

Curso de

# JavaScript

# Resolução --- Exercícios

# AJAX

# AJAX (Asynchronous JavaScript and XML)

É uma forma de fazer a página **HTML** se comunicar com o servidor de forma **assíncrona**, sem que seja preciso recarregar a página para exibir informações.

Significa que o site pode buscar informações em **segundo plano**, para depois atualizar



JS

```
1 const btnAdd = document.getElementById("add");
```



JS

```
1 btnAdd.onclick = () => {  
2   const h1 = document.getElementById("texto");  
3   h1.innerText = "Olá!"  
4};
```



```
1 <button onclick="alert('Você clicou no botão!')>Clique aqui</button>
2 <br><br>
```

**HTML**

```
1 <input id="texto" type="text" placeholder="Digite seu nome"
2 onkeydown="if(event.key === 'Enter'){ alert(`Seu nome é: ${document.getElementById('texto').value}`);}
3 <br><br>
```

**HTML**

```
1 <p>Role a página para disparar um evento de scroll.</p>
2 <p id="contagem"></p>
3
4 <script>
5   let contador = 0;
6
7   window.onscroll = function() {
8     const texto = document.getElementById("contagem")
9     contagem.innerText = `A contagem de scroll está em ${contador}`
10    contador +=1;
11  };
12 </script>
```

**HTML**

```
1 <form onsubmit="alert('Formulário enviado!'); return false;">
2   <input type="text" placeholder="Seu nome" />
3   <button type="submit">Enviar</button>
4 </form>
5 <br><br>
```

**HTML**

# Alguns tipos de Eventos

# Requisições

Requisições são a maneira como o **JavaScript** chama dados que não estão dentro da página **HTML**.

Esses dados podem estar, por exemplo, em um arquivo **JSON** no projeto, ou em uma **API** online.



JS

```
1 function carregarDados() {  
2     const xhr = new XMLHttpRequest();  
3     xhr.open("GET", "dados.json");  
4     xhr.onload = function () {  
5         if (xhr.status === 200) {  
6             const dados = JSON.parse(xhr.responseText);  
7             document.getElementById("conteudo").innerText =  
8                 `Nome: ${dados.nome}, Idade: ${dados.idade}, Cidade: ${dados.cidade}`;  
9         }  
10    };  
11    xhr.send();  
12 }
```



JSON

```
1 {  
2     "nome": "João",  
3     "idade": 25,  
4     "cidade": "São Paulo"  
5 }
```



HTML

```
1 <h1>Aqui será exibido o texto</h1>  
2 <p id="conteudo"></p>  
3  
4 <button onclick="carregarDados()">Carregar dados</button>
```

# Extraindo dados de um arquivo



JSON

```
1 [  
2   { "id": 1, "nome": "Gabriel", "idade": 25 },  
3   { "id": 2, "nome": "Ana", "idade": 30 },  
4   { "id": 3, "nome": "Carlos", "idade": 28 },  
5   { "id": 4, "nome": "Maria", "idade": 22 },  
6   { "id": 5, "nome": "Lucas", "idade": 35 },  
7   { "id": 6, "nome": "Julia", "idade": 27 }  
8 ]  
9
```



JS

```
1 function listarTodos() {  
2   const xhr = new XMLHttpRequest();  
3   xhr.open("GET", "dados.json", true);  
4   xhr.onload = function () {  
5     if (xhr.status === 200) {  
6       const dados = JSON.parse(xhr.responseText);  
7       const lista = document.getElementById("lista");  
8       lista.innerHTML = ""; // limpa antes de listar  
9  
10      dados.forEach(item => {  
11        const li = document.createElement("li");  
12        li.textContent = `ID: ${item.id} | Nome: ${item.nome} | Idade: ${item.idade}`;  
13        lista.appendChild(li);  
14      });  
15    }  
16  };  
17  xhr.send();  
18 }
```

