MATERIAL EXTRA

Métodos nativos

growdev

MÉTODOS NATIVOS

Métodos nativos em JavaScript são funções pré-definidas que fazem parte das construções básicas da linguagem. Eles são integrados no ambiente JavaScript e estão disponíveis para uso sem a necessidade de importação ou definição adicional. Cada tipo de dado primitivo ou objeto em JavaScript, como strings, números e arrays, possui seu próprio conjunto de métodos nativos.

Para que servem?

- **Manipulação de Dados**: Métodos nativos permitem que você realize operações comuns em dados, como manipulação de strings, cálculos numéricos, operações com arrays, e manipulação de objetos. Por exemplo, você pode usar métodos de array como map() ou filter() para processar listas de dados de forma eficiente.

- **Facilidade de Uso**: Eles simplificam tarefas complexas, tornando o código mais limpo e mais fácil de entender. Em vez de escrever suas próprias funções para operações comuns, você pode utilizar métodos nativos que já fazem isso.

- **Padronização**: Os métodos nativos oferecem uma interface consistente para trabalhar com diferentes tipos de dados. Isso ajuda a garantir que os desenvolvedores usem abordagens semelhantes em diferentes partes do código, facilitando a colaboração e a manutenção.

Por que são tão importantes?

- **Eficiência**: Os métodos nativos são geralmente otimizados para desempenho, o que significa que eles tendem a ser mais rápidos do que as funções personalizadas. Usar esses métodos pode resultar em melhorias significativas na eficiência do seu código.

- **Abstração**: Eles abstraem a complexidade da implementação, permitindo que os desenvolvedores se concentrem na lógica de negócios em vez de se preocupar com os detalhes de como certas operações são realizadas.

- **Consistência e Compatibilidade**: Como esses métodos são parte do padrão ECMAScript, eles são suportados em todos os navegadores modernos. Isso garante que o código que usa esses métodos funcione de maneira consistente em diferentes ambientes.

- **Facilidade de Aprendizado**: Conhecer os métodos nativos é fundamental para qualquer desenvolvedor JavaScript, pois eles formam a base das operações que você realizará frequentemente. Isso também ajuda a acelerar o aprendizado e a produtividade.

Exemplos de Uso

- **Strings**: Para manipulação de texto, como concatenar ou dividir strings.
- **Números**: Para operações matemáticas, como arredondar ou formatar números.
- **Arrays**: Para manipulação de coleções de dados, como adicionar, remover, ou transformar elementos.

Em resumo, os métodos nativos são uma parte fundamental do JavaScript, permitindo que os desenvolvedores escrevam código eficiente, claro e fácil de manter. Conhecer e entender como usar esses métodos é essencial para qualquer pessoa que trabalhe com a linguagem, então vamos conhecer como funcionam alguns dos métodos mais utilizados para cada um dos exemplos de uso citados acima.

Métodos de String

1. trim()

É usado para remover espaços em branco do início e do fim de uma string. Retorna uma nova string sem os espaços em branco no início e no final.

```
const str = '   Olá, mundo!   '
const trimmedStr = str.trim()
console.log(trimmedStr) // "Olá, mundo!"
```

Existem outros dois métodos que se assemelham bastante com o método trim, são eles:

- **trimStart**: é usado para remover espaços em branco apenas no início de uma string.
- **trimEnd**: é usado para remover espaços em branco apenas do final de uma string.

