Aula 6

- Laço de repetição (for of)
- Funções
- Metodos de array

Laço de Repetição (for of)

O laço de repetição `for of` é uma estrutura de controle que permite percorrer elementos de um iterável, como um array, Map, Set, entre outros. Ele facilita a iteração sobre os elementos sem a necessidade de acompanhar o índice ou utilizar o método `forEach`.

Exemplo de Uso com Array

```
const nomes = ['João', 'Maria', 'Pedro'];

for (const nome of nomes) {
  console.log(nome);
}
```

Exemplo de Uso com Map

```
const alunos = new Map();
alunos.set(1, 'João');
alunos.set(2, 'Maria');
alunos.set(3, 'Pedro');

for (const [id, nome] of alunos) {
  console.log(`ID: ${id}, Nome: ${nome}`);
}
```

O laço `for of` também pode ser utilizado com outros tipos de iteráveis como Strings e outros.

Exemplo de Uso com String

```
const frase = 'Olá, mundo!';

for (const letra of frase) {
  console.log(letra);
}
```

Exemplo de Uso com Objeto

```
const pessoa = {
  nome: 'Maria',
  idade: 30,
  cidade: 'São Paulo'
};

for (const propriedade of Object.keys(pessoa)) {
  console.log(`${propriedade}: ${pessoa[propriedade]}`);
}
```

Funções

As funções são blocos de código que podem ser reutilizados para executar uma tarefa específica. Elas permitem agrupar um conjunto de instruções em um único bloco, facilitando a organização e a reutilização do código.

Function vs Arrow Function

Existem duas maneiras de declarar funções em JavaScript: usando a palavra-chave `function` ou utilizando a sintaxe de função de seta (`arrow function`). A diferença principal entre elas é a forma como lidam com o escopo do `this`.

```
// Function
function somar(a, b) {
  return a + b;
}

// Arrow Function
const multiplicar = (a, b) => a * b;
```

Funções em Variáveis

Em JavaScript, as funções também podem ser atribuídas a variáveis. Isso permite passá-las como argumentos para outras funções, armazená-las em estruturas de dados e reutilizá-las de forma flexível.

```
const saudacao = function(nome) {
  console.log(`Olá, ${nome}!`);
};

saudacao('Maria');
```

Função Autoexecutável

Uma função autoexecutável (ou IIFE - Immediately Invoked Function Expression) é uma função que é executada imediatamente após ser definida. Elas são úteis para criar um escopo isolado e evitar poluição do escopo global.

```
(function() {
  const mensagem = 'Olá, mundo!';
  console.log(mensagem);
})();
```

Função Recursiva

Uma função recursiva é uma função que chama a si mesma dentro de sua própria definição. Isso permite resolver problemas de forma iterativa, dividindo-os em subproblemas menores.

```
function calcularFatorial(n) {
  if (n === 0) {
    return 1;
  }

  return n * calcularFatorial(n - 1);
}

const resultado = calcularFatorial(5); // 5! = 5 * 4 * 3 * 2 * 1 = 120
```

Exemplo de Funções Encadeadas

```
function somar(a, b) {
  return a + b;
}
```

```
function multiplicar(a, b) {
  return a * b;
}

function dividir(a, b) {
  return a / b;
}

const resultado = dividir(multiplicar(somar(2, 3), 4), 2);
console.log(resultado); // Output: 10
```

Neste exemplo, temos três funções: `somar`, `multiplicar` e `dividir`. Essas funções são utilizadas de forma encadeada para realizar uma série de operações matemáticas. Primeiro, os números 2 e 3 são somados, resultando em 5. Em seguida, o resultado da soma é multiplicado por 4, resultando em 20. Por fim, o resultado da multiplicação é dividido por 2, resultando em 10.

``

Metodos de Array

`push()`

O método `push()` é utilizado para adicionar um ou mais elementos ao final de um array. Ele modifica o array original e retorna o novo tamanho do array após a adição dos elementos.

Sintaxe

```
array.push(valor1[, valor2[, ..., valorN]])
```

Exemplo

```
const numbers = [1, 2, 3];
const length = numbers.push(4, 5);
console.log(numbers); // [1, 2, 3, 4, 5]
console.log(length); // 5
```

No exemplo acima, `push()` é utilizado para adicionar os números `4` e `5` ao final do array `numbers`. O resultado é a exibição do novo array `[1, 2, 3, 4, 5]` e do novo tamanho do array `5`.

`at()`

o metodo `at()` recebe um valor inteiro e retorna o item referente ao index dele, permitindo valores positivos ou negativos. Valores negativos contam a partir do último item do array.

O método `at()` é utilizado para acessar um elemento específico de um array com base no índice fornecido como argumento. Ele retorna o elemento correspondente ao índice especificado.

Exemplos

```
const fruits = ["apple", "banana", "orange", "grape", "mango"];
console.log(fruits.at(1)); // "banana"
console.log(fruits.at(-2)); // "grape"
```

No primeiro exemplo, `at()` retorna o elemento no índice 1 do array `fruits`, que é a string `"banana"`. No segundo exemplo, `at()` retorna o elemento que está duas posições à esquerda do último elemento do array `fruits`, que é a string `"grape"`.

`every()`

O método `every()` é utilizado para testar se todos os elementos de um array satisfazem a condição implementada pela função fornecida. A função deve retornar um valor booleano que indica se o elemento passou ou não no teste. O método retorna `true` se todos os elementos passarem no teste e `false` caso contrário.

