eles podem surgir por vários motivos, como:

- Erro de digitação
- Variável que não existe
- Tipo de dado errado
- Valor fora do limite permitido
- Código escrito de forma incorreta (erro de sintaxe)

🛠️ Tratamento de Erros em JavaScript

🔹 try...catch

O `try...catch` é usado para **testar um código que pode dar erro** e **tratar esse erro sem quebrar o programa**.

```
try {
  // código que pode dar erro
} catch (err) {
  // código que trata o erro
}
```

- O código dentro do `try` é executado normalmente.
- Se ocorrer um erro, o JavaScript pula para o `catch`.
- O erro é armazenado na variável `err`.

🔹 Tipos de Erros em JavaScript

1️⃣ ReferenceError

Acontece quando você tenta usar uma **variável que não existe**.

```
try {
  x = y + 1; // y não existe
} catch (err) {
  console.log(err.name);
}
```

Também ocorre quando a variável é usada **antes de ser declarada**:

```
let x = y;
let y = 5;
```

---

2 TypeError

Ocorre quando o **tipo de dado é incompatível** com a operação.

```
try {
  let num = 1;
  num.toUpperCase(); // número não tem esse método
} catch (err) {
  console.log(err.name);
}
```

Outro exemplo comum:

```
Anna(); // Anna não é uma função
```

---

3 RangeError

O valor usado está **fora do limite permitido**.

```
try {
  new Array(-1);
} catch (err) {
  console.log(err.name);
}
```

Outro exemplo:

```
let num = 1;
num.toPrecision(500); // limite máximo é 100
```

---

4 URIError

Ocorre ao usar funções de URI com **caracteres inválidos**.

```
try {
  decodeURI("%%%");
} catch (err) {
  console.log(err.name);
}
```

---

5 SyntaxError

Erro de **escrita do código**, ou seja, erro de gramática do JavaScript.

```
let nome = "João; // string não fechada
```

Outro exemplo:

```
Math.round(4.6;
```

⚠️ **Importante:**
`SyntaxError` **não pode ser capturado com try...catch**, porque:

- O código nem chega a executar
- O erro aparece direto no console

6️⃣ EvalError (obsoleto ⚠️)

Era usado antigamente com `eval()`.
Hoje não é mais utilizado e deve ser evitado.

📌 JavaScript – Silent Errors (Erros Silenciosos)

❗ O que são erros silenciosos?

São erros que:

- Não quebram o programa
- Não geram mensagens de erro
- O código continua rodando
- Mas o resultado fica errado

📜 Motivo histórico

No início, o JavaScript **não tinha try...catch**, então muitos erros passaram a ser aceitos sem avisar.

⚠️ Exemplos Comuns de Silent Errors

1️⃣ Divisão por zero

```
let x = 1 / 0;
```

- Não gera erro
- Resultado: `Infinity`

2️⃣ Atribuição (=) no lugar de comparação (== ou ===)

```
let isActive = false;

if (isActive = true) {
  console.log("Active!");
}
```

- `=` atribui valor
- `==` ou `===` compara
- O `if` sempre será verdadeiro

3️⃣ Operações inválidas retornam NaN

```
const result = parseInt("abc");
```

- Código continua rodando
- Resultado: `NaN` (Not a Number)

4️⃣ Propriedade inexistente retorna undefined

```
const user = {};
let name = user.name;
```

- Nenhum erro acontece
- Resultado: `undefined`

5️⃣ Coerção automática de tipos

O JavaScript converte tipos automaticamente:

```
let a = '5' + '2'; // "52"
let b = '5' - '2'; // 3
```

O código funciona, mas a lógica pode ficar errada.

🧠 Como evitar Silent Errors

- Usar `===` em vez de `==`
- Ativar `"use strict";`
- Verificar valores `undefined` e `NaN`
- Usar `console.log()` para testar
- Utilizar linters como ESLint

📌 JavaScript – Error Statements

❗ Para que servem?

Permitem **tratar erros sem parar o programa**.

Principais palavras-chave:

- `try`
- `catch`
- `finally`
- `throw`

🔹 try

Bloco onde fica o código que pode gerar erro.

```
try {
  // código perigoso
}
```

Se não houver erro, o `catch` é ignorado.

🔹 catch

Executa apenas se houver erro no `try`.

```
catch (err) {
  // tratamento do erro
}
```

O objeto `err` contém:

- `err.name`
- `err.message`

🔁 finally (opcional)

- Sempre executa
- Com erro ou sem erro
- Usado para limpeza de recursos

```
finally {
  // sempre roda
}
```

🔥 throw – Criar erros manualmente

Usado para **forçar um erro**.

```
throw "Erro!";
throw 404;
```

Pode lançar:

- String
- Number
- Boolean
- Object

☑ Exemplo de validação com throw

```
try {
  if (x === "") throw "vazio";
  if (isNaN(x)) throw "não é número";
  if (x < 5) throw "muito baixo";
  if (x > 10) throw "muito alto";
} catch (err) {
  console.log("Erro: " + err);
}
```

🖌️ Exemplo com finally

```
try {
  // validação
} catch (err) {
  console.log("Erro: " + err);
} finally {
  input.value = "";
}
```

O campo é limpo sempre, mesmo com erro.

📌 JavaScript – Error Reference (Objeto Error)

❗ O que é o objeto Error?

É um objeto interno do JavaScript criado automaticamente quando ocorre um erro.
Ele guarda informações sobre o problema ocorrido.

🧱 Propriedades principais

- `name` → tipo do erro
- `message` → descrição do erro

```
try {
  x = y + 1;
} catch (err) {
  console.log(err.name);
  console.log(err.message);
}
```

🔥 Tipos de Error (name)

- EvalError → obsoleto
- RangeError → valor fora do limite
- ReferenceError → variável inexistente
- SyntaxError → erro de escrita
- TypeError → tipo inválido
- URIError → erro em encodeURI ou decodeURI

⚠️ Propriedades NÃO padronizadas (evitar)

Não funcionam em todos os navegadores:

- stack
- lineNumber
- fileName
- description
- caller
- arguments

☑ Em sites reais, use apenas:

- `name`
- `message`