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.

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

1.1Criando e Manipulando Arrays

Criando Arrays:

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

Acessando e alterando valores:

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

Adicionando e removendo elementos:

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

1.2 Principais Métodos de Arrays

🔸 `forEach` — Percorre todos os itens

Executa uma função para cada elemento do array.

```javascript
let nomes = ["Ana", "Bruno", "Carlos"];

nomes.forEach((nome) => {
  console.log("Olá, " + nome);
});
```

🔸 `map` — Cria um novo array com base no original

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

```javascript
let numeros = [1, 2, 3];

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

🔸 `filter` — Filtra os elementos

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

javascript

```javascript
let idades = [12, 18, 25, 16];

let maioresDeIdade = idades.filter((idade) => idade >= 18);
console.log(maioresDeIdade); // [18, 25]
```

🔸 `reduce` — Reduz o array a um único valor

Usado para acumular valores (como soma ou multiplicação).

javascript

```javascript
let valores = [10, 20, 30];

let total = valores.reduce((acumulador, atual) => acumulador + atual, 0);
console.log(total); // 60
```

🔸 `find` — Encontra o primeiro valor que satisfaz uma condição

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

javascript

```javascript
let produtos = ["notebook", "teclado", "mouse"];

let encontrado = produtos.find((item) => item === "teclado");
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.

javascript

```javascript
let pessoa = {
  nome: "Maria",
  idade: 30,
  profissao: "Desenvolvedora"
};
```

APOSTILA DE JAVA JAVASCRIPT | PROF: NIXON OLIVEIRA MARINHO

2.2Definição e Manipulação de Objetos

✅ Criando um objeto:

javascript

```javascript
let carro = {
  marca: "Toyota",
  modelo: "Corolla",
  ano: 2022
};
```

✅ Acessando propriedades:

- Notação de ponto:

javascript

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

- Notação de colchetes:

javascript

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

✅ Modificando valores:

javascript

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

✅ Removendo uma propriedade:

javascript

```javascript
delete carro.cor;
```

✅ Adicionando métodos (funções dentro do objeto):

javascript

```javascript
let usuario = {
  nome: "João",
  saudacao: function() {
    console.log("Olá, " + this.nome);
  }
};

usuario.saudacao(); // "Olá, João"
```

3. Diferença entre Objetos e Arrays

Característica	Objetos	Arrays
Organização	Por **chave/valor**	Por **índice numérico**
Quando usar?	Quando os dados têm **nomes específicos**	Quando os dados são **lista ordenada**
Exemplo	`{nome: "Ana", idade: 25}`	`["Ana", 25]`
Acesso	`obj.nome` ou `obj["nome"]`	`array[0]`

2.3Acessando e Modificando Propriedades

✅ Acessar propriedade existente:

javascript

```javascript
let livro = {
  titulo: "1984",
  autor: "George Orwell"
};

console.log(livro.titulo); // "1984"
```

✅ Adicionar nova propriedade:

javascript

```javascript
livro.ano = 1949;
```

✅ Alterar valor de uma propriedade:

javascript

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

✅ Remover uma propriedade:

javascript

```javascript
delete livro.autor;
```

✅ Verificar se uma propriedade existe:

javascript

```javascript
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

✅ Criando uma classe:

javascript

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

  apresentar() {
    console.log(`Olá, meu nome é ${this.nome} e tenho ${this.idade} anos.`);
  }
}
```

✅ Criando uma instância da classe (objeto):

javascript

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

✅ Vários objetos com base na mesma classe:

javascript

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

3.4 Herança

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

✅ Criando uma subclasse:

javascript

```javascript
class Aluno extends Pessoa {
  constructor(nome, idade, curso) {
    super(nome, idade); // chama o construtor da classe Pai (Pessoa)
    this.curso = curso;
  }

  apresentar() {
    console.log(`Sou ${this.nome}, tenho ${this.idade} anos e estudo ${this.curso}.`);
  }
}

let aluno1 = new Aluno("Carlos", 20, "Engenharia");
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

APOSTILA DE JAVA JAVASCRIPT | PROF: NIXON OLIVEIRA MARINHO

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.