# FUNÇÕES SICRONAS E ASSICRONAS

Funções Síncronas

- Executam uma instrução por vez, de forma sequencial.

- A próxima instrução só é executada após a anterior ser concluída.

Exemplo Síncrono:

javascript

function funcaoSincrona() {

console.log('Primeiro');

console.log('Segundo');

}

funcaoSincrona();

// Saída:

// Primeiro

// Segundo

Neste exemplo, os comandos são executados um após o outro.

Funções Assíncronas

- Não bloqueiam a execução. Elas permitem que outras instruções sejam executadas enquanto esperam por uma operação (como requisição a um servidor) ser concluída.

- Utilizam funções como `setTimeout`, `promises`, ou `async/await`.

Exemplo Assíncrono com `setTimeout`:

javascript

function funcaoAssincrona() {

console.log('Primeiro');

setTimeout(() => {

console.log('Terceiro');

}, 1000);

console.log('Segundo');

}

funcaoAssincrona();

// Saída:

// Primeiro

// Segundo

// Terceiro (após 1 segundo)

Aqui, o `setTimeout` não bloqueia o código. Ele espera 1 segundo antes de executar, enquanto o código segue executando.

Comparativo com `Promise`:

javascript

function promessa() {

return new Promise((resolve) => {

setTimeout(() => resolve('Terceiro'), 1000);

});

}

function funcaoAssincronaComPromise() {

console.log('Primeiro');

promessa().then((resultado) => console.log(resultado));

console.log('Segundo');

}

funcaoAssincronaComPromise();

// Saída:

// Primeiro

// Segundo

// Terceiro (após 1 segundo)

Neste caso, usamos uma `Promise` que resolve após 1 segundo, sem bloquear a execução das outras instruções.

Aqui está um exemplo de como usar o `fetch` de forma assíncrona em JavaScript, utilizando `async/await`:

Exemplo Assíncrono com `fetch`:

javascript

async function fetchDados() {

console.log('Iniciando a requisição...');

try {

const response = await fetch('https://jsonplaceholder.typicode.com/posts/1');

const dados = await response.json();

console.log('Dados recebidos:', dados);

} catch (erro) {

console.error('Erro na requisição:', erro);

}

console.log('Requisição finalizada');

}

fetchDados();

Saída:

Iniciando a requisição...

Dados recebidos: {userId: 1, id: 1, title: "...", body: "..."}

Requisição finalizada

Explicação:

- O código usa `async/await` para aguardar a resposta do `fetch`.

- O `fetch` é assíncrono, então ele não bloqueia a execução enquanto espera os dados.

- O `try/catch` captura erros que podem ocorrer na requisição, como falha de conexão.