<LAB365>

Arrays

AGENDA | M1S06 - A1

- Arrays
  - O que são?
  - Criação e manipulação
  - Métodos avançados

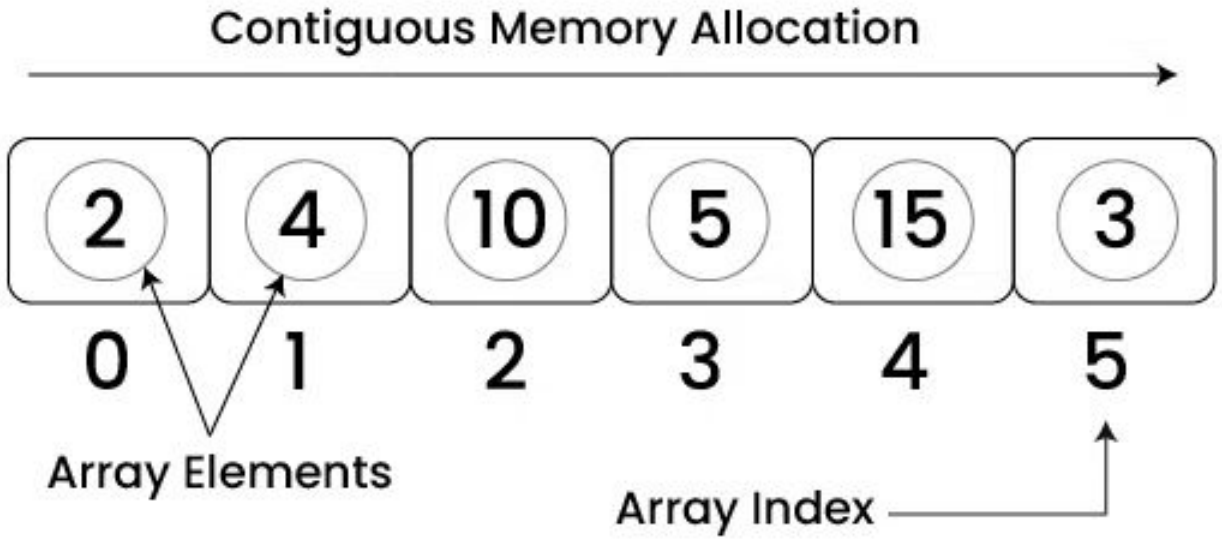
ARRAYS

Um array é uma estrutura de dados fundamental que armazena uma **coleção** ordenada de elementos. Ele permite agrupar **múltiplos valores** sob um único nome de variável, facilitando o gerenciamento e a manipulação de dados.

Características Principais

- **Indexação**: Os elementos em um array são acessados por meio de índices numéricos, geralmente começando em 0 (zero-based).
- **Tamanho Dinâmico**: Em muitas linguagens (como JavaScript), os arrays podem crescer ou diminuir conforme necessário.
- **Homogeneidade vs. Heterogeneidade**: Em linguagens dinâmicas (como JavaScript ou Python), arrays podem armazenar diferentes tipos de dados.
- **Alocação Contígua**: Na memória, os elementos são armazenados em posições sequenciais, permitindo acesso rápido por índice.

ARRAYS


Contiguous Memory Allocation
2
4
10
5
15
3
0
1
2
3
4
5
Array Elements
Array Index

ARRAYS

Para iniciarmos precisamos ver as opções que temos em relação a criação de arrays no JavaScript.

```
// Formas de criar arrays
const frutas = ['maçã', 'banana', 'laranja']; // Literal (recomendado)
const numeros = new Array(1, 2, 3); // Construtor - POO
const vazio = []; // Array vazio
```

ARRAYS

Já vimos como criar os arrays, agora temos também de saber como **acessar** cada elemento dentro do mesmo e fazer manipulações, nesses conceitos temos funções de adição, exclusão e acesso de elementos de um array.

