

A woman with long dark hair, wearing glasses, is shown from the chest up, looking down at a laptop screen. The background is dark and slightly blurred.

IN

# INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 05 - ARRAYS

# O QUE IREMOS APRENDER

- 01** REVISÃO DA AULA ANTERIOR
- 02** VARIAVEIS SIMPLES E COMPOSTAS
- 03** ARRAYS
- 04** MANIPULANDO ARRAYS
- 05** PERCORRENDO ARRAYS

# Revisão da Aula Anterior

- WHILE
- DO WHILE
- WHILE TRUE



IN

# VARIÁVEIS SIMPLES E COMPOSTAS

As variáveis simples são ideais para armazenar informações únicas e específicas, enquanto as variáveis compostas são essenciais para gerenciar coleções de dados relacionados. Compreender a diferença entre esses dois tipos de variáveis é fundamental para a manipulação eficaz de dados.

## Comparação

### Variáveis Simples:

- Armazenam um único valor.
- Simplicidade: Armazenam e manipulam um único valor.
- Utilizadas para armazenar informações diretas e específicas, como idade, nome, status de atividade, etc.

### Variáveis Compostas:

- Armazenam múltiplos valores em uma estrutura organizada.
- Podem armazenar múltiplos valores e diferentes tipos de dados.
- Permitem organizar e estruturar dados de maneira mais complexa.

# VARIÁVEIS SIMPLES

As variáveis simples, também conhecidas como variáveis escalares, são usadas para armazenar um único valor por vez. Em JavaScript, os tipos primitivos de variáveis simples incluem números, strings, booleanos, null, e undefined.

---

## Exemplos:

```
1 let idade = 25;      // Número
2 let nome = "João";   // String
3 let ativo = true;    // Booleano
4 let saldo = null;    // Null
5 let indefinido;     // Undefined
```

# VARIÁVEIS COMPOSTAS

As variáveis compostas, também conhecidas como coleções ou estruturas de dados, são usadas para armazenar múltiplos valores em uma única variável. Em JavaScript, os principais tipos de variáveis compostas incluem arrays e objetos.

---

## Exemplos

```
1 let frutas = ["Maçã", "Banana", "Laranja"]; //Lista
2 let pessoa = { nome: "Carlos", idade: 40 }; //Objeto
```

# ARRAYS

## O que são Arrays?

Arrays são objetos usados para armazenar múltiplos valores em uma única variável. Eles podem conter elementos de qualquer tipo, incluindo números, strings e outros arrays. Os elementos são acessados por meio de índices, começando do zero.

## Sintaxe:

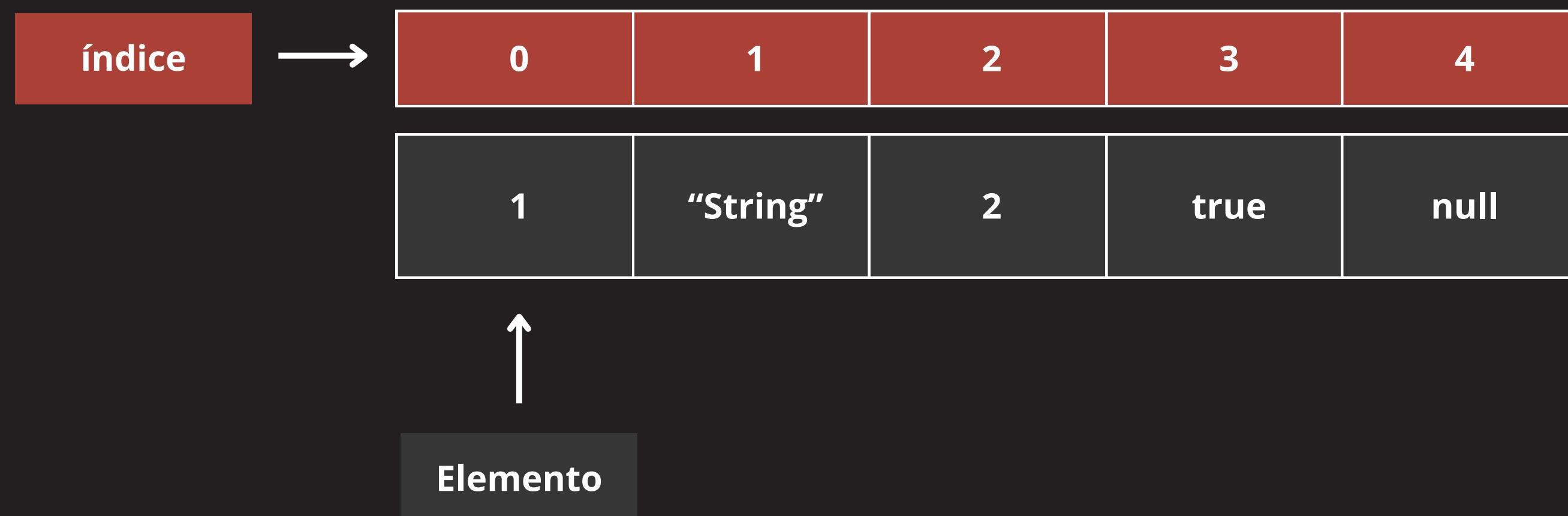
```
1 let arrayVazio = []; // Array vazio
2 let arrayDeNumeros = [1, 2, 3, 4, 5]; // Array com números
3 let arrayDeStrings = ["Maçã", "Banana", "Laranja"]; // Array com strings
```

