KORU


DESENVOLVE
programa de formação
e inclusão de talentos
em tecnologia
Grupo
Boticário
KORU

ARRAYS E OBJETOS

O QUE VAMOS APRENDER HOJE?

- **Estruturas de Dados** - por que precisamos agrupar dados?
- **Arrays**: Listas ordenadas
- Criação, acesso, métodos essenciais
- Iteração e transformação
- **Objetos**: Coleções chave-valor
- Criação, acesso, métodos
- Iteração
- **Destructuring**: "desempacotando" dados

POR QUE AGRUPAR DADOS?

POR QUE AGRUPAR DADOS?

dados → variáveis → objetos

Imagine termos que guardar informações de 10 usuários:

```
let userName1 = "Carlos";
let userAge1 = 30;
let userName2 = "João";
let userAge2 = 25;
// ... muito repetitivo e difícil de gerenciar!
```

Existe uma forma melhor de organizar dados relacionados:

Seja através de **listas ordenadas**, seja através de **coleções de propriedades**.

ARRAYS

ARRAYS

Nada mais são do que **listas**

Coleção **ordenada** de valores

A **ordem importa**

Acesso aos itens por um índice que é numérico (e começa com 0)

fonte: https://sites.duke.edu/writing270_02_ss12021_ecyoung/2021/06/28/data-structures-in-our-everyday-lives-arrays/

CRIANDO E ACESSANDO ARRAYS

Forma mais comum de **criar arrays**: sintaxe literal []

```
// Array de strings
let shoppingList = ["Leite", "Pão", "Ovos"];

// Array de números
let scores = [95, 80, 100, 75];

// Array com tipos mistos (possível!)
let mixedData = ["Text", 10, true, null];
```

CRIANDO E ACESSANDO ARRAYS

Acessando elementos (índice):

```
// Array de strings
let shoppingList = ["Leite", "Pão", "Ovos"];

// Array de números
let scores = [95, 80, 100, 75];

// Array com tipos mistos (possível!)
let mixedData = ["Text", 10, true, null];
```

```
console.log(shoppingList[0]); // Output: Leite
console.log(scores[2]);       // Output: 100
console.log(shoppingList[5]); // Output: undefined (índice não existe)
```

CRIANDO E ACESSANDO ARRAYS

Acessando elementos com array.at():

```
// Array de strings
let shoppingList = ["Leite", "Pão", "Ovos"];

// Array de números
let scores = [95, 80, 100, 75];

// Array com tipos mistos (possível!)
let mixedData = ["Text", 10, true, null];
```

```
// Primeiro elemento
console.log(shoppingList.at(0)); // Output: Leite

// Último elemento
console.log(scores.at(-1)); // Output: 75

// Qualquer elemento (de trás para frente)
console.log(mixedData.at(-3)); // Output: 10
```

CRIANDO E ACESSANDO ARRAYS

Contando elementos

```
// Array de strings
let shoppingList = ["Leite", "Pão", "Ovos"];

// Array de números
let scores = [95, 80, 100, 75];

// Array com tipos mistos (possível!)
let mixedData = ["Text", 10, true, null];
```

```
console.log(shoppingList.length); // Output: 3
console.log(scores.length); // Output: 4
console.log(mixedData.length); // Output: 4
```