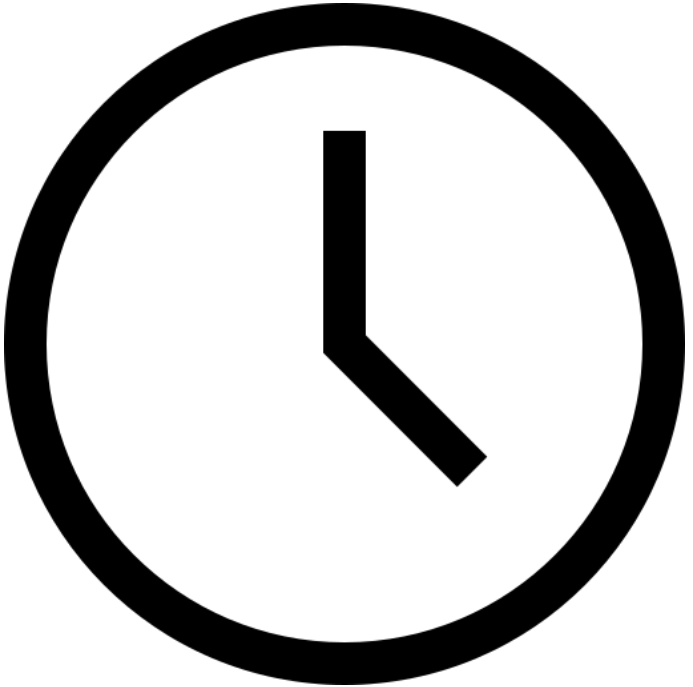
```
let primeiro = frutas[0]; // 'maçã'
//Acessar elemento
let ultimo = frutas[frutas.length - 1]; // 'laranja'


console.log("Primeiro:", primeiro, "Ultimo:", ultimo);

frutas[1] = 'pera'; // Substitui 'banana' por 'pera'
frutas.push('manga'); // Adiciona no final
frutas.pop(); // Remove do final
frutas.unshift('morango'); // Adiciona no início
frutas.shift(); // Remove do início

primeiro = frutas[0]; // 'maçã'
//Acessar elemento
ultimo = frutas[frutas.length - 1]; // 'laranja'

console.log("Primeiro:", primeiro, "Ultimo:", ultimo);
```

INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:10
Retorno: 20:30



ARRAYS: MÉTODOS AVANÇADOS

Existem três métodos são fundamentais para manipulação de arrays em JavaScript, seguindo o paradigma de programação funcional. Eles permitem transformar, filtrar e consolidar dados de forma limpa e eficiente, sem modificar o array original (imutabilidade).

São eles:

- **map**: Cria um novo array aplicando uma função a cada elemento do array original.
- **filter**: Cria um novo array apenas com os elementos que passam em um teste (função de filtro).
- **reduce**: Processa o array e retorna um único valor acumulado, aplicando uma função redutora.

ARRAYS: MÉTODOS AVANÇADOS

```
const novoArrayMap = arrayOriginal.map((elemento, índice, array) => {
    return novoValor;
});

const novoArrayFilter = arrayOriginal.filter((elemento, índice, array) => {
return condição; // true (inclui) ou false (exclui)
});

const resultado = arrayOriginal.reduce((acumulador, elemento, índice, array) => {
return novoAcumulador;
}, valorInicial);
```

EXEMPLO

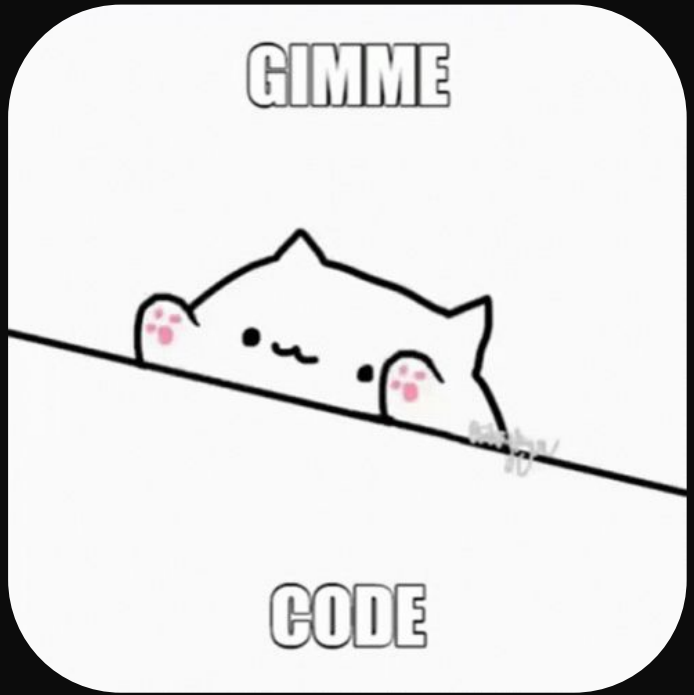
```
const produtos = [
    { id: 1, nome: 'Notebook', preco: 2500, estoque: true },
    { id: 2, nome: 'Tablet', preco: 1200, estoque: false },
    { id: 3, nome: 'Celular', preco: 1800, estoque: true },
    { id: 4, nome: 'Monitor', preco: 900, estoque: true },
    { id: 5, nome: 'Teclado', preco: 150, estoque: false }
];

// 1. Filtra apenas produtos em estoque (filter)
// 2. Extrai apenas os preços (map)
// 3. Soma todos os preços (reduce)

const valorTotalEstoque = produtos
  .filter(produto => produto.estoque)          // Filtra produtos com estoque true
  .map(produto => produto.preco)               // Cria array só com preços
  .reduce((total, preco) => total + preco, 0); // Soma todos os preços

console.log(`Valor total em estoque: R$${valorTotalEstoque.toFixed(2)}`);
// Saída: "Valor total em estoque: R$5200.00"
```