## Vantagens:

A utilização de arrays oferece diversas vantagens em comparação com variáveis simples, especialmente quando se trata de armazenar e manipular múltiplos valores de maneira eficiente e organizada. Arrays proporcionam métodos embutidos poderosos, facilitam a iteração e permitem um agrupamento lógico de dados relacionados, tornando o código mais limpo e fácil de manter.

# ÍNDICES

Os índices são números inteiros que representam a posição de elementos dentro de um array. Eles começam em zero, o que significa que o primeiro elemento está na posição 0, o segundo na posição 1, e assim por diante.



# TAMANHO

É possível acessar o tamanho de um array utilizando a propriedade **length**. Esta propriedade não é um método, mas uma propriedade embutida que retorna o número de elementos presentes no array. Para utilizá-la, basta referenciar a propriedade diretamente no array desejado.

**Exemplo:**

```
1 const array = [1, 2, 3, 4, 5];
2 console.log(array.length);
3 // Saída: 5
```

Neste exemplo, `array.length` retorna o valor 5, que é o número de elementos no array.

# ACESSANDO ELEMENTOS DE UM ARRAY

## Acessando Elementos por Índice:

Para acessar um elemento de um array, utilizamos a notação de colchetes [ ], colocando o índice do elemento que queremos acessar dentro dos colchetes.

## Exemplos:

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

# DINÂMICA DE AULA

```
1 let animais = ["Cachorro", "Gato", "Papagaio", "Tartaruga"];
```

Dado o array acima, será que os alunos conseguem dizer qual o nome do animal que está presente na posição 2?

---

**Objetivo:** dinâmica de turma para e testar o conhecimento sobre índices de arrays

# ATIVIDADE PRÁTICA

## **Atividade 01**

Primeiro, crie um array chamado cores contendo os seguintes elementos: Vermelho, Verde, Azul, Amarelo. Em seguida, acesse e exiba no console o elemento na posição 3.

## **Objetivo:**

Praticar o uso de índices para acessar elementos específicos em um array.

# MANIPULANDO ARRAYS

## Métodos:

são funções embutidas que podem ser aplicadas a arrays para realizar várias operações. Esses métodos facilitam a manipulação, iteração, transformação, pesquisa e ordenação dos dados armazenados.

## Importância:

- Simplificam operações complexas.
- Tornam o código mais legível e conciso.
- Melhoram a eficiência e a performance ao trabalhar com grandes conjuntos de dados.

# ARRAY METHODS

## Push():

Adiciona um ou mais elementos ao final do array e retorna o novo comprimento do array.

### Sintaxe:

```
1 let frutas = ["Maçã", "Banana"];
2 frutas.push("Laranja");
3 // ["Maçã", "Banana", "Laranja"]
```

## Pop():

Remove o último elemento do array e o retorna.

### Sintaxe:

```
1 let frutas = ["Maçã", "Banana", "Laranja"];
2 let ultimaFruta = frutas.pop();
3 // ["Maçã", "Banana"]
```

# ARRAY METHODS

## Shift():

Remove o primeiro elemento do array e o retorna..

### Sintaxe:

```
1 let frutas = ["Maçã", "Banana", "Laranja"];
2 let primeiraFruta = frutas.shift();
3 // ["Banana", "Laranja"]
```

## Unshift():

Adiciona um ou mais elementos ao início do array e retorna o novo comprimento do array.

### Sintaxe:

```
1 let frutas = ["Banana", "Laranja"];
2 frutas.unshift("Maçã");
3 // ["Maçã", "Banana", "Laranja"]
```

# ARRAY METHODS

## Splice():

Altera o conteúdo de um array, adicionando, removendo ou substituindo elementos.

### Sintaxe:

```
1 let frutas = ["Maçã", "Banana", "Laranja"];
2 frutas.splice(1, 0, "Morango", "Uva");
3 console.log(frutas);
4 // Saída: ["Maçã", "Morango", "Uva", "Banana", "Laranja"]
```

## Slice():

Retorna uma cópia superficial de uma parte de um array, selecionada desde o início até o fim (fim não incluído). O array original não é modificado.