PRÁTICA 01. ACESSANDO ELEMENTOS

PRÁTICA

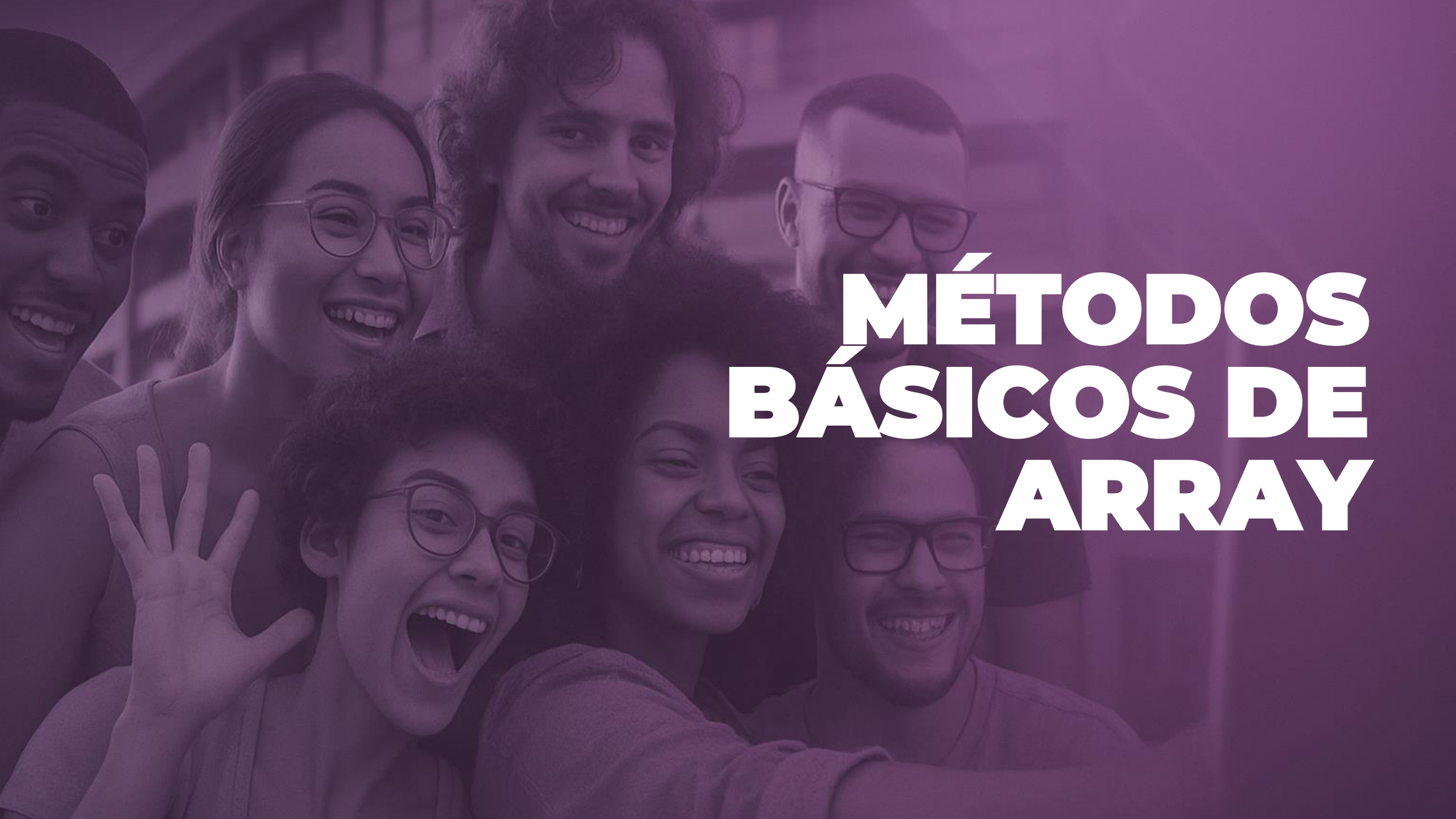
Vamos criar um array - mas não sabemos quantos itens ele possui.

Digite isso no seu VSCode:

- const myArray = Array.from({ length: Math.floor(Math.random() * 500) }, (_, i) => i);

- Quantos itens o array criado possui?
- Acesse o primeiro elemento desse array.
- Acesse o último elemento desse array.
- Acesse o penúltimo elemento desse array.
- Extra: qual é o maior e qual é o menor valor do array?

```
// Criando um array com números variados de 0 a 500 elementos
const myArray = Array.from({ length: Math.floor(Math.random() * 500) }, (_, i) => i);
```