TREINANDO NOSSAS HABILIDADES!

Vamos desenvolver um sistema para cadastramento de assentos em aviões.

Você deve usar arrays para simular os assentos de um avião e fazer uma lógica para que a pessoa escolha a poltrona e seja feita uma verificação se a mesma está disponível.

- Estando disponível a pessoa deve ser cadastrada como “locatária” da poltrona;
- Caso não esteja deve aparecer uma sugestão de poltrona livre e solicitar um número novo (fazer a verificação para saber se está livre, etc...);
- Também precisaremos um uma opção para verificar quantas poltronas estão ocupadas.



<LAB365>

<LAB365>

JavaScript: Módulos, Assincronicidade e Browser API

AGENDA | M1S06 - A2

- Módulos
- Browser API
- Assincronicidade

MÓDULOS

Os módulos no JavaScript permitem **dividir o código** em arquivos separados, organizando a aplicação em partes **menores e reutilizáveis**. Eles são essenciais para projetos grandes, pois **facilitam a manutenção, evitam conflitos de nomes e melhoram a legibilidade do código**.

Principais vantagens dos módulos:

- **Organização do código** – Dividir funcionalidades em arquivos distintos (ex: auth.js, utils.js, api.js).
- **Escopo isolado** – Variáveis e funções não vazam para o escopo global, evitando conflitos.
- **Reutilização** – Exportar funções, classes ou objetos para uso em diferentes partes do projeto.
- **Manutenção facilitada** – Alterações em um módulo não afetam outros, desde que a interface (export/import) seja mantida.
- **Carregamento eficiente** – Módulos podem ser carregados sob demanda (lazy loading), melhorando performance.

MÓDULOS

Basicamente quando estamos falando de módulo temos de utilizar o **export e import**, são as palavras reservadas para estarmos disponibilizando o módulo criado e após isso utilizando o mesmo em outra parte do nosso projeto.

```
export function soma(a, b) { return a + b; }
export const PI = 3.14;
```

```
import { soma, PI } from './calculos.js';
console.log(soma(2, 3));
```

EXEMPLO

```html
<!DOCTYPE html>
<html>
<head>
  <title>Lista Tarefas Simples</title>
</head>
<body>
  <h1>Minhas Tarefas</h1>
  <input type="text" id="task-input" placeholder="Digite uma tarefa">
  <button id="add-btn">Adicionar</button>
  <ul id="task-list"></ul>

  <script type="module" src="app.js"></script>
</body>
</html>
```

EXEMPLO

```
let tasks = [];

export function addTask(text) {
  if (text.trim() === '') return null;

  tasks.push(text);
  return text;
}

export function removeTask(text) {
    tasks = tasks.filter(task => task !== text);
}

export function getTasks() {
  return tasks;
}
```

EXEMPLO

```
import { addTask, removeTask, getTasks} from './tarefas.js';

const taskInput = document.getElementById('task-input');
const addBtn = document.getElementById('add-btn');
const taskList = document.getElementById('task-list');

// Atualiza a lista na tela
function updateTaskList() {
  taskList.innerHTML = '';

  getTasks().forEach(task => {
    const li = document.createElement('li');
    li.innerHTML = `
      <span>${task}</span>
      <button class="delete">X</button>
    `;

    // Evento para deletar
    li.querySelector('.delete').addEventListener('click', () => {
      removeTask(task);
      updateTaskList();
    });

    taskList.appendChild(li);
  });
}
```

```
// Adiciona nova tarefa
addBtn.addEventListener('click', () => {
  if (addTask(taskInput.value)) {
    taskInput.value = '';
    updateTaskList();
  }
});

// Permite adicionar com Enter
taskInput.addEventListener('keypress', (e) => {
  if (e.key === 'Enter') {
    addBtn.click();
  }
});

// Inicia a lista
updateTaskList();
```