2. replaceAll

O método replaceAll foi introduzido no ECMAScript 2021 (ES12) e, como o nome sugere, substitui todas as ocorrências de uma substring ou correspondência de expressão regular na string por uma nova substring.

```
const str = 'Olá, mundo! Olá, todos!'
const newStr = str.replaceAll('Olá', 'Oi')
console.log(newStr) // "Oi, mundo! Oi, todos!"
```

Um outro método que se assemelha muito ao replaceAll é o método replace. A diferença é que, enquanto o replaceAll substitui todas as ocorrências de uma substring, **o replace substitui apenas a primeira ocorrência.**

```
const str = 'Olá, mundo! Olá, todos!'
const newStr = str.replace('Olá', 'Oi')
console.log(newStr) // "Oi, mundo! Olá, todos!"
```

3. toLowerCase e toUpperCase

O método toLowerCase converte todos os caracteres de uma string para letras minúsculas. Ele não altera a string original, mas retorna uma nova string com a conversão aplicada.

```
const str = 'Olá, Mundo!'
const lowerStr = str.toLowerCase()
console.log(lowerStr) // "olá, mundo!"
```

Já o método toUpperCase converte todos os caracteres de uma string para letras maiúsculas. Assim como o toLowerCase, ele não modifica a string original e retorna uma nova string com a conversão aplicada.

```
const str = 'Olá, Mundo!'
 const upperStr = str.toUpperCase()
 console.log(upperStr) // "OLÁ, MUNDO!"
```

4. indexOf e lastIndexOf

O método indexOf em JavaScript é usado para encontrar a posição da primeira ocorrência de uma substring em uma string. Ele retorna o índice (posição) da substring encontrada ou -1 se a substring não estiver presente na string.

```
const str = 'Olá, Mundo!'
const index = str.indexOf('Mundo')
console.log(index) // 5
```

Enquanto indexOf encontra a primeira ocorrência de uma substring, o método lastIndexOf encontra a última ocorrência. A sintaxe e o uso são semelhantes:

```
const str = 'Olá, Mundo! Olá, Planeta!'
const lastIndex = str.lastIndexOf('Olá')
console.log(lastIndex) // 13
```

5. substring

O método substring em JavaScript é usado para extrair uma parte de uma string, com base em dois índices que representam o início e o fim do trecho que você deseja capturar. Ele retorna a parte da string localizada entre os índices informados, sem alterar a string original.

```
const str = "JavaScript é incrível!"
const subStr = str.substring(0, 10)
console.log(subStr); // "JavaScript"
```

Neste exemplo, estamos extraindo a parte da string que vai do índice 0 (inclusivo) ao índice 10 (exclusivo), ou seja, "JavaScript".

Algumas regras quanto a utilização do método:

- Se você fornecer apenas o **startIndex**, o método extrai a partir desse ponto até o final da string.
- Se o valor de **startIndex** for maior que **endIndex**, o substring inverterá automaticamente os valores.
- Se um dos índices for negativo ou maior que o comprimento da string, ele será tratado como 0 (zero) ou o comprimento da string, respectivamente.

6. slice

O método `slice` em JavaScript é usado para extrair uma parte de uma string, retornando uma nova parte sem modificar o valor original. Ele pode ser aplicado tanto em strings quanto em arrays, mas o comportamento é semelhante em ambos os casos.

```
const str = 'JavaScript é divertido!'
const part = str.slice(0, 10)
console.log(part) // "JavaScript"
```

Neste exemplo, estamos extraindo do índice 0 (inclusivo) até o índice 10 (exclusivo), ou seja, "JavaScript".

O método slice é similar ao substring. O que os diferencia é que, enquanto o substring trata índices negativos como 0 (zero), o slice permite índices negativos para contar de trás para frente, ou seja, da última à primeira posição da string.

```
const str = 'JavaScript'
const sliceStr = str.slice(-6)
console.log(sliceStr) // "Script"
```

7. startsWith e endsWith

Os métodos startsWith e endsWith em JavaScript são usados para verificar se uma string começa ou termina com uma determinada sequência de caracteres. Esses métodos retornam true ou false, dependendo do resultado da verificação. Ambos são úteis para fazer validações e checar padrões em strings.

