



IN

INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 13 – TRY CATCH

O QUE IREMOS APRENDER

01

RESUMO DA AULA PASSADA

02

O QUE É TRY CATCH

IN

TRY CATCH

try e **catch** são usados para lidar com exceções ou erros que podem ocorrer durante a execução do seu código.

Em JavaScript, utilizamos **try** e **catch** porque eles permitem lidar com erros e exceções de maneira controlada, evitando que o código pare de funcionar abruptamente.



TRY CATCH

try (tentar): Este bloco envolve o código onde você espera que uma exceção possa ocorrer. Se uma exceção ocorrer dentro do bloco **try**, o controle do programa é transferido para o bloco **catch**

catch (capturar): Este bloco é usado para manipular a exceção capturada. Dentro do bloco **catch**, você pode escrever código para lidar com a exceção, registrar mensagens de erro ou tomar qualquer outra ação apropriada.

TRY CATCH

Quando usamos códigos JavaScript, podem ocorrer diversos tipos de erros na execução das páginas:

- Erros de codificação de comandos (digitação, sintaxe, ..);
- Erros causados por uso indevido de recursos ou ferramentas;
- Erros resultantes de condições do ambiente;
- Erros previstos (ou não) relacionados a regras de negócios.

TRY CATCH

A linguagem JavaScript oferece um conjunto de comandos que permitem detectar e tratar esses erros dentro do próprio código.

Atenção: Alguns tipos de erros são mais facilmente detectados em linguagens compiladas, como C ou Java, pois o compilador acusará o erro no momento da compilação.

Em linguagens de script (não compiladas), é mais comum que alguns erros ocorram durante a execução.

TRY CATCH

```
<script>

try {
    // Código que pode lançar uma exceção
    // Por exemplo:
    // const resultado = algumaFuncao();
    // Se algumaFuncao() lançar uma exceção, o bloco catch irá lidar com ela.
} catch (erro) {
    // Código para lidar com a exceção
    // O parâmetro 'erro' contém informações sobre a exceção lançada
    // Por exemplo:
    // console.error("Ocorreu um erro:", erro.message);
} finally {
    // Código que é executado independentemente de uma exceção ter sido lançada ou não
    // Este bloco é opcional e pode ser omitido
}

</script>
```

TRY CATCH

O código que pode gerar uma exceção é colocado dentro do bloco **try**. No exemplo abaixo, dentro do **try**, ele tenta transformar um número em letra maiúscula, mas essa transformação não é possível. Portanto, o controle vai para o **catch**, que exibe o erro ocorrido.

```
<script>
    let num = 1;
    try {
        num.toUpperCase(); // não tem como usar o toUpperCase em numeros
    }
    catch (err) {
        alert(err.name);
    }
</script>
```

TRY CATCH

Se o erro/exceção ocorrer, o bloco **catch** entra em ação e você faz o tratamento do erro. No exemplo, o **catch** entrará em ação e mostrará o erro.

```
<script>
    let num = 1;
    try {
        num.toUpperCase(); // não tem como usar o toUpperCase em numeros
    }
    catch (err) {
        alert(err.name);
    }
</script>
```

TRY CATCH

Dentro do bloco **finally**, você coloca o código que deverá ser executado sempre que ocorra ou não a exceção. Em outras palavras, o **finally** sempre será executado, independentemente de ocorrer um erro ou não.

```
<script>
  let num = 1
  try {
    num.toUpperCase(); //não tem como usar o toUpperCase em numeros
  }
  catch (err) {
    alert(err.name)
  }
  finally {
    console.log("Terminei o código")
  }
</script>
```

TRY CATCH

```
<script>
  try {
    const valor = 10 / 0; // Isso vai lançar uma exceção de 'Divisão por zero'
    console.log("Valor:", valor); // Esta linha não será executada devido à exceção
  } catch (erro) {
    console.error("Ocorreu um erro:", erro.message); // Saída: Ocorreu um erro: Divisão por zero
  } finally {
    // Saída: Este código é executado independentemente de exceções.
    console.log("Este código é executado independentemente de exceções.");
  }

</script>
```

TRY CATCH

```
<script>
  try {//tente
    const valor = 10 / 2; //10 dividido por 2
    console.log("Valor:", valor); //mostrar no console o valor
  } catch (erro) {
    console.error("Ocorreu um erro:", erro.message);
  } finally {
    console.log("Este código é executado independentemente de exceções.");
  }

</script>
```

TRY CATCH

```
<body>
    <input type="text" id="nome">
    <script>
        try {
            let nome = document.getElementById("nome").value
            cadastroUsuario(nome)
        } catch (e) {
            alert("Algo deu errado " + e.message)
        }

    </script>
</body>
```

TRY CATCH

É possível criar nossos próprios erros para situações específicas e usar os blocos **try/catch** para controlar o fluxo do código. Quando uma exceção é disparada, seja ela criada por nós ou não, dizemos que ela foi lançada (`throw`).