### Sintaxe:

```
1 let frutas = ["Maçã", "Banana", "Laranja", "Morango"];
2 let citricos = frutas.slice(1, 3);
3 console.log(citricos);
4 // Saída: ["Banana", "Laranja"]
5 console.log(frutas);
6 // Saída: ["Maçã", "Banana", "Laranja", "Morango"]
```

# ARRAY METHODS

## Concat():

Usado para mesclar dois ou mais arrays. Este método não altera os arrays existentes, mas retorna um novo array.

## Sintaxe:

```
1 let frutas1 = ["Maçã", "Banana"];
2 let frutas2 = ["Laranja", "Morango"];
3 let todasFrutas = frutas1.concat(frutas2);
4 console.log(todasFrutas);
5 // Saída: ["Maçã", "Banana", "Laranja", "Morango"]
```

## Reverse():

Inverte a ordem dos elementos de um array no local. O primeiro elemento se torna o último, e o último elemento se torna o primeiro.

## Sintaxe:

```
1 let numeros = [1, 2, 3, 4, 5];
2 numeros.reverse();
3 console.log(numeros);
4 //Saída: [5, 4, 3, 2, 1]
```

# ARRAY METHODS

## Join():

Une todos os elementos de um array em uma string e retorna esta string. É possível especificar um separador.

## Sort():

Ordena os elementos de um array no local e retorna o mesmo. A ordenação não é necessariamente estável. A ordem padrão é crescente, convertendo os elementos em strings.

## Sintaxe:

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

## Sintaxe:

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

# Atividade Pratica

*Objetivo: Praticar operações básicas de criação, acesso e modificação de elementos em arrays.*

*Estrutura:*

Crie um array chamado animais contendo os seguintes elementos, Cachorro, Gato, Papagaio. *Em seguida:*

- 1 - *Adicione o elemento Tartaruga ao final do array*
- 2 - *Remova o primeiro elemento e adicione o elemento Coelho no início.*
- 3 - *Depois, substitua o elemento na posição 2 por Hamster.*
- 4 - *Exiba o comprimento atual do array.*
- 5 - *Acesse e exiba no console o elemento na posição 1.*
- 6 - *Finalmente, exiba o array final no console.*

# Atividade Prática

**Objetivo:** Praticar operações básicas de criação, acesso e modificação de elementos em arrays.

## Estrutura:

Crie um array chamado cores contendo as seguintes cores, Vermelho, Verde, Azul.

- 1 - Adicione as cores Amarelo e Roxo ao final do array.
- 2 - Em seguida, remova a última cor.
- 3 - Use o método splice para remover "Verde" e adicionar "Laranja" e "Marrom" em seu lugar.
- 4 - Crie um novo array chamado novasCores contendo as primeiras duas cores do array cores.
- 5 - Use o método join para criar uma string com todas as cores do array cores, separadas por uma vírgula.
- 6 - Inverta a ordem dos elementos no array cores.
- 7 - Exiba os arrays cores e novasCores, e a string resultante no console.

## Dica Bônus

### GITHUB:

Suba essa pequena atividade no GITHUB.



# PERCORRENDO ARRAYS

Em JavaScript, percorrer arrays é uma tarefa comum e essencial para manipular e processar dados armazenados em listas. Existem várias maneiras de iterar sobre os elementos de um array, cada uma com suas próprias vantagens e casos de uso específicos.

JavaScript oferece diversas formas de percorrer arrays, cada uma adequada a diferentes situações. Desde o loop for tradicional, que oferece controle total sobre o índice, até métodos mais modernos e declarativos como for...of e forEach, há uma solução para cada necessidade.

for()

for(of)

forEach()

IN

# PERCORRENDO ARRAYS COM FOR

## Loop **for** Tradicional:

- Permite controle total sobre o índice do array.
- Ideal para quando você precisa acessar o índice do elemento.

## exemplo:

```
1 const numeros = [1, 2, 3, 4, 5];
2 for (let i = 0; i < numeros.length; i++) {
3     console.log(numeros[i]);
4 }
5 // Saída: 1, 2, 3, 4, 5
```

# PERCORRENDO ARRAYS COM FOR...OF

## Loop **for...of**:

- Simples e legível, focado nos valores dos elementos
- Não fornece o índice dos elementos.

## exemplo:

```
1 const frutas = ["maçã", "banana", "laranja"];
2 for (const fruta of frutas) {
3     console.log(fruta);
4 }
5 // Saída: "maçã", "banana", "laranja"
```

# SE LIGA NO CONTEÚDO DA PRÓXIMA AULA!

AULA 06 DE JAVASCRIPT.  
FUNÇÕES I.



INFINITY SCHOOL  
VISUAL ART CREATIVE CENTER

# Aula 06 - Funções

FUNÇÕES

PARÂMETROS DE FUNÇÕES

RETORNANDO VALORES

A woman with long dark hair, wearing glasses, is shown from the chest up, looking down at a laptop screen. The background is dark and slightly blurred.

IN

# INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 05 - ARRAYS