BROWSER APIS

Browser APIs (Application Programming Interfaces) são conjuntos de ferramentas e funcionalidades fornecidas pelos **navegadores** que permitem ao JavaScript interagir com diversos aspectos do ambiente do navegador e do sistema do usuário. Elas funcionam como intermediárias entre o código JavaScript e as capacidades nativas do navegador.

Suas principais categorias são:

- APIs de Manipulação do Documento;
- APIs de Comunicação;
- APIs de Armazenamento;
- APIs de Multimídia;
- APIs de Hardware;
- APIs de Desempenho.

BROWSER APIS: LOCAL STORAGE

O localStorage é uma API de **armazenamento** no navegador que permite salvar dados persistentes (que não são apagados quando o navegador é fechado).

Características:

- Armazena dados como pares chave-valor (sempre strings)
- Capacidade de ~5MB
- Dados permanecem após fechar o navegador
- Acesso apenas no mesmo domínio (segurança)

```
localStorage.setItem('chave', 'valor'); // Salva um item
const valor = localStorage.getItem('chave'); // Recupera um item
localStorage.removeItem('chave'); // Remove um item
localStorage.clear(); // Limpa todo o armazenamento
```

EXEMPLO

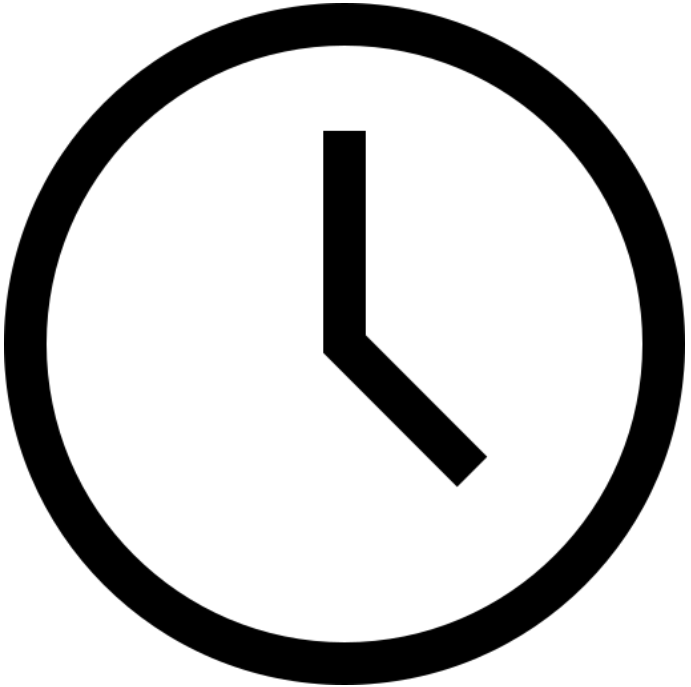
```
// Salvar preferência do tema
function salvarTemaPreferido(tema) {
    localStorage.setItem('temaPreferido', tema);
}

// Carregar tema ao iniciar a página
function carregarTema() {
    const tema = localStorage.getItem('temaPreferido') || 'claro';
    //aplicar o tema
}

// Exemplo de uso
salvarTemaPreferido('escuro');
carregarTema(); // Aplica o tema escuro
```

INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:20
Retorno: 20:40



BROWSER APIS: SETTIMEOUT E SETINTERVAL

setTimeout é uma função que executa uma ação **uma única vez após um tempo determinado**.

Características:

- Executa o código apenas uma vez
- O tempo é contado a partir do momento da chamada
- Retorna um ID que pode ser usado para cancelar

Já o setInterval é uma função que **executa uma ação repetidamente**, com um **intervalo fixo** entre execuções.