O método startsWith verifica se uma string **começa** com os caracteres fornecidos.

```
const str = 'JavaScript é incrível!'
console.log(str.startsWith('Java')) // true
console.log(str.startsWith('Script')) // false
```

Já o método endsWith verifica se uma string **termina** com os caracteres fornecidos.

```
const str = 'JavaScript é incrível!'
console.log(str.endsWith('incrível!')) // true
console.log(str.endsWith('Java')) // false
```

8. includes

O método includes em JavaScript é usado para verificar se uma string contém uma determinada substring em algum ponto da string original. Ele retorna true se a substring estiver presente, e false caso contrário.

```
const str = 'JavaScript é incrível!'
console.log(str.includes('incrível')) // true
console.log(str.includes('Python')) // false
```

Neste exemplo, a string "JavaScript é incrível!" contém a substring "incrível", então includes retorna true. Já a substring "Python" não está presente, retornando false.

9. split

O método split em JavaScript é usado para dividir uma string em várias partes, criando um array de substrings com base em um delimitador especificado. É muito útil para converter strings em arrays quando você precisa manipular ou processar diferentes partes de um texto.

```
const fruits = 'maçã, banana, laranja, uva'
const fruitArray = fruits.split(', ')
console.log(fruitArray) // ["maçã", "banana", "laranja", "uva"]
```

Se você quiser limitar o número de divisões que o split faz, pode usar o segundo parâmetro, limit.

```
const fruits = 'maçã, banana, laranja, uva'
const fruitArray = fruits.split(', ', 2)
console.log(fruitArray) // ["maçã", "banana"]
```

10. concat

O método concat em JavaScript é usado para concatenar (ou seja, juntar) duas ou mais strings, criando uma nova string como resultado. Este método não modifica as strings originais, mas retorna uma nova estrutura combinada.

```
const str1 = "Olá, "
const str2 = "mundo!"
const result = str1.concat(str2)
console.log(result) // "Olá, mundo!"
```

Aqui, a string "Olá, " foi concatenada com "mundo!", criando uma nova string "Olá, mundo!".
Você também pode passar várias strings ao mesmo tempo para concat.

```
const str1 = "Bem-vindo "
const str2 = "ao "
const str3 = "JavaScript!"
const result = str1.concat(str2, str3)
console.log(result) // "Bem-vindo ao JavaScript!"
```

Quando usado em arrays, o concat combina dois ou mais arrays, retornando um novo array contendo todos os elementos dos arrays originais.

```
const arr1 = [1, 2, 3]
const arr2 = [4, 5, 6]
const result = arr1.concat(arr2)
console.log(result) // [1, 2, 3, 4, 5, 6]
```

Métodos de Numbers

1. Math.pow

O método Math.pow em JavaScript é usado para calcular a potência de um número. Ele eleva um número base a um expoente especificado, retornando o resultado.

```
console.log(Math.pow(2, 3)) // 8 (2³ = 2 * 2 * 2)
```

2. Math.sqrt

O método Math.sqrt em JavaScript é usado para calcular a **raiz quadrada** de um número. Ele retorna o número que, ao ser multiplicado por si mesmo, resulta no número original.

```
console.log(Math.sqrt(16)) // 4 (pois 4 * 4 = 16)
```

3. Math.abs

O método Math.abs em JavaScript retorna o **valor absoluto** de um número, ou seja, o valor numérico sem seu sinal (positivo ou negativo). Ele converte números negativos para positivos e mantém números positivos inalterados.

```
console.log(Math.abs(-5)) // 5
console.log(Math.abs(3)) // 3
```

4. parseFloat

O método parseFloat em JavaScript é usado para analisar uma string e retornar um número de ponto flutuante (decimal). Ele tenta extrair um número da string, começando da esquerda, e ignora qualquer caractere que não faça parte de um número após o primeiro número válido.

```
console.log(parseFloat('3.14')) // 3.14
console.log(parseFloat("10.5abc")) // 10.5 (ignora o texto após o número)
console.log(parseFloat("abc10.5")) // NaN (Não é um número)
console.log(parseFloat("4.5e2")) // 450 Notação científica (4.5 * 10^2)
```