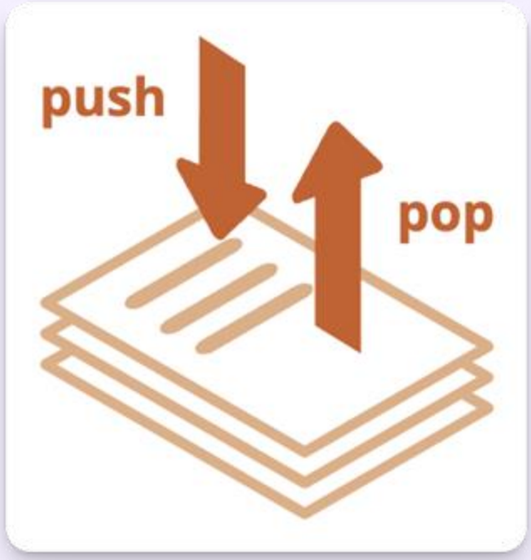
MÉTODOS BÁSICOS DE ARRAY

MÉTODOS BÁSICOS DE ARRAY

⚠️ Cuidado - esses métodos **modificam** o array original! (existem outros métodos que retornam uma cópia do array original).

pop/**push** - Remove a adiciona item ao final.

shift/**unshift -** Remove e adiciona item ao início



fonte: https://javascript.info/array#methods-pop-push-shift-unshift

MÉTODOS BÁSICOS NÃO DESTRUTIVOS

MÉTODOS BÁSICOS NÃO DESTRUTIVOS

Vamos ver outros métodos (slice, includes, indexOf, join, concat)

Nenhum deles modificam o array original (geralmente retornam um novo array)

slice (start, end): retorna uma cópia de parte do array:

```
let animals = ['Formiga', 'Boi', 'Cobra', 'Pato', 'Elefante'];
let middleAnimals = animals.slice(1, 4); // Do índice 1 até ANTES do 4
console.log(middleAnimals); // Output: ['Boi', 'Cobra', 'Pato']
console.log(animals);      // Output: ['Formiga', 'Boi', 'Cobra', 'Pato', 'Elefante']
(Original intacto!)
```

MÉTODOS BÁSICOS NÃO DESTRUTIVOS

Includes(item): verifica se um item existe no array

indexOf(item): Retorna o primeiro índice do item, ou -1 se não encontrar

```
// indexOf
let fruits = ["Maçã", "Banana", "Melancia"];

console.log(fruits.indexOf("Banana")); // 1
console.log(fruits.indexOf("Pera"));   // -1 (não encontrado)
```

```
let fruits = ["Maçã", "Banana", "Melancia"];

console.log(fruits.includes("Banana")); // true
console.log(fruits.includes("Pera"));   // false
```

MÉTODOS BÁSICOS NÃO DESTRUTIVOS

join(separator): Junta os elementos em uma *string*

concat(otherArray): Cria um novo array juntando outros

```
let fruits = ["Maçã", "Banana", "Melancia"];
let fruits2 = ["Pera", "Uva", "Pêssego"];

let combinedFruits = fruits.concat(fruits2);
console.log(combinedFruits); // Output: ["Maçã", "Banana", "Melancia", "Pera",
"Uva", "Pêssego"]

console.log(fruits.join(' / ')); // Output: "Maçã / Banana / Melancia"
```

LOOPS BÁSICOS EM ARRAYS

LOOPS BÁSICOS DE ARRAY

O mais tradicional é o famoso **for loop**.

```
let arr = ["Maçã", "Banana", "Uva"];


for (let i = 0; i < arr.length; i++) {
  console.log( arr[i] );
}
```