Características:

- Executa o código repetidamente no intervalo especificado
- A primeira execução ocorre após o intervalo, não imediatamente
- Retorna um ID que pode ser usado para parar

BROWSER APIS: SETTIMEOUT E SETINTERVAL

Diferenças Principais

Característica	setTimeout	setInterval
Execução	Uma única vez	Repetida
Tempo	Delay antes da execução	Intervalo entre execuções
Primeira execução	Após o delay especificado	Após o primeiro intervalo
Uso típico	Atrasos únicos	Animações, atualizações periódicas
Cancelamento	clearTimeout(id)	clearInterval(id)
Acumulação	Não ocorre	Pode causar acúmulo se o código for lento

EXEMPLO

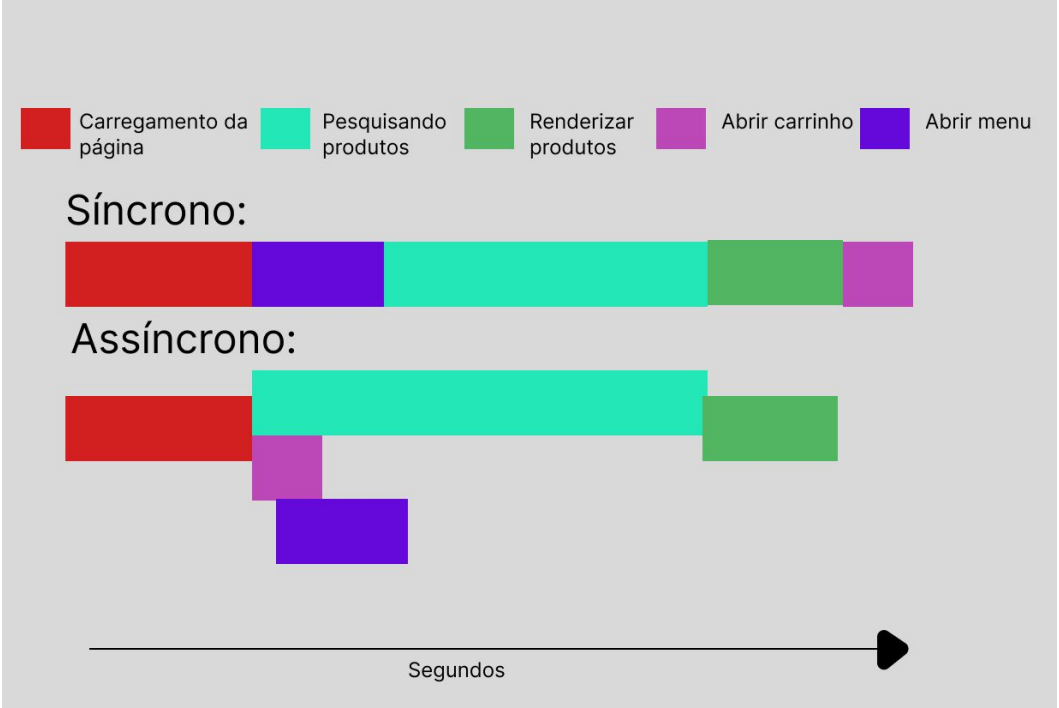
```
// Exibir mensagem após 3 segundos
setTimeout(() => {
    console.log('Esta mensagem aparece após 3 segundos');
}, 3000);

// Com parâmetros
function saudacao(nome) {
    console.log(`Olá, ${nome}!`);
}
setTimeout(saudacao, 2000, 'Maria'); // "Olá, Maria!" após 2 segundos
```

EXEMPLO

```
let segundos = 0;

// Atualizar cronômetro a cada segundo
const cronometro = setInterval(() => {
  segundos++;
  console.log(`Tempo decorrido: ${segundos} segundos`);
}, 1000);

// Parar após 10 segundos
setTimeout(() => {
  clearInterval(cronometro);
  console.log('Cronômetro parado!');
}, 10000);
```

ASSINCRONICIDADE

JavaScript é uma linguagem single-threaded, o que significa que **executa uma operação de cada vez**. Para lidar com operações que podem levar tempo (como requisições de rede, leitura de arquivos, etc.) o JavaScript usa conceitos de assincronicidade.



ASSINCRONICIDADE: PROMISES

Na abordagem tradicional eram usados callbacks para operações assíncronas, mas uma problema que normalmente ocorria era o **Callback Hell** (pirâmide de callbacks aninhados), que dificulta a leitura e manutenção.