5. parseInt

O método parseInt em JavaScript é usado para analisar uma string e retornar um número inteiro. Ele lê a string da esquerda para a direita e converte os caracteres em um número inteiro, parando quando encontra um caractere que não pode ser parte de um número.

```
console.log(parseInt('42')) // 42
console.log(parseInt('42.5')) // 42 (ignora casas decimais)
console.log(parseInt('10.5abc')) // 10 (ignora o decimal e o texto)
console.log(parseInt('abc10')) // NaN (Não é um número)
console.log(parseInt('1010', 2)) // 10 (converte de binário para decimal)
console.log(parseInt('1A', 16)) // 26 (converte de hexadecimal para decimal)
```

6. Number

O construtor Number em JavaScript é uma função que pode ser usada para converter valores em números. Ele é utilizado tanto como uma função para conversão quanto como um objeto para trabalhar com números.

```
console.log(Number('42')) // 42
console.log(Number("3.14")) // 3.14
console.log(Number("   10   ")) // 10 (ignora os espaços em branco)
console.log(Number("42abc")) // NaN (Não é um número)
```

7. Number.isInteger

O método Number.isInteger em JavaScript é utilizado para verificar se um valor fornecido é um número inteiro. Ele retorna true se o valor é um número inteiro e false caso contrário.

```
console.log(Number.isInteger(4)) // true
console.log(Number.isInteger(3.14)) // false
```

8. Number.isNaN

O método isNaN em JavaScript é usado para verificar se um valor é **NaN** (Not-a-Number). Esse método é útil para determinar se uma operação matemática resultou em um valor inválido ou não numérico.

```
const inteiro = parseInt('10')
const decimal = parseFloat('10.5')
const invalido = Number('abc')

console.log(Number.isNaN(inteiro))  // false
console.log(Number.isNaN(decimal))  // false
console.log(Number.isNaN(invalido)) // true
```

9. Math.floor

O método Math.floor em JavaScript é usado para arredondar um número para **baixo**, retornando o maior inteiro menor ou igual ao número fornecido. Isso significa que ele elimina a parte decimal e arredonda para o inteiro mais próximo que está abaixo do valor original.

```
console.log(Math.floor(3.7)) // 3
console.log(Math.floor(5.9)) // 5
```

10. Math.ceil

O método Math.ceil em JavaScript é usado para arredondar um número para **cima**, retornando o menor inteiro maior ou igual ao número fornecido. Isso significa que ele elimina a parte decimal e arredonda para o inteiro mais próximo que está acima do valor original.

```
console.log(Math.ceil(3.1)) // 4
console.log(Math.ceil(5.9)) // 6
```

11. Math.round

O método Math.round em JavaScript é usado para arredondar um número para o inteiro mais próximo. Ele segue a regra de arredondamento padrão, onde números com parte decimal igual ou superior a 0,5 são arredondados para cima, e números com parte decimal inferior a 0,5 são arredondados para baixo.

```
console.log(Math.round(3.2)) // 3
console.log(Math.round(3.5)) // 4
console.log(Math.round(3.8)) // 4
```

12. toFixed

O método toFixed em JavaScript é utilizado para formatar um número com um número específico de casas decimais, retornando uma string representando o número formatado. Ele é especialmente útil para lidar com questões de precisão em cálculos financeiros ou ao exibir resultados numéricos.

```
let num = 3.14159
console.log(num.toFixed(2)) // "3.14"
console.log(num.toFixed(4)) // "3.1416" (arredondado para cima)
```

13. toPrecision

O método toPrecision em JavaScript é usado para formatar um número para uma representação em string com um número específico de dígitos significativos. Esse método é útil quando você precisa controlar a precisão de um número e não apenas o número de casas decimais, como é o caso do toFixed.

```
let num = 123.456

console.log(num.toPrecision(2)) // "1.2e+2" (notação científica)
console.log(num.toPrecision(4)) // "123.5" (arredondado para 4
dígitos significativos)
console.log(num.toPrecision(6)) // "123.456" (sem arredondamento)
```