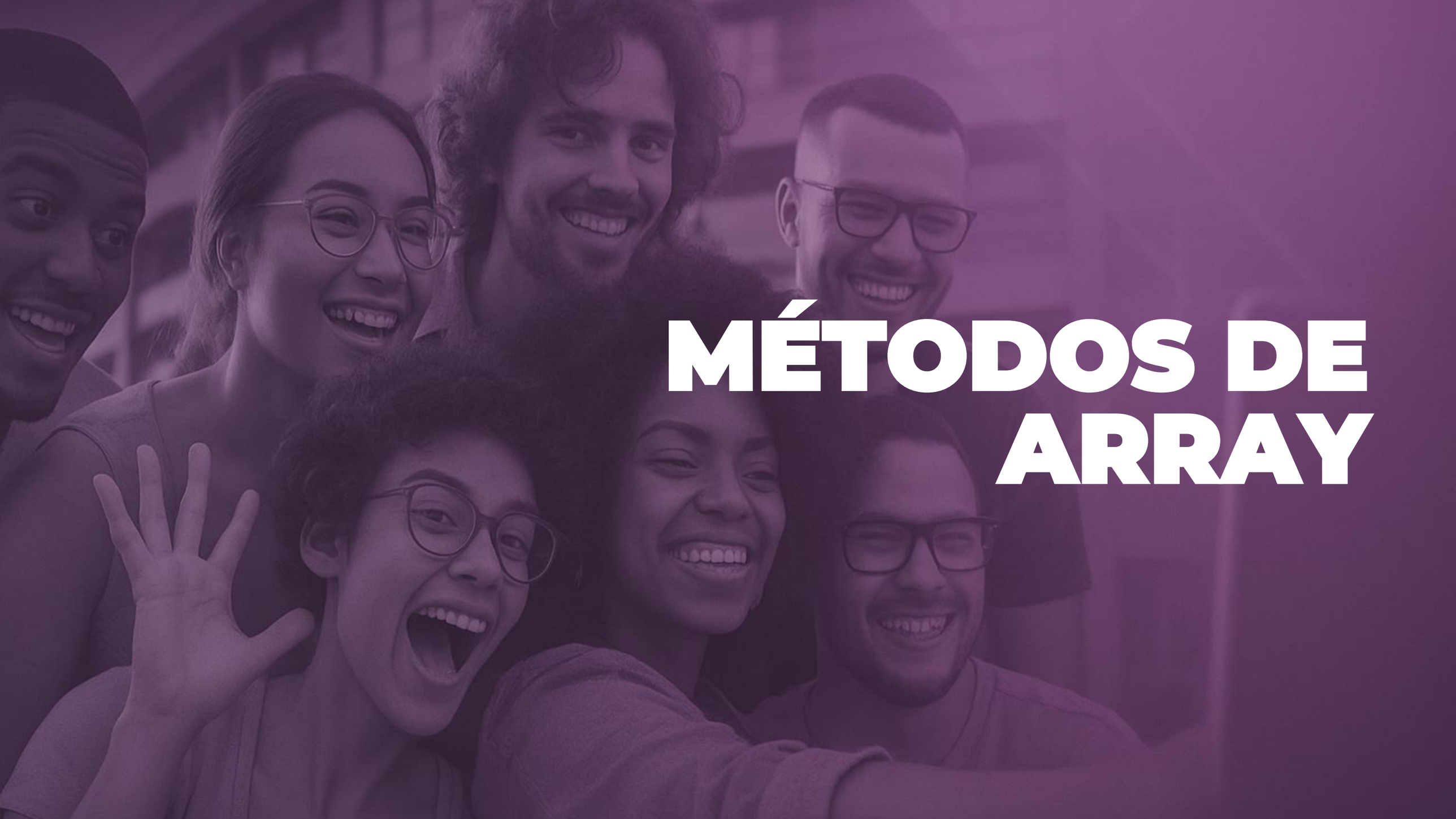
LOOPS BÁSICOS DE ARRAY

for ... of também é uma possibilidade em arrays!

O **for ... of** não fornece acesso ao índice do elemento atual, apenas ao seu valor.

```
let fruits = ["Maçã", "Banana", "Melancia"];

// iterates over array elements
for (let fruit of fruits) {
  console.log( fruit );
}
```



MÉTODOS DE ARRAY

MÉTODOS DE ARRAY

Vamos ver agora o **map**, **filter** e **find**, que também são chamados de higher order functions.