Promises foram introduzidas no ES6 para **resolver o problema** do Callback Hell e consequentemente tomaram o lugar dessas utilizações de callback.

```
const minhaPromise = new Promise((resolve, reject) => {
    // Operação assíncrona
    setTimeout(() => {
        const sucesso = true; // Simulando sucesso/falha
        if (!sucesso) {
            resolve('Dados obtidos com sucesso');
        } else {
            reject('Erro ao obter dados');
        }
    }, 1000);
});
```

```
minhaPromise
    .then((resultado) => {
        console.log(resultado); // Se resolvida
    })
    .catch((erro) => {
        console.error(erro); // Se rejeitada
    });
```

ASSINCRONICIDADE: ASYNC/AWAIT

Async/Await é uma forma moderna e elegante de trabalhar com código assíncrono em JavaScript, introduzida no ES2017 (ES8). Ele é construído sobre Promises, mas oferece uma sintaxe mais limpa e intuitiva.

Quando você declara uma função com async, ela automaticamente retorna uma Promise.

```
async function minhaFuncao() {
    return 42;
}

// Equivalente a:
function minhaFuncao() {
    return Promise.resolve(42);
}
```

ASSINCRONICIDADE: ASYNC/AWAIT

Dentro de uma função async, você pode usar await para pausar a execução até que uma Promise seja resolvida.

:

```
async function exemplo() {
    const resultado = await algumaPromise();
    console.log(resultado);
}
```

EXEMPLO

```
// Banco de dados de tarefas como arrays simples [id, título, completado]
let tasks = [
    [1, "Aprender JavaScript", 0],
    [2, "Praticar Async/Await", 0],
    [3, "Criar projeto real", 1]
];

// Função para simular delay de rede
function wait(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
}

// 1. Buscar todas as tarefas (assíncrono)
async function getTasks() {
    await wait(500); // Simula delay de rede
    return tasks;
}

// 2. Adicionar nova tarefa (assíncrono)
async function addTask(title) {
    await wait(300);
    const newId = tasks.length > 0 ? tasks[tasks.length - 1][0] + 1 : 1;
    tasks.push([newId, title, 0]);
    return newId;
}
```

```
// 3. Marcar tarefa como completa (assíncrono)
async function completeTask(id) {
    await wait(200);
    for (let i = 0; i < tasks.length; i++) {
        if (tasks[i][0] === id) {
        tasks[i][2] = 1;
        return true;
        }
    }
    return false;
}
```

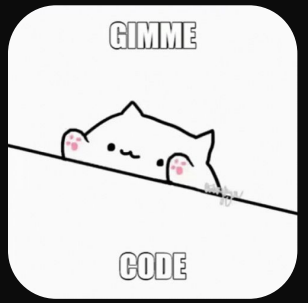
EXEMPLO

```
// 4. Função principal que usa as outras
async function main() {
console.log("Sistema de Tarefas Simples");

// Mostrar tarefas atuais
console.log("\nTarefas iniciais:");
const initialTasks = await getTasks();
console.log(initialTasks);

// Adicionar nova tarefa
console.log("\nAdicionando tarefa...");
const newId = await addTask("Estudar Promises");
console.log(`Tarefa adicionada com ID: ${newId}`);
```

```
// Mostrar tarefas atualizadas
console.log("\nTarefas após adição:");
console.log(await getTasks());

// Completar tarefa
console.log("\nCompletando tarefa...");
const success = await completeTask(2);
console.log(success ? "Tarefa completada!" : "Tarefa não encontrada");

// Resultado final
console.log("\nTarefas finais:");
console.log(await getTasks());
}

// Executar o programa
main().catch(err => console.error("Erro:", err));
```

TREINANDO NOSSAS HABILIDADES!

Você foi contratado para desenvolver um sistema de quiz interativo que será executado no console. O programa deve testar conhecimentos gerais dos usuários através de perguntas objetivas, fornecendo feedback imediato e um relatório final de desempenho.



Requisitos Técnicos

Estrutura de Dados:

- Todas as perguntas e respostas devem ser armazenadas em arrays bidimensionais no formato [pergunta, resposta]
- A pontuação deve ser armazenada em um array no formato [acertos, erros]
- Proibido uso de objetos ou classes (POO)

Funcionalidades Obrigatórias:

- Sistema deve simular carregamento assíncrono das perguntas
- Cada resposta deve ser validada com simulação de processamento assíncrono
- Interface interativa via console (input/output)
- Relatório final com estatísticas de desempenho

AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

Clique aqui ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.

<LAB365>