

JavaScript Impressionador - Módulo 7 - Tratamento de Erro

Exercícios

TRY / CATCH:

Exercício 1: Tratamento de Erro em Variável Não Definida Crie um código JavaScript que inclua uma função chamada `tratarErroVariavelNaoDefinida()`. Dentro dessa função, siga estas etapas:

Utilize um bloco `try...catch` para lidar com o erro que ocorre ao tentar acessar a variável `variavelNaoDefinida` (que não foi definida). No bloco `catch`, exiba uma mensagem de erro personalizada usando `console.error()`. Esta mensagem deve informar que ocorreu um erro ao tentar acessar uma variável não definida e incluir o erro capturado na mensagem de erro.

```
//Resposta
function tratarErroVariavelNaoDefinida() {
  try {
    console.log(variavelNaoDefinida);
  } catch (erro) {
    console.error('Erro: A variável não está definida.', erro);
  }
}

tratarErroVariavelNaoDefinida();
```

Exercício 2: Exercício: Tratamento de Erro em Chamada de Função Inexistente Crie um código JavaScript que inclua uma função chamada `tratarErroFuncaoInexistente()`. Dentro dessa função, siga estas etapas:

Utilize um bloco `try...catch` para lidar com o erro que ocorre ao tentar chamar uma função `funcaoInexistente()` que não está definida. Dentro do bloco `try`, tente chamar a função `funcaoInexistente()`. No bloco `catch`, exiba uma mensagem de erro personalizada usando `console.error()`. Esta mensagem deve informar que ocorreu um erro ao tentar chamar uma função que não está definida e incluir o erro capturado na mensagem de erro.

```
//Resposta
function tratarErroFuncaoInexistente() {
  try {
    funcaoInexistente();
  } catch (erro) {
    console.error('Erro: A função não existe.', erro);
  }
}

tratarErroFuncaoInexistente();
```

Exercício 3: Tratamento de Erro ao Acessar Propriedade de null Crie um código JavaScript que inclua uma função chamada `tratarErroAcessarPropriedadeNull()`. Dentro dessa função, siga estas etapas:

Declare uma variável chamada `objeto` e atribua a ela o valor `null`. Tente acessar uma propriedade chamada `propriedade` do objeto `objeto` e armazene o resultado em uma variável. Utilize um bloco `try...catch` para lidar com o erro que ocorre ao tentar acessar a propriedade de um valor `null`. No bloco `catch`, exiba uma mensagem de erro personalizada usando `console.error()` que informe que ocorreu um erro e inclua o erro capturado na mensagem de erro.

```
//Resposta
function tratarErroAcessarPropriedadeNull() {
  try {
    const objeto = null;
    console.log(objeto.propriedade);
  } catch (erro) {
    console.error('Ocorreu um erro:', erro);
  }
}

tratarErroAcessarPropriedadeNull();
```

Exercício 4: Tratamento de Erro em Acesso a Propriedade Inexistente Crie um código JavaScript que inclua uma função chamada `tratarErroPropriedadeInexistente()`. Dentro dessa função, siga estas etapas:

Declare uma variável chamada `objeto` e atribua a ela um objeto vazio `{}`. Utilize um bloco `try...catch` para lidar com o erro que ocorre ao tentar acessar a propriedade inexistente. No bloco `catch`, exiba uma mensagem de erro personalizada usando `console.error()`. Esta mensagem deve informar que ocorreu um erro ao tentar acessar uma propriedade que não existe no objeto e incluir o erro capturado na mensagem de erro.

```
//Resposta
function tratarErroPropriedadeInexistente() {
  try {
    const objeto = {};
    console.log(objeto.propriedadeInexistente.valor);
  } catch (erro) {
    console.error('Erro: A propriedade não existe no objeto.', erro);
  }
}

tratarErroPropriedadeInexistente();
```

Exercício 5: Tratamento de Erro em Conversão de Tipos

Crie um código JavaScript que inclua uma função chamada `tratarErroConversaoTipo()`. Dentro dessa função, siga estas etapas:

Declare uma variável chamada `numero` e atribua a ela uma string que não represente um número válido, por exemplo, "Hashtag". Tente converter a variável `numero` para um número inteiro usando a função `parseInt()`. Após a conversão, verifique se o resultado é NaN (não é um número) usando `isNaN()`. Se o resultado for NaN, lance um erro intencionalmente usando `throw new Error()`. Este erro deve indicar que não foi possível converter a string em número. Utilize um bloco `try...catch` para lidar com o erro que ocorre ao tentar converter a string em um número. No bloco `catch`, exiba a mensagem de erro capturada no console usando `console.error()`.

```
//Resposta
function tratarErroConversaoTipo() {
  try {
    const numero = "Hashtag";
    const resultado = parseInt(numero);
    if (isNaN(resultado)) {
      throw new Error("Erro: Não foi possível converter a string em número.");
    }
    console.log("Número convertido:", resultado);
  } catch (erro) {
    console.error(erro.message);
  }
}

tratarErroConversaoTipo();
```