14. Math.max

O método Math.max em JavaScript é usado para encontrar **o maior** valor entre os números fornecidos como argumentos. Ele retorna o maior número dentre os argumentos passados e, se nenhum argumento for fornecido, retorna -Infinity. Se algum dos argumentos não for um número, será convertido para um número, e se não puder ser convertido, será tratado como NaN.

```
console.log(Math.max(1, 2, 3)) // 3
console.log(Math.max(10, 5, 20, 15)) // 20
console.log(Math.max()) // -Infinity
console.log(Math.max(1, 2, NaN)) // NaN (NaN é ignorado no cálculo)
console.log(Math.max(1, 2, 3, "4")) // 4 (string "4" é convertida
para número)
```

15. Math.min

O método Math.min em JavaScript é usado para encontrar **o menor** valor entre os números fornecidos como argumentos. Ele retorna o menor número dentre os argumentos passados e, se nenhum argumento for fornecido, retorna Infinity. Se algum dos argumentos não for um número, será convertido para um número, e se não puder ser convertido, será tratado como NaN.

```
console.log(Math.min(1, 2, 3)) // 1
console.log(Math.min(10, 5, 20, 15)) // 5
console.log(Math.min()) // -Infinity
console.log(Math.min(1, 2, NaN)) // NaN (NaN é ignorado no cálculo)
console.log(Math.min(1, 2, 3, '4')) // 1 (string "4" é convertida
para número)
```

16. Math.random

O método Math.random em JavaScript é usado para gerar um número pseudo-aleatório entre 0 (inclusivo) e 1 (exclusivo). Isso significa que o valor retornado pode ser 0, mas nunca será igual a 1. É frequentemente utilizado em situações onde você precisa de aleatoriedade, como em jogos, simulações ou em qualquer aplicação que necessite de variabilidade.

```
let randomValue = Math.random()
console.log(randomValue) // Exemplo de saída: 0.54321
```

Se você deseja um número inteiro aleatório entre um valor mínimo e um valor máximo, você pode usar:

```
function getRandomIntInRange(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min
}

// Exemplo de uso:
console.log(getRandomIntInRange(1, 10)) // Exemplo de saída: 7
```

A aleatoriedade gerada por Math.random não é verdadeiramente aleatória; é chamada de "**pseudo-aleatória**". Isso significa que, embora pareça aleatória, a sequência de números gerados pode ser repetida.

Métodos de Arrays

1. push

O método push em JavaScript é usado para adicionar um ou mais elementos ao final de um array. Ele modifica o array original e retorna o novo comprimento do array após a adição dos elementos. Este método é muito útil para construir listas dinâmicas e manipular dados em arrays.

```
const frutas = ['maçã', 'banana']

frutas.push('laranja')
frutas.push('uva', 'melancia')

console.log(frutas)
// [ 'maçã', 'banana', 'laranja', 'uva', 'melancia' ]
```

2. unshift

O método unshift em JavaScript é usado para adicionar um ou mais elementos ao início de um array. Assim como o método push, ele modifica o array original e retorna o novo comprimento do array após a adição dos elementos. Este método é útil quando você precisa inserir elementos na frente de uma lista ou coleção.

```
const frutas = ['banana', 'laranja']

frutas.unshift('maçã')
frutas.unshift('uva', 'melancia')

console.log(frutas)
// [ 'uva', 'melancia', 'maçã', 'banana', 'laranja' ]
```

3. pop

O método pop em JavaScript é usado para remover o último elemento de um array. Ele modifica o array original e retorna o elemento que foi removido. Se o array estiver vazio, pop retornará undefined. Esse método é útil quando você precisa remover e obter o último item de uma lista ou coleção.

```
const frutas = ['maçã', 'banana', 'laranja']

const frutaRemovida = frutas.pop()

console.log(frutaRemovida) // "laranja"
console.log(frutas) // ["maçã", "banana"]
```

