

Módulo 2: Trabalhando com Objetos e Arrays

1. O que são Arrays?

Arrays (ou vetores) são estruturas de dados que armazenam uma coleção de valores em uma única variável.

Você pode pensar em arrays como uma "lista" de valores, onde cada item tem uma posição (índice), começando do zero.



```
1 let frutas = ["Maçã", "Banana", "Laranja"];  
2 console.log(frutas[0]); // Maçã
```

1.1 Criando e Manipulando Arrays

1.1.1 Criando Arrays:



```
1 let numeros = [10, 20, 30, 40];  
2 let vazio = []; // Array vazio  
3 let misto = ["Texto", 25, true];
```

1.1.2 Acessando e alterando valores:



```
1 let cores = ["vermelho", "azul", "verde"];  
2 cores[1] = "amarelo"; // altera o valor na posição 1  
3 console.log(cores); // ["vermelho", "amarelo", "verde"]
```

1.1.3 Adicionando e removendo elementos:

```
1 let animais = ["gato", "cachorro"];
2 animais.push("papagaio"); // adiciona no final
3 animais.pop();           // remove o último
4 animais.unshift("tigre"); // adiciona no início
5 animais.shift();          // remove o primeiro
```

1.2 Principais Métodos de Arrays

1.2.1 **forEach** — Percorre todos os itens

Executa uma função para cada elemento do array.

```
1 let nomes = ["Ana", "Bruno", "Carlos"];
2
3 nomes.forEach((nome) => {
4   console.log("Olá, " + nome);
5 });
```

1.2.2 **map** — Cria um novo array com base no original

Retorna um **novo array** com os resultados da função aplicada a cada item.

```
1 let numeros = [1, 2, 3];
2
3 let dobrados = numeros.map((n) => n * 2);
4 console.log(dobrados); // [2, 4, 6]
```

1.2.3 filter — Filtra os elementos

Cria um **novo array** contendo apenas os elementos que **passam em uma condição**.

```
1 let idades = [12, 18, 25, 16];
2
3 let maioresDeIdade = idades.filter((idade) => idade >= 18);
4 console.log(maioresDeIdade); // [18, 25]
```

1.2.4 reduce — Reduz o array a um único valor

Usado para acumular valores (como soma ou multiplicação).

```
1 let valores = [10, 20, 30];
2
3 let total = valores.reduce((acumulador, atual) => acumulador + atual, 0);
4 console.log(total); // 60
```

1.2.5 find — Encontra o primeiro valor que satisfaz uma condição

Retorna o **primeiro elemento** que passa na condição, ou undefined se nenhum for encontrado.

```
1 let produtos = ["notebook", "teclado", "mouse"];
2
3 let encontrado = produtos.find((item) => item === "teclado");
4 console.log(encontrado); // "teclado"
```

Exercícios Práticos

1. Crie um array com 5 nomes e exiba cada um deles usando forEach.
2. Use map para criar um novo array que contenha o dobro de cada número do array [2, 4, 6].
3. Crie um array com números de 1 a 10 e use filter para retornar apenas os pares.
4. Use reduce para somar todos os valores do array [5, 10, 15].
5. Crie um array de objetos com nomes e idades. Use find para retornar o primeiro objeto com idade maior que 18.

2. Objetos no JavaScript

2.1 O que são Objetos?

Um objeto é uma estrutura que permite armazenar múltiplos valores em pares de chave e valor. Cada chave (ou "propriedade") tem um nome e está associada a um valor (que pode ser qualquer tipo: número, string, função, array, outro objeto, etc.). Objetos modelam entidades do mundo real — como pessoas, carros, produtos, etc. Exemplo básico:

```
1 let pessoa = {  
2   nome: "Maria",  
3   idade: 30,  
4   profissao: "Desenvolvedora"  
5 };  
6
```

2.2 Definição e Manipulação de Objetos

2.2.1 Criando Objetos


```
1 let carro = {  
2   marca: "Toyota",  
3   modelo: "Corolla",  
4   ano: 2022  
5 };
```

2.3 Acessando propriedades:

2.3.1 Notação de ponto:


```
1 console.log(carro.marca); // "Toyota"
```

2.3.2 Notação de colchetes:




```
1 console.log(carro["modelo"]); // "Corolla"
```

2.3.3 Modificando valores:




```
1 carro.ano = 2023;  
2 carro.cor = "preto"; // adiciona nova propriedade
```

2.3.4 Removendo uma propriedade:



```
1 delete carro.cor;
```

2.3.5 Adicionando Métodos (Funções dentro do Objeto):



```
1 let usuario = {  
2     nome: "João",  
3     saudacao: function() {  
4         console.log("Olá, " + this.nome);  
5     }  
6 };  
7  
8 usuario.saudacao(); // "Olá, João"
```

2.3.6 Diferença entre Objetos e Arrays

CARACTERÍSTICA	OBJETOS	ARRAYS
ORGANIZAÇÃO	Por chave/valor	Por índice numérico
QUANDO USAR?	Quando os dados tem nomes específicos	Quando os dados são listas ordenadas
EXEMPLO	{nome: "Ana", Idade: 25}	["Ana", 25]
ACESSO	obj.nome ou obj["nome"]	Array[0]