Isso porque cada um desses métodos recebe uma função que será chamada para cada item do array.

Map: serve para mudar o formato do array, mas com o mesmo número de itens.

Filter: retorna um novo array no mesmo formato mas com número de itens de acordo com o filtro.

Find: retorna o primeiro item que passar no teste (função que retorna true)

MÉTODOS DE ARRAY

fonte: https://www.reddit.com/r/ProgrammerHumor/comments/55ompo/map_filter_reduce_explained_with_emojis/

MAP

Map() é provavelmente o método de array mais usado

Você passa uma função para o map(). Essa função irá ser chamada para cada item. O item será substituído pelo retorno da função

fonte: https://dev.to/devmount/javascript-array-methods-explained-with-emojis-2amn

FILTER

Muito utilizado pois serve para filtrar o array.

Você passa uma **função**. Essa função irá ser chamada para cada item. O item irá permanecer no novo array se a função retornar true.

```
const emojis = ["🍎", "🍌", "🍕", "🍇", "🍔"];

function isFruit(emoji) {
  const fruitEmojis = ["🍎", "🍌", "🍇"];
  return fruitEmojis.includes(emoji);
}

const fruits = emojis.filter(isFruit);

console.log(fruits); // Output: ["🍎", "🍌", "🍇"]
```

FIND

Ao contrário dos outros métodos, o **find retorna um item** do array e não um novo array.

O item retornado será o primeiro que passar no teste da função (que retornar true).

```
const emojis = ["🚗", "🐶", "🍕", "🐱", "✈️"];

function isDog(emoji) {
  return emoji === "🐶";
}

const dog = emojis.find(isDog);

console.log(dog); // Output: "🐶"
```

RESUMO DE ARRAYS

Arrays são listas ordenadas acessadas por índice ([0], [1], ...)

Propriedade **length**: mostra o tamanho do array.

Métodos que modificam: **push**, **pop**, **shift**, **unshift**.

Métodos que não modificam: **slice**, **concat**, **includes**, **indexOf**.

Iteração: **for** loop, **for...of** (valores).

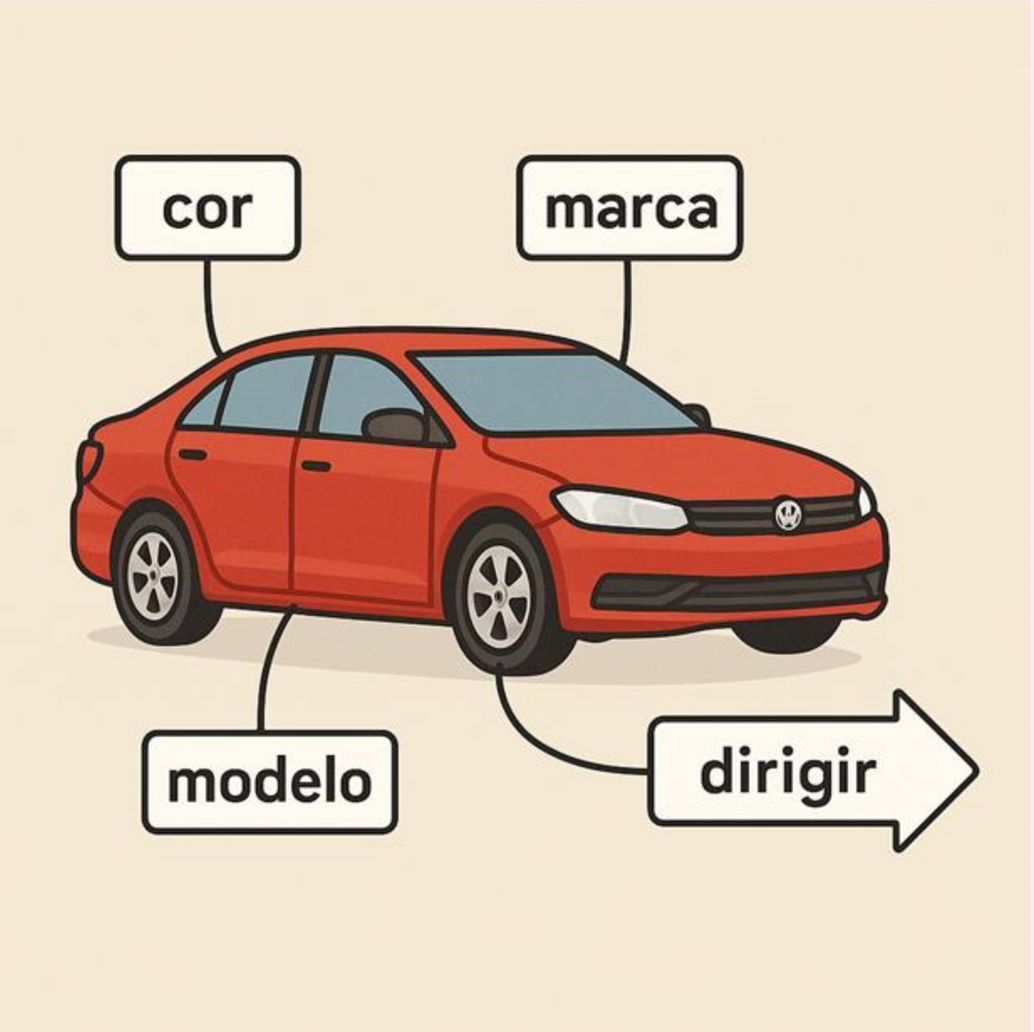
Transformação/seleção: **map**, **filter**, **find**.