É através do comando **throw** que podemos lançar exceções. Ele interrompe a execução do bloco **try** imediatamente, pulando diretamente para o **catch**.

TRY CATCH

```
<script>
    function dividir(a, b) {
        if (b === 0) {
            throw "Divisão por zero não é permitida.";
        }
        return a / b;
    }

    try {
        const resultado = dividir(10, 0);
        console.log("Resultado:", resultado);
    } catch (erro) {
        console.error("Ocorreu um erro:", erro); // Saída: Ocorreu um erro: Divisão por zero não é permitida.
    }

</script>
```

TRY CATCH

Podemos então instanciar um objeto de erro próprio do JavaScript para facilitar. O objeto **Error** engloba todos os tipos de erros que podem ser lançados.

No entanto, também existem objetos de erros específicos, como **TypeError** (erro de tipo) e **ReferenceError** (erro de referência, chamando uma variável ou função que não existe), entre outros.

TRY CATCH

```
<script>

function validarIdade(idade) {
    if (typeof idade !== 'number' || idade < 0) {
        throw "A idade deve ser um número positivo.";
    }
    if (idade < 18) {
        throw "Você é menor de idade e não pode acessar este conteúdo.";
    }
    console.log("Acesso permitido. Bem-vindo!");
}

try {
    validarIdade(15); // Tente alterar a idade para ver diferentes cenários
} catch (erro) {
    console.error("Ocorreu um erro:", erro); // Aqui, usamos diretamente 'erro' sem acessar 'erro.message'
}

</script>
```

ATIVIDADE PRÁTICA

Atividade 01:

Suponha que você está desenvolvendo um formulário web e deseja validar os dados inseridos pelo usuário. Utilize os blocos **try** e **catch** para garantir que os dados sejam processados corretamente e para mostrar mensagens de erro amigáveis quando ocorrerem erros de validação.

ATIVIDADE PRÁTICA

Atividade 02:

Você está desenvolvendo um aplicativo que responde a eventos do usuário, como cliques em botões. Utilize blocos **try** e **catch** para lidar com exceções que podem ocorrer dentro dos event listeners e garantir que a aplicação continue respondendo.

ATIVIDADE PRÁTICA

Atividade 03:

Crie um formulário de cadastro para uma empresa que tem lojas espalhadas por todo o país. O formulário deve solicitar o nome da loja, a cidade e o valor total vendido no mês. Utilize uma estrutura “**try/catch/throw/finally**” para testar as seguintes regras:

- Se todos os campos estão preenchidos.
- Se o valor total de venda é maior ou igual a zero.

ATIVIDADE PRÁTICA

Atividade 04

Exiba uma mensagem no formulário informando qualquer erro ou regra desobedecida. Ao final da validação, exiba um botão que recarrega a página.

Importante: o botão só deve aparecer ao final da validação (antes disso, deve permanecer oculto).

DESAFIO PRÁTICO

Crie um jogo simples de adivinhação, em que o computador escolhe um número aleatório entre 1 e 10, e o jogador tenta adivinhar o número. Aqui estão os passos:

1. O computador escolhe um número aleatório entre 1 e 10 e o armazena.
2. O programa solicita ao jogador que insira um palpite.

DESAFIO PRÁTICO

(Continuação)

3. Utilize um bloco **try** e **catch** para lidar com a entrada do jogador. Se o jogador inserir algo que não seja um número, exiba uma mensagem indicando que apenas números são aceitos.
4. Se o palpite estiver correto, exiba uma mensagem de parabéns. Se estiver errado, informe ao jogador se o número escolhido pelo computador é maior ou menor que o palpite.

SE LIGA NO CONTEÚDO DA PRÓXIMA AULA!

AULA 14 DE JAVASCRIPT.
CONSUMO DE API



INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Consumo API com Fetch

API é a sigla para

"Interface de Programação de Aplicativo". Ela é como um conjunto de regras e protocolos que permite que diferentes programas de computador se comuniquem e interajam entre si. Imagine uma analogia com um restaurante:

1. Você é o cliente.
2. O garçom é a API.
3. A cozinha é o sistema ou serviço que você deseja usar.



Consumo API com Fetch

Exemplos da API em JavaScript:

```
const apiUrl = 'https://api.exemplo.com/dados'; // Substitua pela URL real da API
fetch(apiUrl)
  .then(response => {
    // Manipule a resposta da API aqui
  })
  .catch(error => {
    // Lide com erros aqui, se houver algum problema com a solicitação
});
```



IN

INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 13 – TRY CATCH