2.3.7 Acessar propriedade existente:

```
1 let livro = {  
2   titulo: "1984",  
3   autor: "George Orwell"  
4 };  
5  
6 console.log(livro.titulo); // "1984"
```

2.3.8 Adicionar nova propriedade:

```
1 livro.ano = 1949;
```

2.3.9 Alterar valor de uma propriedade:

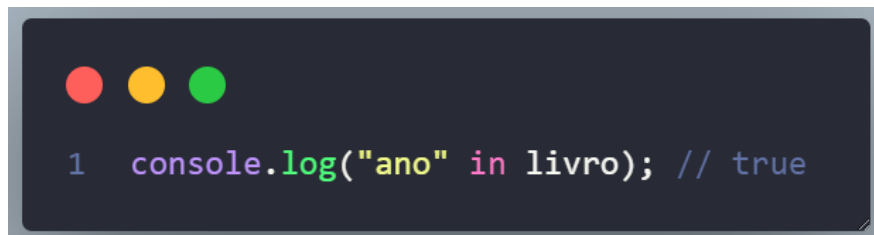
```
1 livro.titulo = "A Revolução dos Bichos";
```

2.3.10 Remover uma propriedade:



```
1 delete livro.autor;
```

2.3.11 Verificar se uma propriedade existe:



```
1 console.log("ano" in livro); // true
```

Exercícios sugeridos

1. Crie um objeto chamado aluno com propriedades: nome, matricula e curso.
2. Adicione uma nova propriedade chamada notaFinal ao objeto.
3. Modifique o valor da notaFinal.
4. Crie um método chamado resumo() que exiba uma frase com nome e curso.
5. Compare um objeto com um array, criando os dois com os mesmos dados e mostrando a diferença de acesso.

3. Programação Orientada a Objetos (POO) em JavaScript

3.1 Introdução à Programação Orientada a Objetos (POO)

A **POO (Programação Orientada a Objetos)** é um **paradigma de programação** baseado em "objetos", que representam entidades do mundo real com **propriedades (atributos)** e **comportamentos (métodos)**.

3.2 Conceitos principais da POO:

- **Classe:** um molde ou estrutura para criar objetos.
- **Objeto (instância):** uma cópia de uma classe com seus próprios valores.
- **Encapsulamento:** esconder a complexidade e expor apenas o necessário.
- **Herança:** uma classe pode herdar comportamentos e atributos de outra.
- **Polimorfismo:** objetos diferentes podem usar o mesmo método de maneiras diferentes.

JavaScript é uma linguagem baseada em objetos e **suporta POO com sintaxe de classes (ES6+)**.

Classes no JavaScript a partir do ES6 (ECMAScript 2015), que trouxeram uma forma mais clara e moderna de trabalhar com **Programação Orientada a Objetos (POO)** na linguagem.

3.3 Classes e Instâncias

3.3.1 Criando uma classe

```
1 class Pessoa {
2   constructor(nome, idade) {
3     this.nome = nome;
4     this.idade = idade;
5   }
6
7   apresentar() {
8     console.log(`Olá, meu nome é ${this.nome} e tenho ${this.idade} anos.`);
9   }
10 }
```

3.1.1 Criando uma instância da classe (objeto)

```
1 let pessoa1 = new Pessoa("Lucas", 28);
2 pessoa1.apresentar(); // "Olá, meu nome é Lucas e tenho 28 anos."
```

3.1.2 Vários objetos com base na mesma classe:

```
1 let pessoa2 = new Pessoa("Juliana", 33);
2 pessoa2.apresentar();
```


3.4 Herança

Herança permite que uma classe **herde propriedades e métodos** de outra, evitando duplicação de código.

3.4.1 Criando uma subclasse

```
1 class Aluno extends Pessoa {
2     constructor(nome, idade, curso) {
3         super(nome, idade); // chama o construtor da classe Pai (Pessoa)
4         this.curso = curso;
5     }
6
7     apresentar() {
8         console.log(`Sou ${this.nome}, tenho ${this.idade} anos e estudo ${this.curso}.`);
9     }
10 }
11
12 let aluno1 = new Aluno("Carlos", 20, "Engenharia");
13 aluno1.apresentar(); // sobrescreve o método da classe Pai
```

3.5 Polimorfismo

Polimorfismo acontece quando duas ou mais classes têm métodos com o mesmo nome, mas com comportamentos diferentes. No exemplo acima, o método `apresentar()` é definido em `Pessoa` e redefinido em `Aluno`. Cada classe se comporta de forma diferente ao chamar o mesmo método.

3.6 Resumo dos Benefícios da POO:

- Reutilização de código (com herança)
- Organização mais clara e modular
- Facilita manutenção e testes
- Permite abstração e encapsulamento

Exercícios sugeridos:

1. Crie uma classe `Carro` com as propriedades `marca` e `ano`, e um método `exibirInfo()`.
2. Crie uma instância da classe `Carro` e exiba as informações.
3. Crie uma classe `Moto` que herda de `Carro` e inclua a propriedade `cilindradas`.
4. Redefina o método `exibirInfo()` em `Moto` para incluir as `cilindradas` (polimorfismo).
5. Crie uma lista de objetos com carros e motos, e execute o método `exibirInfo()` para todos.