OBJETOS

OBJETOS

Coleção não ordenada de pares chave: valor.

- Chaves *são geralmente strings*
- Valores podem ser de qualquer tipo (string, number, boolean, array, outro objeto, function).
- Acesso aos valores pela chave.
- Objeto: analogia a um carro:
  - Propriedades (*cor*, *marca*, *modelo*)
  - Métodos (função *dirigir*)



CRIANDO OBJETOS

Criando Objetos

Sintaxe literal: `{}`

Acessando Propriedades

Dot Notation

Bracket Notation (em alguns casos necessário)

```
let student = {
  firstName: "João",
  lastName: "Silva",
  course: "Desenvolvimento Web",
  grades: [10, 9, 8],
  isActive: true
};
```

```
let propName = "lastName";
console.log(student[propName]);     // Output: Silva
student["favorite-food"] = "Pizza"; // Adicionando chave com hífen
console.log(student["favorite-food"]); // Output: Pizza
```

```
console.log(student.firstName); // Output: João
console.log(student.course);    // Output: Desenvolvimento Web
```

MODIFICANDO OBJETOS

Modificando Objetos

- Modificando
- Adicionando
- Removendo

```
student.isActive = false;
student["course"] = "Advanced Web Dev";
console.log(student.isActive); // Output: false
```

```
student.email = "joao.silva@email.com";
console.log(student.email); // Output: joao.silva@email.com
```

```
delete student["favorite-food"];
console.log(student["favorite-food"]); // Output: undefined
```

MÉTODOS EM OBJETOS

Métodos em Objetos

Métodos são funções armazenadas como propriedades de um objetos

this: dentro de um método de objeto, this geralmente se refere ao objeto à esquerda do ponto `.` na chamada do método (existem exceções mas não se preocupem com elas).

```
let calculator = {
  num1: 0,
  num2: 0,
  add: function() {
    // 'this' se refere ao próprio objeto 'calculator'
    return this.num1 + this.num2;
  },
  setNumbers: function(n1, n2) {
    this.num1 = n1;
    this.num2 = n2;
  }
};

calculator.setNumbers(5, 10);
console.log(calculator.add()); // Output: 15
```

ITERANDO SOBRE OBJETOS

Percorrendo propriedades

for...in itera sobre as chaves (strings).