4. shift

O método shift em JavaScript é usado para remover o primeiro elemento de um array. Ele modifica o array original e retorna o elemento que foi removido. Se o array estiver vazio, shift retornará undefined. Esse método é útil quando você precisa retirar o primeiro item de uma lista ou coleção.

```
const frutas = ['maçã', 'banana', 'laranja']

const frutaRemovida = frutas.shift()

console.log(frutaRemovida) // "maçã"
console.log(frutas) // ["banana", "laranja"]
```

5. filter

O método filter em JavaScript é usado para criar um novo array contendo todos os elementos de um array original que satisfazem uma condição específica definida por uma função de teste. Este método não altera o array original e é muito útil para filtrar dados com base em critérios definidos.

```
const pessoas = [
  { nome: 'João', idade: 25 },
  { nome: 'Maria', idade: 30 },
  { nome: 'Pedro', idade: 20 },
]

let pessoasMaisDeVinte = pessoas.filter((pessoa) => {
  return pessoa.idade > 20
})

console.log(pessoasMaisDeVinte)
// [{ nome: "João", idade: 25 }, { nome: "Maria", idade: 30 }]
```

6. find

O método find em JavaScript é usado para retornar o primeiro elemento de um array que satisfaz uma condição específica definida por uma função de teste. Se nenhum elemento satisfizer a condição, o método retorna undefined. Esse método é útil quando você deseja localizar um único elemento em um array com base em um critério.

```
const pessoas = [
  { nome: 'João', idade: 25 },
  { nome: 'Maria', idade: 30 },
  { nome: 'Pedro', idade: 20 },
]

let pessoaMaria = pessoas.find((pessoa) => {
  return pessoa.nome === 'Maria'
})

let pessoaJose = pessoas.find((pessoa) => {
  return pessoa.nome === 'José'
})

console.log(pessoaMaria) // { nome: "Maria", idade: 30 }
console.log(pessoaJose) // undefined
```

7. findIndex

O método findIndex em JavaScript é usado para retornar o índice do primeiro elemento de um array que satisfaz uma condição específica definida por uma função de teste. Se nenhum elemento satisfizer a condição, o método retorna -1. Esse método é útil quando você deseja encontrar a posição de um elemento em um array com base em um critério.

```
const pessoas = [
  { nome: 'João', idade: 25 },
  { nome: 'Maria', idade: 30 },
  { nome: 'Pedro', idade: 20 },
]

let indicePessoaMaria = pessoas.findIndex((pessoa) => {
  return pessoa.nome === 'Maria'
})

let indicePessoaJose = pessoas.findIndex((pessoa) => {
  return pessoa.nome === 'José'
})

console.log(indicePessoaMaria) // 1
console.log(indicePessoaJose) // -1
```

8. some

O método some em JavaScript é usado para verificar se pelo menos um elemento em um array satisfaz uma condição específica definida por uma função de teste. Ele retorna true se algum elemento atender à condição e false caso contrário. Esse método é útil quando você deseja saber se existe pelo menos um elemento que atende a um critério específico em um conjunto de dados.

```
const pessoas = [
  { nome: 'João', idade: 25 },
  { nome: 'Maria', idade: 30 },
  { nome: 'Pedro', idade: 20 },
]

const temPessoaMaiorDeVinte = pessoas.some((pessoa) => {
    return pessoa.idade > 20
})

console.log(temPessoaMaiorDeVinte) // true (porque Maria e João têm mais de 20 anos)
```

9. every

O método every em JavaScript é usado para verificar se todos os elementos em um array satisfazem uma condição específica definida por uma função de teste. Ele retorna true se todos os elementos atenderem à condição e false se pelo menos um não atender. Esse método é útil quando você deseja garantir que todos os elementos em um conjunto de dados cumpram um critério específico.

```
const pessoas = [
  { nome: 'João', idade: 25 },
  { nome: 'Maria', idade: 30 },
  { nome: 'Pedro', idade: 20 },
]

const todasMaiorIdade = pessoas.every((pessoa) => {
  return pessoa.idade >= 18
})

console.log(todasMaiorIdade) // true (todas as pessoas são maiores de idade)
```