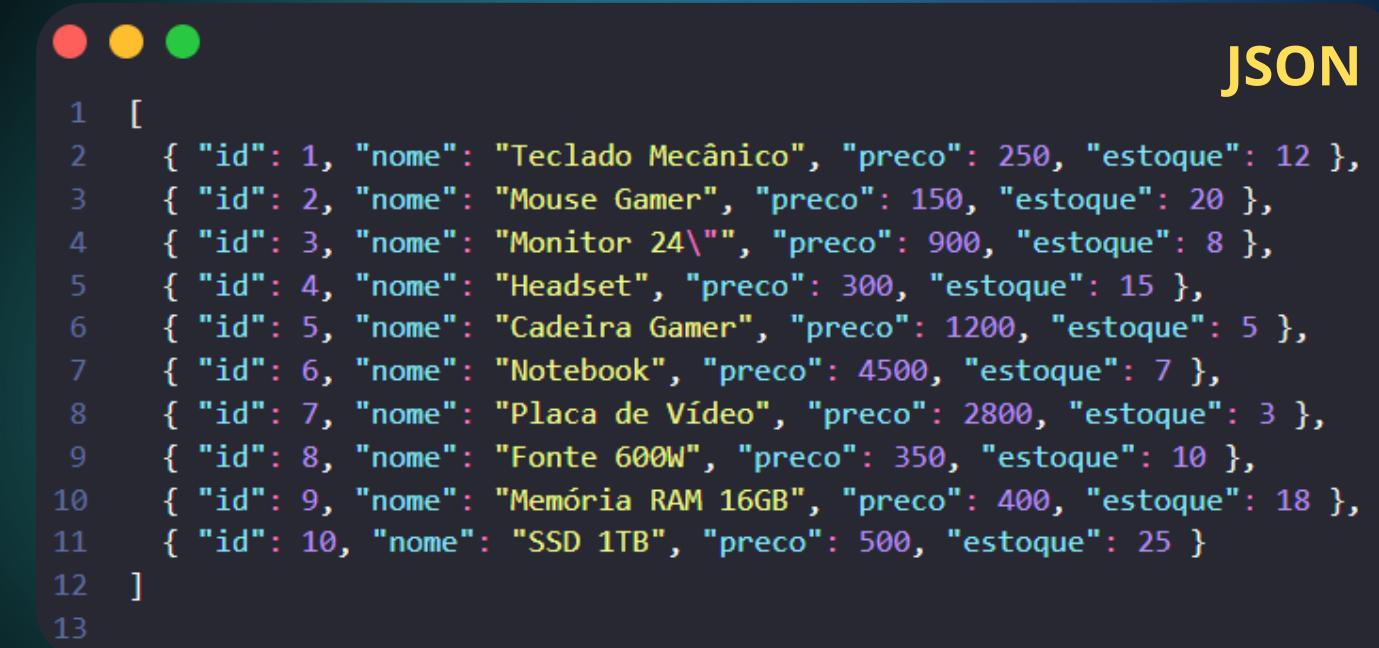
# Exemplo

---

Pesquisando dados em um json

# Importante

Não é possível alterar um arquivo apenas usando **JavaScript**, isso só é possível utilizando outras ferramentas (**Node.Js por exemplo**), por isso, para a nossa aula, sempre que for necessário editar dados, usaremos variáveis para tal.



```
1 [  
2   { "id": 1, "nome": "Teclado Mecânico", "preco": 250, "estoque": 12 },  
3   { "id": 2, "nome": "Mouse Gamer", "preco": 150, "estoque": 20 },  
4   { "id": 3, "nome": "Monitor 24\"", "preco": 900, "estoque": 8 },  
5   { "id": 4, "nome": "Headset", "preco": 300, "estoque": 15 },  
6   { "id": 5, "nome": "Cadeira Gamer", "preco": 1200, "estoque": 5 },  
7   { "id": 6, "nome": "Notebook", "preco": 4500, "estoque": 7 },  
8   { "id": 7, "nome": "Placa de Vídeo", "preco": 2800, "estoque": 3 },  
9   { "id": 8, "nome": "Fonte 600W", "preco": 350, "estoque": 10 },  
10  { "id": 9, "nome": "Memória RAM 16GB", "preco": 400, "estoque": 18 },  
11  { "id": 10, "nome": "SSD 1TB", "preco": 500, "estoque": 25 }  
12 ]  
13
```



```
1 let produtos = [] // aqui ficam os produtos carregados  
2  
3 // --- Carregar produtos do JSON para o array ---  
4 function listarProdutos() {  
5   const xhr = new XMLHttpRequest();  
6   xhr.open("GET", "dados.json", true);  
7   xhr.onload = function () {  
8     if (xhr.status === 200) {  
9       produtos = JSON.parse(xhr.responseText); // guarda em variável  
10      renderizar();  
11    }  
12  };  
13  xhr.send();  
14 }
```