Exercício 6: Tratamento de Erro em Divisão por Zero

Crie um código JavaScript que inclua uma função chamada `tratarErroDivisaoPorZero()`. Dentro dessa função, siga estas etapas:

Declare duas variáveis chamadas `dividendo` e `divisor` e atribua a elas valores numéricos. Verifique se o valor da variável `divisor` é igual a zero. Se o divisor for igual a zero, lance um erro intencionalmente usando `throw new Error()`. Este erro deve indicar que a divisão por zero não é permitida. Utilize um bloco `try...catch` para lidar com o erro que ocorre ao tentar realizar uma divisão por zero. No bloco `catch`, exiba a mensagem de erro capturada no console usando `console.error()`.

```
//Resposta
function tratarErroDivisaoPorZero() {
  try {
    const dividendo = 10;
    const divisor = 10;
    if (divisor === 0) {
      throw new Error("Erro: Divisão por zero não é permitida.");
    }
    const resultado = dividendo / divisor;
    console.log("Resultado da divisão:", resultado);
  } catch (erro) {
    console.error(erro.message);
  }
}
```

```
}  
  
tratarErroDivisaoPorZero();
```

Exercício 7: Manipulação de Arquivo com Try, Catch e Finally

Crie um código JavaScript que inclua uma função chamada lerArquivo(). Dentro dessa função, siga estas etapas:

Tente ler um arquivo chamado "arquivo.txt" utilizando uma operação fictícia de leitura de arquivo. Utilize um bloco try...catch para lidar com qualquer erro que ocorra durante a tentativa de leitura do arquivo. No bloco try, se a leitura do arquivo for bem-sucedida, exiba o conteúdo do arquivo no console. No bloco catch, exiba uma mensagem de erro personalizada informando que ocorreu um erro durante a leitura do arquivo. Utilize um bloco finally para garantir que determinadas operações (por exemplo, fechar o arquivo, se necessário) sejam executadas, independentemente de ocorrer um erro ou não.

```
//Resposta 1  
function lerArquivoFicticio() {  
  try {  
    // Simulando a leitura de um arquivo (operação fictícia)  
    const conteudoDoArquivo = "Este é o conteúdo do arquivo fictício.";  
    console.log("Conteúdo do arquivo:", conteudoDoArquivo);  
  } catch (erro) {  
    console.error('Erro: Ocorreu um erro durante a leitura do arquivo.', erro);  
  } finally {  
    console.log("Bloco finally executado, encerrando a operação.");  
  }  
}  
  
lerArquivoFicticio();  
  
//Resposta 2  
const fs = require('fs'); // Módulo de sistema de arquivos do Node.js  
  
function lerArquivoComFS() {  
  try {  
    const conteudoDoArquivo = fs.readFileSync('arquivo.txt', 'utf-8');  
    console.log("Conteúdo do arquivo:", conteudoDoArquivo);  
  } catch (erro) {  
    console.error('Erro ao ler o arquivo:', erro.message);  
  } finally {  
    console.log("Bloco finally executado, encerrando a operação.");  
  }  
}  
  
lerArquivoComFS();
```

Exercício 8: Conexão com Banco de Dados com Try, Catch e Finally

Crie um código JavaScript que inclua uma função chamada `conectarBancoDeDados()`. Dentro dessa função, siga estas etapas:

Tente estabelecer uma conexão com um banco de dados fictício utilizando uma operação de conexão fictícia. Utilize um bloco `try...catch` para lidar com qualquer erro que ocorra durante a tentativa de conexão com o banco de dados. No bloco `try`, se a conexão com o banco de dados for bem-sucedida, exiba uma mensagem informando que a conexão foi estabelecida. No bloco `catch`, exiba uma mensagem de erro personalizada informando que ocorreu um erro durante a conexão com o banco de dados. Utilize um bloco `finally` para garantir que a conexão com o banco de dados seja fechada, se necessário, independentemente de ocorrer um erro ou não.

```
//Resposta 1
function conectarBancoDeDadosFicticio() {
  try {
    // Simulando a conexão com um banco de dados (operação fictícia)
    console.log("Conexão com o banco de dados estabelecida.");
  } catch (erro) {
    console.error('Erro: Ocorreu um erro durante a conexão com o banco de dados.',
erro);
  } finally {
    console.log("Bloco finally executado, encerrando a conexão com o banco de
dados.");
  }
}

conectarBancoDeDadosFicticio();

//Resposta 2
let banco = {
  // Simulando registros de um "banco de dados"
  registro1: { id: 1, nome: "Item 1" },
  registro2: { id: 2, nome: "Item 2" },
  registro3: { id: 3, nome: "Item 3" },
};

function conectarBancoDeDadosFake() {
  try {
    // Simulando uma consulta a um registro no "banco de dados"
    const resultado = banco.registro1;
    if (!resultado) {
      throw new Error("Erro: Registro não encontrado.");
    }
    console.log("Registro encontrado:", resultado);
  } catch (erro) {
    console.error('Erro: Ocorreu um erro durante a consulta.', erro.message);
  } finally {
    console.log("Bloco finally executado, encerrando a conexão com o 'banco de
dados'.");
  }
}

conectarBancoDeDadosFake();
```

