

## GUIA 06 – JavaScript: Estruturas de Dados

### Objetivos

- Utilizar vetores para organizar e manipular dados em JavaScript.
- Criar e manipular classes e objetos.
- Armazenar e recuperar dados de forma persistente com localStorage.

### JavaScript: vetor (ou array)

Um vetor (ou array) em JavaScript é uma estrutura unidimensional usada para armazenar uma sequência de elementos, como números, textos ou objetos. Ele é útil quando precisamos lidar com múltiplos dados de forma organizada.

Por exemplo, suponha que desejamos armazenar informações de várias pessoas, como nome e idade. Após inserir esses dados em um vetor, podemos percorrê-lo para identificar e exibir os nomes das pessoas cuja idade está acima da média do grupo.

```
var idades = [];  
var nomes = [];  
var posicao = 0;  
var media = 0;  
var total = 0;  
  
while(posicao < 10){  
    nomes[posicao] = prompt("Informe o nome da pessoa " + (posicao + 1) + ":");  
    idades[posicao] = parseInt(prompt("Informe a idade de " + nomes[posicao]));  
    total = total + idades[posicao];  
    posicao++;  
}  
  
media = total / 10;  
  
for(posicao = 0; posicao < 10; posicao++){  
    if(idades[posicao] > media){  
        alert("Nome: " + nomes[posicao])  
    }  
}
```

Arquivo: Código JS do Exemplo 01

	var idades = [];									
valores	30	15	20	10	25	32	14	13	30	14
posições	0	1	2	3	4	5	6	7	8	9
	var nomes = [];									
valores	Maria	José	Ana	Pedro	Aline	Lara	Iara	João	Paulo	Flávia
posições	0	1	2	3	4	5	6	7	8	9

Figura 01: Ilustração do vetor preenchido

### Recursos utilizadas no Exemplo 01

- **prompt:** Exibe uma mensagem e um caixa de entrada (input) que permite ao usuário digitar uma informação.
  - Documentação: <https://developer.mozilla.org/pt-BR/docs/Web/API/window/prompt>
- **alert:** Exibe uma mensagem de aviso e um botão de confirmação (OK).
  - Documentação: <https://developer.mozilla.org/pt-BR/docs/Web/API/Window/alert>

JavaScript: classes e objetos

O exemplo anterior adota uma abordagem baseada em vetores paralelos, que não se alinha aos princípios da programação orientada a objetos. Nesse modelo, os dados de uma mesma entidade — como o nome e a idade de uma pessoa — são armazenados separadamente, o que dificulta a coesão e a manutenção do código. Uma alternativa mais adequada seria utilizar uma classe para representar a pessoa, encapsulando seus atributos e comportamentos em uma única estrutura. Isso torna o código mais organizado, legível e fácil de expandir. Por exemplo:

```
class Pessoa {
  constructor(nome, idade) {
    this.nome = nome;
    this.idade = idade;
  }
}

var pessoas = [];
var total = 0;
var media = 0;

for (let p = 0; p < 10; p++) {
  let nome = prompt("Informe o nome da pessoa "+ (p + 1)+ ":");
  let idade = parseInt(prompt("Informe a idade de " + nome));

  total += idade;

  let pessoa = new Pessoa(nome, idade);
  pessoas.push(pessoa)
}

media = total / pessoas.length;

pessoas.forEach(pessoa => {
  if(pessoa.idade > media){
    alert("Nome: " + pessoa.nome)
  }
});

ou

for (let i = 0; i < pessoas.length; i++) {
  if (pessoas[i].idade > media) {
    alert("Nome: " + pessoas[i].nome);
  }
}
```

Arquivo: Código JS do Exemplo 02

Recursos utilizadas nos Exemplo 02 e 03

Função/Propriedade	Descrição
Push()	Adiciona um ou mais elementos ao final do array
forEach()	Executa uma função para cada elemento do array
length	Retorna o número de elementos do array
pop()	Remove o último elemento do array
shift()	Remove o primeiro elemento do array
filter()	Cria um novo array com valores que atendem a uma condição
find()	Retorna o primeiro elemento que satisfaz a condição
indexOf()	Retorna o índice do valor (ou -1 se não encontrar)
some()	Verifica se um elemento está presente no array (true/false)

- Em JavaScript, **var** tem escopo limitado à função onde é declarado, enquanto **let** é visível apenas dentro do bloco {} onde foi definido.
- O símbolo => que aparece no Exemplo 02 é chamado de **arrow function** (ou função de seta) no JavaScript. É uma forma mais curta e moderna de escrever funções anônimas.

Arrow function	Função anônima
pessoas.forEach(pessoa => { alert(pessoa.nome); });	pessoas.forEach(function(pessoa) { alert(pessoa.nome); });

## JavaScript: localStorage

O localStorage é uma funcionalidade do JavaScript que permite armazenar dados localmente no navegador do usuário, de forma persistente — ou seja, os dados permanecem salvos mesmo após o usuário fechar o navegador ou recarregar a página.

Para armazenar objetos ou vetores no localStorage, é necessário convertê-los para o formato JSON (JavaScript Object Notation), uma representação textual simples e eficiente de estruturas de dados.

