# Tópicos de Javascript:

**var**

**var** é a forma mais antiga de declarar variáveis em Javascript. Ela tem escopo de função ou global, o que significa que a variável declarada com **var** é visível dentro da função em que foi declarada ou globalmente, se não estiver dentro de uma função.

Exemplo de uso de **var**:

var numero = 20;

if (true) {

    var numero = 10;

    console.log("Valor dentro:"+numero); // Saída: 10

}

console.log("Valor Fora:"+numero); // Saída: 10

**let**

**let** foi introduzido no ECMAScript 6 (também conhecido como ES6) e é preferido sobre **var** na maioria dos casos. Ele possui escopo de bloco, o que significa que a variável declarada com **let** é visível apenas dentro do bloco em que foi declarada.

Exemplo de uso de **let**:

let numero = 20;

if (true) {

    let numero = 10;

    console.log("Valor dentro:"+numero); // Saída: 10

}

console.log("Valor Fora:"+numero); // Saída: 20

**const**

**const** também foi introduzido no ECMAScript 6 e é usado para declarar constantes, ou seja, valores que não podem ser alterados após a sua inicialização. Assim como **let**, **const** também tem escopo de bloco.

Exemplo de uso de **const**:

const PI = 3.14159;

console.log(PI); // Saída: 3.14159

PI = 3.14; // Erro: Assignment to constant variable.

É importante notar que enquanto o valor de uma variável declarada com **const** não pode ser alterado, se a variável é um objeto ou um vetor, suas propriedades ou elementos ainda podem ser modificados.

const pessoa = {

    nome: 'João',

    idade: 30

};

console.log(pessoa.nome); // Saída: João

pessoa.nome = 'Maria'; // Isso é permitido

console.log(pessoa.nome); // Saída: Maria

**Quando usar cada um?**

* **var:** Evite seu uso sempre que possível, devido ao seu comportamento de escopo pouco intuitivo e propenso a erros.
* **let:** Use **let** para declarar variáveis que precisam ser reatribuídas ou cujo escopo deve ser limitado a um bloco.
* **const:** Use **const** para declarar constantes que não precisam ser reatribuídas. Isso ajuda a tornar o código mais claro e menos suscetível a erros devido a reatribuições acidentais.

# Vetores e Funções de Vetores

## Introdução

JavaScript é uma linguagem de programação amplamente utilizada para desenvolvimento web, permitindo a criação de páginas dinâmicas e interativas. Uma das estruturas de dados fundamentais em JavaScript é o vetor (ou array), que permite armazenar e manipular conjuntos de dados de forma eficiente.

**Vetores em JavaScript**

Um vetor em JavaScript é uma coleção ordenada de elementos. Esses elementos podem ser de qualquer tipo de dado, como números, strings, objetos, ou até mesmo outros vetores. Vetores em JavaScript são dinâmicos, ou seja, eles podem crescer ou diminuir de tamanho durante a execução do programa.

**Criando Vetores**

Em JavaScript, você pode criar um vetor de várias maneiras. A forma mais comum é usando a sintaxe de colchetes **[]**, onde você lista os elementos separados por vírgulas:

let vetor = [1, 2, 3, 4, 5];

let nomes = ['João', 'Maria', 'José'];

Também é possível criar um vetor vazio e adicionar elementos posteriormente:

let outroVetor = [];

outroVetor.push(10);

outroVetor.push(20);

**Acessando Elementos**

Os elementos de um vetor são acessados usando índices numéricos, começando do zero para o primeiro elemento e aumentando sequencialmente:

let vetor = [10, 20, 30, 40, 50];

console.log(vetor[0]); // Saída: 10

console.log(vetor[2]); // Saída: 30

**Propriedades e Métodos de Vetores**

JavaScript fornece várias propriedades e métodos para trabalhar com vetores. Alguns dos métodos mais comuns incluem:

* **length**: Retorna o número de elementos no vetor.
* **push()**: Adiciona um elemento ao final do vetor.
* **pop()**: Remove e retorna o último elemento do vetor.
* **shift()**: Remove e retorna o primeiro elemento do vetor.
* **unshift()**: Adiciona um elemento ao início do vetor.

Esses são apenas alguns dos métodos disponíveis para manipular vetores em JavaScript. Agora, vamos explorar algumas funções de vetores que são extremamente úteis para transformar e filtrar dados em vetores.

**Funções de Vetores**

As funções de vetores em JavaScript permitem operar em cada elemento de um vetor de forma eficiente e expressiva. Uma das funções mais poderosas é **map()**, que permite aplicar uma função a cada elemento de um vetor e retornar um novo vetor com os resultados.

**A função map()**

A função **map()** recebe uma função de callback como argumento, essa função é aplicada a cada elemento do vetor, e o resultado é armazenado em um novo vetor.

let numeros = [1, 2, 3, 4, 5];

let dobrados = numeros.map(function(numero) {

    return numero \* 2;

});

console.log(dobrados); // Saída: [2, 4, 6, 8, 10]

**A função filter()**

A função **filter()** é usada para filtrar elementos de um vetor com base em uma condição especificada. Ela retorna um novo vetor contendo apenas os elementos para os quais a condição é verdadeira.

let numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

let pares = numeros.filter(function(numero) {

    return numero % 2 === 0;

});

console.log(pares); // Saída: [2, 4, 6, 8, 10]

**A função forEach()**

A função **forEach()** é usada para iterar sobre os elementos de um vetor e executar uma função de callback para cada elemento.

let nomes = ['Alice', 'Bob', 'Charlie'];

nomes.forEach(function(nome) {

    console.log(nome);

});

// Saída:

// Alice

// Bob

// Charlie

* **map():** Retorna um novo vetor com o mesmo número de elementos, mas com cada elemento transformado de acordo com a função de callback.
* **filter():** Retorna um novo vetor contendo apenas os elementos que satisfazem a condição especificada na função de callback.
* **forEach():** Itera sobre os elementos do vetor e executa uma função de callback para cada elemento, mas não retorna um novo vetor.