10. includes

O método includes em JavaScript é usado para verificar se um array contém um determinado elemento. Ele retorna true se o elemento for encontrado e false caso contrário. Esse método é útil para determinar a presença de um valor específico dentro de um array.

```
const frutas = ['maçã', 'banana', 'laranja']

const temBanana = frutas.includes('banana')
console.log(temBanana) // true

const temManga = frutas.includes('manga')
console.log(temManga) // false
```

O método includes utiliza comparação estrita (===), o que significa que não fará coerção de tipos. Por exemplo, 1 e '1' são considerados diferentes.

11. forEach

O método forEach em JavaScript é usado para executar uma função em cada elemento de um array. É uma maneira simples e eficaz de iterar sobre os elementos, permitindo que você realize operações em cada um deles. O método não retorna um novo array e, em vez disso, simplesmente executa a função fornecida para cada item.

```
const pessoas = [
  { nome: 'João', idade: 25 },
  { nome: 'Maria', idade: 30 },
  { nome: 'Pedro', idade: 20 },
]

pessoas.forEach((pessoa) => {
  console.log(`${pessoa.nome} tem ${pessoa.idade} anos.`)
})
// Saída:
// João tem 25 anos.
// Maria tem 30 anos.
// Pedro tem 20 anos.
```

12. map

O método map em JavaScript é usado para criar um novo array aplicando uma função a cada elemento de um array original. Esse método não altera o array original e é muito útil para transformar dados, como converter uma lista de valores em outra forma.

```
const pessoas = [
  { nome: 'João', idade: 25 },
  { nome: 'Maria', idade: 30 },
  { nome: 'Pedro', idade: 20 },
]

const nomes = pessoas.map((pessoa) => {
  return pessoa.nome
})

console.log(nomes) // ["João", "Maria", "Pedro"]
```

13. reduce

O método reduce em JavaScript é usado para reduzir um array a um único valor, aplicando uma função de callback a cada elemento do array, acumulando o resultado ao longo do caminho. É especialmente útil para operações que envolvem somas, concatenações ou transformações complexas.

```
const pessoas = [
  { nome: 'João', idade: 25 },
  { nome: 'Maria', idade: 30 },
  { nome: 'Pedro', idade: 20 },
]

const somaIdades = pessoas.reduce((acc, pessoa) => {
  return acc + pessoa.idade
}, 0)

console.log(somaIdades) // 75
```

14. reverse

O método reverse em JavaScript é usado para inverter a ordem dos elementos de um array. Ele modifica o array original e também retorna o array invertido.

```
const numeros = [1, 2, 3, 4, 5]

const numerosInvertidos = numeros.reverse()

console.log(numerosInvertidos) // [5, 4, 3, 2, 1]
console.log(numeros) // [5, 4, 3, 2, 1] (o array original também é modificado)
```

15. join

O método join em JavaScript é usado para unir todos os elementos de um array em uma única string. O método converte cada elemento do array em uma string e depois os concatena, utilizando um separador especificado (ou uma vírgula, por padrão).

```
const numeros = [1, 2, 3, 4, 5]

console.log(numeros.join('-')) // "1-2-3-4-5"
console.log(numeros.join(', ')) // "1, 2, 3, 4, 5"
console.log(numeros.join('')) // "12345"
```

16. slice

O método slice em JavaScript é usado para copiar uma parte de um array e retornar um novo array contendo os elementos selecionados. Ele não modifica o array original.

```
const frutas = ['maçã', 'banana', 'laranja', 'uva']

const subArray = frutas.slice(2)

console.log(subArray) // ["laranja", "uva"]
```

17. splice

O método splice em JavaScript é usado para **adicionar, remover ou substituir** elementos em um array. Ao contrário de slice que não altera o array original, splice **modifica o array original**.