- Objeto na estrutura JSON

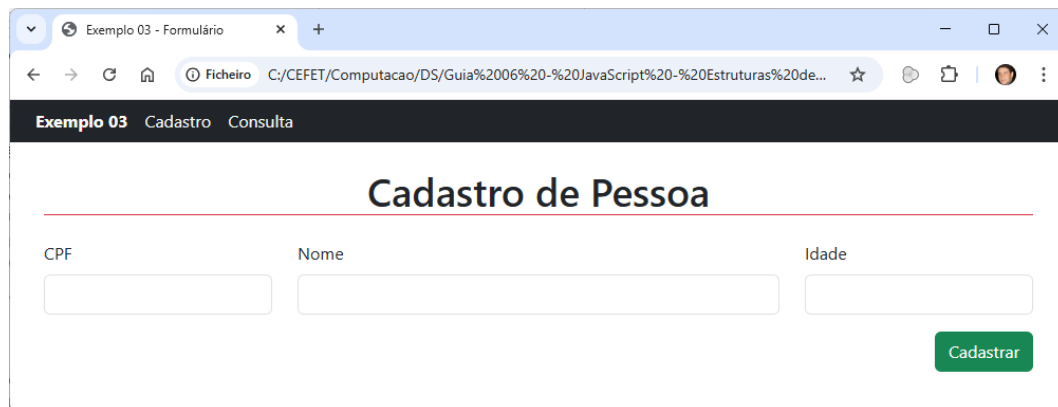
```
{nome: "Maria", idade: 33}
```

- Vetor de objetos na estrutura JSON

```
[  
  { nome: "Maria", idade: 33},  
  { nome: "José", idade: 43}  
]
```

Como o localStorage só armazena dados no formato de texto (string), é necessário utilizar as funções abaixo para manipular objetos ou vetores:

- **JSON.stringify()**: converte um objeto ou vetor JavaScript em uma string no formato JSON.
- **JSON.parse()**: converte uma string JSON de volta para um objeto ou vetor JavaScript.



Exemplo 03 - Formulário

Cadastro Consulta

### Cadastro de Pessoa

CPF Nome Idade

Figura 02: Visualizar do Exemplo 03 – Cadastro



Exemplo 03 - Tabela

Cadastro Consulta

### Consulta de Pessoas

CPF	Nome	Idade
111.111.111-11	Maria	10
222.222.222-22	José	20
333.333.333-33	Ana	30

Idade Filtro

Figura 03: Visualizar do Exemplo 03 - Consulta

## Recursos utilizadas no Exemplo 03

- **window.location.href**: redireciona para uma nova página HTML
  - Documentação: <https://developer.mozilla.org/pt-BR/docs/Web/API/Document/location>

- **localStorage.setItem**: adiciona ou atualiza dados no localStorage
  - Documentação: <https://developer.mozilla.org/pt-BR/docs/Web/API/Storage/setItem>
- **localStorage.getItem**: recupera dados do localStorage
  - Documentação: <https://developer.mozilla.org/pt-BR/docs/Web/API/Storage/getItem>
- **onload**: evento que executa instruções quando uma página é completamente carregada
  - Documentação: [https://developer.mozilla.org/pt-BR/docs/Web/API/Window/load\\_event](https://developer.mozilla.org/pt-BR/docs/Web/API/Window/load_event)

## Exercícios

1. Crie uma aplicação web composta por duas páginas, utilizando HTML, CSS e JavaScript, conforme os protótipos ilustrados nas Figuras 04 e 05.
  - a. **Página de Cadastro de Contas (Figura 04)**: Esta página deve permitir o cadastro de contas bancárias dos clientes, com as seguintes regras:
    - i. Cada conta deve ter um número único (sem permitir números duplicados).
    - ii. O saldo inicial de cada conta deve ser sempre zero.

Exercício

http://exercicio.html

Exercício Cadastro Contas

Cadastro de conta

Número da conta Nome do cliente

Cadastrar

Figura 04: Protótipo do exercício 01

- b. **Página de Movimentações e Listagem (Figura 05)**: Esta página deve exibir uma lista com as contas bancárias cadastradas e oferecer as seguintes funcionalidades:
  - i. Permitir que o usuário informe o número da conta, o valor e selecione a operação bancária (saque ou depósito).
  - ii. Atualizar automaticamente o saldo da conta conforme as operações realizadas.
  - iii. Exibir os dados atualizados em uma tabela HTML.

Exercício

http://exercicio.html

Exercício Cadastro Contas

Contas

Cadastrar

Número da conta	Nome do cliente	Saldo
1000-1	Maria	500
1000-2	José	1500
1000-3	Ana	-1000

Número da conta Valor Operação

Selecione uma operação

Confirmar

Figura 05: Protótipo do exercício 01

2. Crie uma aplicação web composta por duas páginas, utilizando HTML, CSS e JavaScript, conforme os protótipos ilustrados nas Figuras 06 e 07.

- a. **Página de Cadastro e Autenticação (Figura 06):** Esta página deve permitir que o usuário:
- Faça o cadastro de um novo usuário (sem permitir logins duplicados).
  - Realize a autenticação no sistema.

Exercício

http://exercicio.html

Usuário

Login

Senha

Cadastrar

Login

Senha

Autenticar

Figura 06: Protótipo do exercício 02

- b. **Página de Registro de Vagas (Figura 07):** Esta página deve ser acessível **apenas para usuários autenticados** e deve permitir:
- Registrar a placa do veículo e o número da vaga ocupada no estacionamento.
  - O estacionamento possui 10 vagas e uma vaga já ocupada não pode ser reutilizada.
  - Não é permitido registrar placas duplicadas.
  - O sistema deve armazenar automaticamente o login do usuário autenticado responsável pelo registro da vaga.

Exercício

http://exercicio.html

Exercício Vagas Sair

Login: Maria

Controle de Vagas

Placa do carro

Número da vaga

Selecione uma vaga

Registrar

Número da vaga	Placa do Carro	Login de registro
1	OBA-4565	Maria
2		
3		
4		
5		
6	XYZ-8794	Maria
7		
8		
9	ABC-7895	José
10		

Figura 07: Protótipo do exercício 02