Sintaxe

```
array.every(function(valor[, índice[, array]]) {
    // código a ser executado no elemento
}[, thisArg])
```

Exemplos

```
const numbers = [1, 2, 3, 4, 5];
const even = numbers.every(num => num % 2 === 0);
console.log(even); // false

const words = ["apple", "banana", "cherry", "date"];
const longEnough = words.every(word => word.length >= 5);
console.log(longEnough); // true
```

No primeiro exemplo, `every()` é utilizado para testar se todos os números do array `numbers` são pares. Como o array contém um número ímpar (o número 1), o resultado retornado é `false`. No segundo exemplo, `every()` é utilizado para testar se todas as palavras do array `words` têm pelo menos 5 caracteres. Como todas as palavras atendem a essa condição, o resultado é `true`.

`filter()`

O método `filter()` é utilizado para criar um novo array contendo todos os elementos que passaram em um teste implementado pela função fornecida. A função `filter()` deve retornar true ou false para cada elemento do array original, e os elementos que retornaram true são incluídos no novo array.

Sintaxe

```
array.filter(function(valor[, índice[, array]]) {
    // código a ser executado no elemento
}[, thisArg])
```

Exemplos

```
const numbers = [1, 2, 3, 4, 5];
const evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers); // [2, 4]

const words = ["apple", "banana", "cherry", "date"];
const longWords = words.filter(word => word.length >= 6);
console.log(longWords); // ["banana", "cherry"]
```

No primeiro exemplo, `filter()` é utilizado para criar um novo array contendo apenas os números pares do array `numbers`. Como somente os números 2 e 4 são pares, o resultado é um novo array contendo apenas esses dois números. No segundo

exemplo, `filter()` é utilizado para criar um novo array contendo apenas as palavras do array `words` que têm pelo menos 6 caracteres. Como somente as palavras "banana" e "cherry" têm pelo menos 6 caracteres, o resultado é um novo array contendo essas duas palavras.

`find()`

O método `find()` é utilizado para buscar um elemento específico em um array, com base em uma condição implementada pela função fornecida. A função `find()` deve retornar um valor booleano que indica se o elemento passou ou não no teste. O método retorna o primeiro elemento que passar no teste ou `undefined` caso nenhum elemento passar.

Sintaxe

```
array.find(function(valor[, índice[, array]]) {
    // código a ser executado no elemento
}[, thisArg])
```

Exemplos

```
const numbers = [1, 2, 3, 4, 5];
const even = numbers.find(num => num % 2 === 0);
console.log(even); // 2

const words = ["apple", "banana", "cherry", "date"];
const longEnough = words.find(word => word.length >= 6);
console.log(longEnough); // "banana"
```

No primeiro exemplo, `find()` é utilizado para buscar o primeiro número par do array `numbers`. Como o primeiro número par é o número 2, o resultado retornado é `2`. No segundo exemplo, `find()` é utilizado para buscar a primeira palavra do array `words` que tem pelo menos 6 caracteres. Como a primeira palavra com pelo menos 6 caracteres é a palavra "banana", o resultado retornado é `"banana"`.

`forEach()`

O método `forEach()` é utilizado para executar uma função fornecida uma vez para cada elemento do array. A função é executada com três argumentos: o valor do

elemento atual, o índice do elemento atual e o array em que o método `forEach()` foi chamado.

Sintaxe

```
array.forEach(function(valorAtual[, indice[, array]]) {
    // código a ser executado no elemento
}[, thisArg])
```

Exemplo

```
const numbers = [1, 2, 3, 4, 5];
numbers.forEach(num => console.log(num));
```

No exemplo acima, `forEach()` é utilizado para executar a função `console.log()` para cada elemento do array `numbers`. O resultado é a exibição de cada número no console, um de cada vez.

`map()`

O método `map()` é utilizado para criar um novo array preenchido com os resultados da chamada de uma função fornecida em cada elemento no array de chamada. Em outras palavras, ele nos permite aplicar uma função a cada elemento do array original e criar um novo array contendo os resultados dessas operações.

Sintaxe

```
array.map(function(valorAtual[, indice[, array]]) {
    // código a ser executado no elemento
}[, thisArg])
```

Exemplo

```
const numbers = [1, 2, 3, 4, 5];
const squares = numbers.map(num => num * num);
console.log(squares); // [1, 4, 9, 16, 25]
```

No exemplo acima, `map()` é utilizado para criar um novo array `squares` contendo os quadrados de cada elemento do array `numbers`. A função `num => num * num` é aplicada a cada elemento do array original, criando um novo array contendo os resultados dessas operações.

Outro exemplo

```
const words = ["apple", "banana", "cherry", "date"];
const upperCaseWords = words.map(word => word.toUpperCase());
console.log(upperCaseWords); // ["APPLE", "BANANA", "CHERRY", "DATE"]
```

No exemplo acima, `map()` é utilizado para criar um novo array `upperCaseWords` contendo as palavras em letras maiúsculas do array `words`. A função `word => word.toUpperCase()` é aplicada a cada elemento do array original, criando um novo array contendo os resultados dessas operações.

Veja mais métodos de array:

📘 Aula métodos de array

Exercícios para praticar métodos de array:

📘 Exercicios métodos de array