Vejo os exemplos de aplicabilidade do splice:

a) Remover elementos de um array:

```
const numeros = [1, 2, 3, 4, 5]

const removidos = numeros.splice(2, 2) // Removerá 2 elementos a partir do índice 2

console.log(numeros) // [1, 2, 5] (array original é modificado)
console.log(removidos) // [3, 4] (elementos removidos)
```

b) Adicionar elementos a um array:

```
const letras = ['a', 'b', 'c', 'd']

letras.splice(2, 0, 'x', 'y') // Insere 'x' e 'y' no índice 2

console.log(letras) // ['a', 'b', 'x', 'y', 'c', 'd']
```

c) Substituir elementos em um array:

```
const cores = ['vermelho', 'verde', 'azul']

cores.splice(1, 1, 'amarelo') // Substitui 'verde' por 'amarelo'

console.log(cores) // ['vermelho', 'amarelo', 'azul']
```

d) Remover todos os elementos a partir de um índice:

```
const frutas = ['maçã', 'banana', 'laranja', 'uva']

frutas.splice(1) // Remove todos os elementos a partir do índice 1

console.log(frutas) // ["maçã"]
```

e) Remover elementos sem adicionar novos:

```
const numeros = [10, 20, 30, 40, 50]

numeros.splice(1, 3) // Removerá 3 elementos a partir do índice 1

console.log(numeros); // [10, 50]
```

18. sort

O método sort em JavaScript é usado para **ordenar os elementos de um array.** Ele modifica o array original, ordenando os elementos de acordo com uma função de comparação (opcional). Se essa função não for fornecida, os elementos serão convertidos em strings e ordenados com base na ordem lexicográfica (alfabética).

Sintaxe:

```
array.sort([compareFunction])
```

A compareFunction (opcional) deve ser uma função que define a ordem de classificação. Recebe dois argumentos, a e b, que são dois elementos do array, e deve retornar:

- Um valor negativo, se a for menor que b.
- Zero, se a for igual a b.
- Um valor positivo, se a for maior que b.

Veja alguns exemplos de aplicabilidade do método sort:

a) Ordenação alfabética padrão (lexicográfica):

```
const letras = ['d', 'a', 'c', 'b']

letras.sort()

console.log(letras) // ['a', 'b', 'c', 'd']
```

b) Ordenando números sem função de comparação (comportamento inesperado):

```
const numeros = [10, 5, 40, 25]

numeros.sort()

console.log(numeros) // [10, 25, 40, 5] (ordem lexicográfica, não numérica)
```

c) Ordenando números corretamente com uma função de comparação:

```
const numeros = [10, 5, 40, 25]

numeros.sort((a, b) => a - b) // Ordem crescente

console.log(numeros) // [5, 10, 25, 40]
```

d) Ordenando em ordem decrescente:

```
const numeros = [10, 5, 40, 25]

numeros.sort((a, b) => b - a) // Ordem decrescente

console.log(numeros) // [40, 25, 10, 5]
```

e) Ordenando um array de objetos por uma propriedade:

```
const pessoas = [
  { nome: 'João', idade: 25 },
  { nome: 'Maria', idade: 30 },
  { nome: 'Pedro', idade: 20 },
]


pessoas.sort((a, b) => a.idade - b.idade)

console.log(pessoas)
```

```
// [
//   { nome: 'Pedro', idade: 20 },
//   { nome: 'João', idade: 25 },
//   { nome: 'Maria', idade: 30 }
// ]
```

f) Ordenando strings ignorando maiúsculas e minúsculas:

```
const nomes = ['João', 'ana', 'Pedro', 'maria']

nomes.sort((a, b) => a.localeCompare(b, undefined, { sensitivity:
'base' }))

console.log(nomes) // ['ana', 'João', 'maria', 'Pedro']
```

BONS ESTUDOS

growdev