É possível também usar os métodos

- Object.**keys**(obj): Array das chaves
- Object.**values**(obj): Array dos valores
- Object.**entries**(obj): Array de [chave, valor]

```
const person = {
  name: "Maria",
  age: 30,
  city: "São Paulo"
};

// Com for...in (potencialmente problemático)
for (let prop in person) {
  console.log(`${prop}: ${person[prop]}`);
  // Percorre propriedades herdadas também!
}

// Melhor: Object.entries()
Object.entries(person).forEach(([key, value]) => {
  console.log(`${key}: ${value}`);
});

// Apenas chaves
const keys = Object.keys(person);
console.log(keys); // ["name", "age", "city"]

// Apenas valores
const values = Object.values(person);
console.log(values); // ["Maria", 30, "São Paulo"]
```

RESUMO DE OBJETOS

Coleções não ordenadas de **pares chave-valor**.

Acesso por chave (Dot **.** ou Bracket **[]**).

Fácil de adicionar, modificar e remover propriedades (delete).

Podem conter **métodos** (funções) e usar **this**.

Iteração segura com Object.keys(), Object.values(), Object.entries().


DESTRUCTURING

DESTRUCTURING

Destructuring é uma sintaxe especial para extrair valores de **arrays** ou **propriedades de objetos** diretamente para variáveis.

Torna o código mais curto, legível e menos repetitivo.

```
// Antes:
let person = { firstName: "Carol", age: 35 };
let personName = person.firstName;
let personAge = person.age;

// Com Destructuring:
let { firstName, age } = person; // Bem melhor!
console.log(firstName, age); // Output: Carol 35
```

ARRAYS DESTRUCTURING

Como o nome diz, serve para mapear itens de arrays diretamente a variáveis.

```
// Array Destructuring
const coordinates = [10, 20, 30];
const [x, y, z] = coordinates;
console.log(x, y, z); // Output: 10 20 30

// Pulando o segundo elemento
const [first, , third] = coordinates;
console.log(first, third); // Output: 10 30

// Pegando o primeiro e o restante
const [winner, ...runnersUp] = ["Ouro", "Prata", "Bronze", "Cobre"];
console.log(winner);     // Output: Ouro
console.log(runnersUp); // Output: ['Prata', 'Bronze', 'Cobre']

// Padrão
const [a = 1, b = 2] = [10];
console.log(a, b); // Output: 10 2 (b usou o padrão)
```

OBJECT DESTRUCTURING

Como o nome diz, serve para mapear propriedades de objetos diretamente a variáveis.

```
// Object Destructuring
const car = { brand: "Tesla", model: "Model 3", year: 2023 };
const { brand, model } = car;
console.log(brand, model); // Output: Tesla Model 3

// Renomeando propriedades
const { brand: carBrand, model: carModel } = car;
console.log(carBrand, carModel); // Output: Tesla Model 3

// Padrão
const { color = "Branco", year } = car;
console.log(color, year); // Output: Branco 2023 (color usou o padrão)

// Aninhamento
const userProfile = { id: 1, info: { userName: "Carla Silva", city: "São Paulo" } };
const { id, info: { userName, city } } = userProfile;
console.log(id, userName, city); // Output: 1 Carla Silva São Paulo

// Rest operator
const { id: userId, ...otherInfo } = userProfile;
console.log(userId);    // Output: 1
console.log(otherInfo); // Output: { info: { userName: 'Carla Silva', city: 'São Paulo' } }
```

RESUMO DE DESTRUCTURING

Forma concisa de extrair dados de **Arrays** e **Objetos**.

Arrays: Usa `[]`, ordem importa, pode pular (`,`), usar rest (`...`), defaults (`=`).

Objetos: Usa `{}`, nomes importam (ou renomear com `:`), pode usar rest (...), defaults (=), aninhamento.

Objetivo: Código mais limpo e direto

Vamos
avaliar o
encontro?

KORU

DESENVOLVE
programa de formação
e inclusão de talentos
em tecnologia

+ Grupo Boticário + KORU