# Exemplo

---

Adicionando um Item na lista

# Promises

As Promises são objetos que representam a eventual conclusão ou falha de uma operação assíncrona.

Ela pode retornar os possíveis resultados:

**pending** (pendente): operação ainda não concluída

**fulfilled** (resolvida): operação concluída com sucesso

**rejected** (rejeitada): operação falhou



# Criando e usando uma promise



JS

```
1 // Promise básica
2 const minhaPromise = new Promise((resolve, reject) => {
3     setTimeout(() => {
4         const sucesso = Math.random() > 0.5;
5
6         if (sucesso) {
7             resolve('Operação bem-sucedida!');
8         } else {
9             reject(new Error('Algo deu errado!'));
10        }
11    }, 1000);
12});
```



JS

```
1 // Usando a promise
2 minhaPromise
3     .then(resultado => {
4         console.log('Sucesso:', resultado);
5     })
6     .catch(erro => {
7         console.error('Erro:', erro);
8     });
9
```

# Promise em AJAX

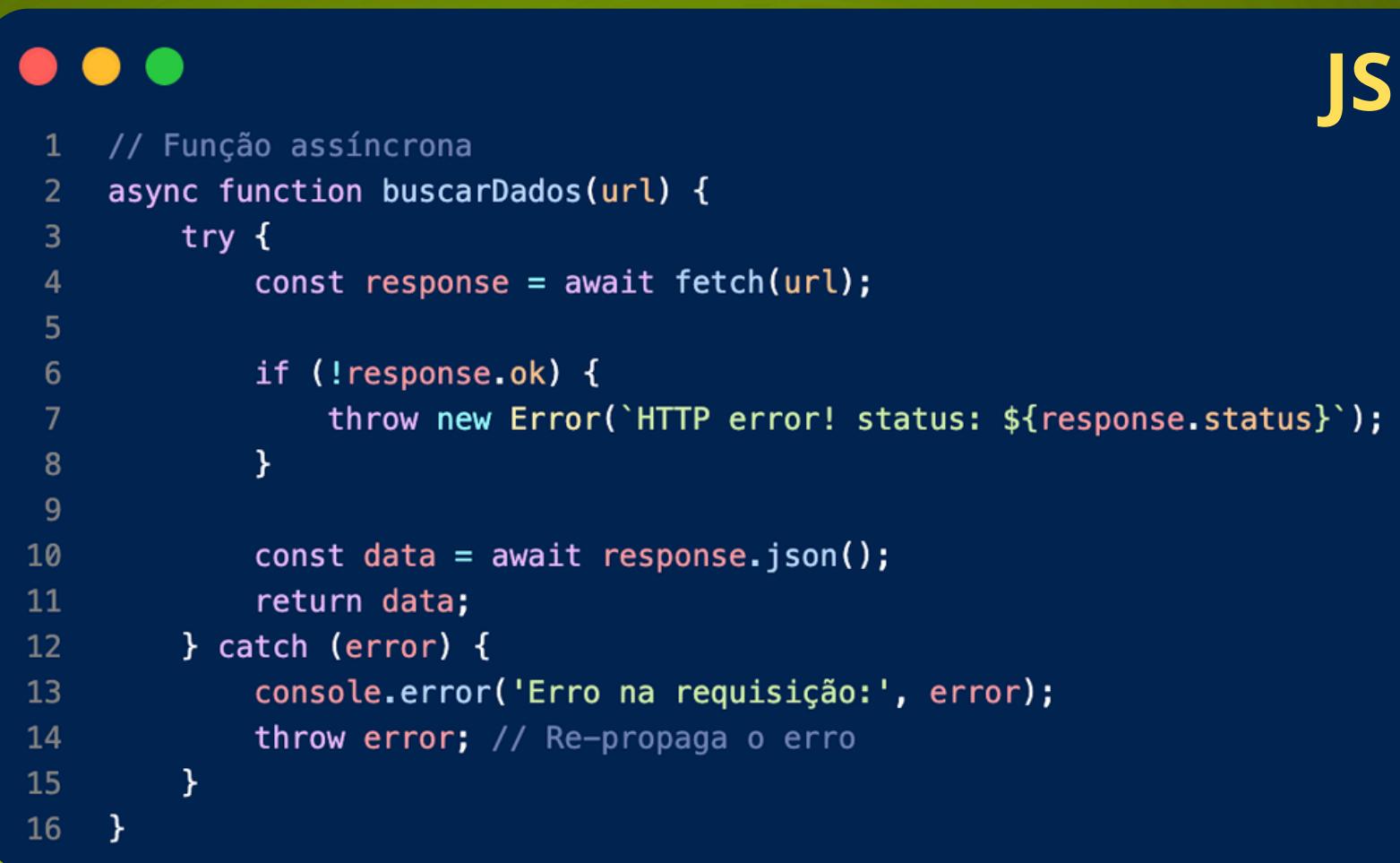
**JS**

```
1 function fazerRequisicaoPromise(url) {  
2     return new Promise((resolve, reject) => {  
3         const xhr = new XMLHttpRequest();  
4  
5         xhr.open('GET', url, true);  
6  
7         xhr.onload = function() {  
8             if (xhr.status >= 200 && xhr.status < 300) {  
9                 resolve(xhr.responseText);  
10            } else {  
11                reject(new Error(`Erro HTTP: ${xhr.status}`));  
12            }  
13        };  
14  
15        xhr.onerror = function() {  
16            reject(new Error('Erro de rede'));  
17        };  
18  
19        xhr.send();  
20    });  
21}
```

**JS**

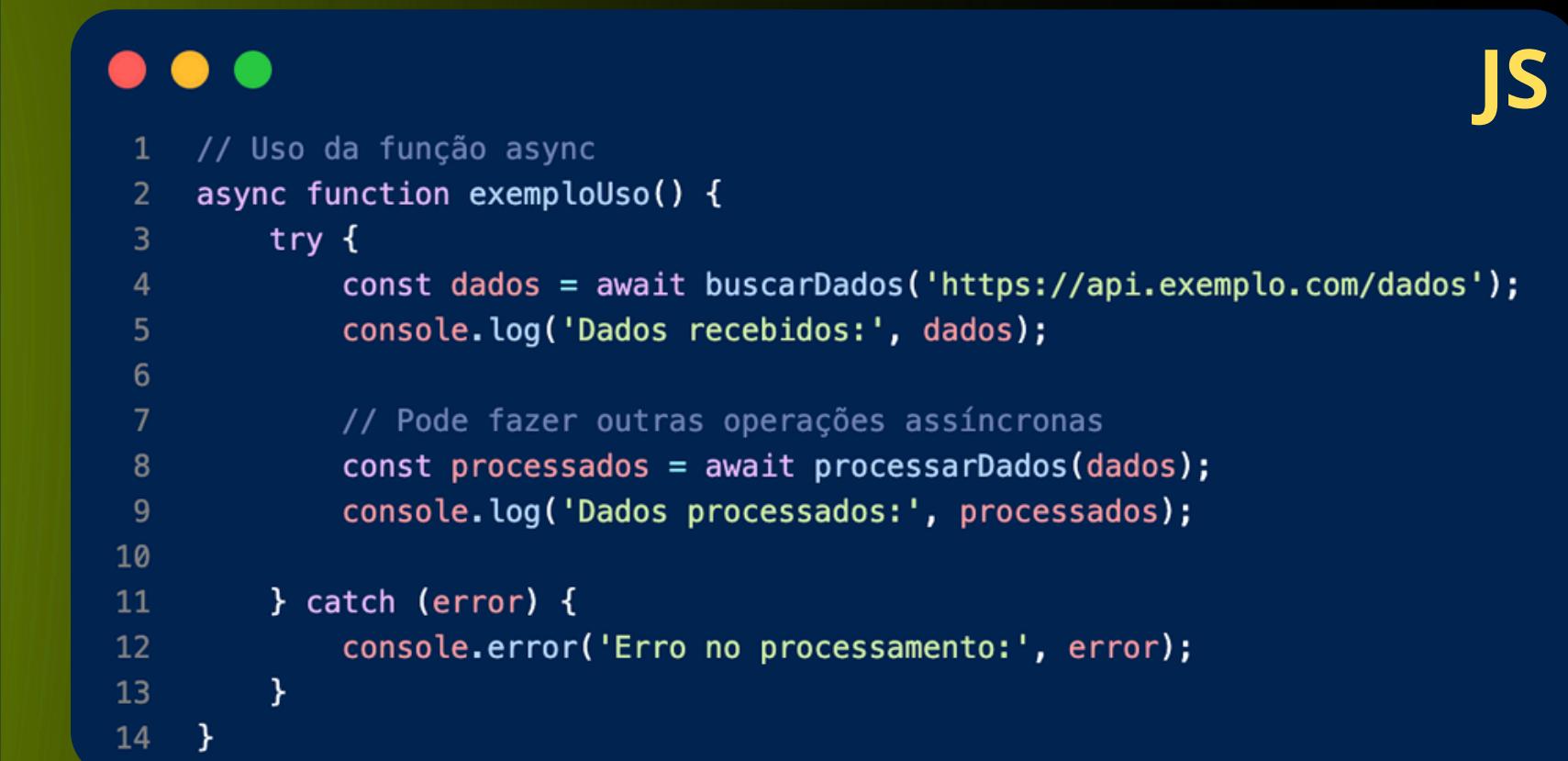
```
1 // Uso da função com promise  
2 fazerRequisicaoPromise('https://api.exemplo.com/dados')  
3     .then(dados => {  
4         console.log('Dados:', dados);  
5         return JSON.parse(dados); // Retorna para o próximo .then()  
6     })  
7     .then(dadosJSON => {  
8         console.log('Dados processados:', dadosJSON);  
9     })  
10    .catch(erro => {  
11        console.error('Erro capturado:', erro);  
12    });
```

# Promise com Async/Await (moderno)



JS

```
1 // Função assíncrona
2 async function buscarDados(url) {
3     try {
4         const response = await fetch(url);
5
6         if (!response.ok) {
7             throw new Error(`HTTP error! status: ${response.status}`);
8         }
9
10        const data = await response.json();
11        return data;
12    } catch (error) {
13        console.error('Erro na requisição:', error);
14        throw error; // Re-propaga o erro
15    }
16}
```



JS

```
1 // Uso da função async
2 async function exemploUso() {
3     try {
4         const dados = await buscarDados('https://api.exemplo.com/dados');
5         console.log('Dados recebidos:', dados);
6
7         // Pode fazer outras operações assíncronas
8         const processados = await processarDados(dados);
9         console.log('Dados processados:', processados);
10    } catch (error) {
11        console.error('Erro no processamento:', error);
12    }
13}
14}
```

# Exemplo

---

Chamando dados de uma API real

# **Exercício**

---

**Exercício já disponível na pasta “Aula 5” na branch master**

# chamada

---

