DESAFIO LÓGICA AFIADA 21 dias

by CódigoFonte

RESOLUÇÕES

DIA 18

DESAFIO 01

Vamos ver como ficou o nosso código para ordenar a playlist de música pelo título e pelo número de reproduções.

```
    // Ordena as músicas por Nome (utilizando Bubble Sort)
    ordenarPorNome: function() {
        let n = this.musicas.length;
        let trocado;

        do {
            trocado = false;
            for (let i = 0; i < n - 1; i++) {
                if (this.musicas[i].nome > this.musicas[i +
1].nome) {
                    let temp = this.musicas[i];
                    this.musicas[i] = this.musicas[i + 1];
                    this.musicas[i + 1] = temp;
                    trocado = true;
                }
            }
        } while (trocado);

        console.log("🔤 Playlist ordenada por Nome.");
    },

    // Ordena as músicas por Número de Reproduções (utilizando
Selection Sort)
    ordenarPorReproducoes: function() {
        let n = this.musicas.length;

        for (let i = 0; i < n - 1; i++) {
            let maxIndex = i;

            for (let j = i + 1; j < n; j++) {
                if (this.musicas[j].reproducoes >
this.musicas[maxIndex].reproducoes) {
                    maxIndex = j;
                }
            }

            let temp = this.musicas[i];
            this.musicas[i] = this.musicas[maxIndex];
            this.musicas[maxIndex] = temp;
```

```
        console.log("🔢 Playlist ordenada por Número de
Reproduções.");
    },
```

Começamos adicionando a função **ordenarPorNome** no nosso objeto literal. Só lembrando que nesse caso vamos implementar utilizando o **Bubble Sort**. Se você tiver alguma dúvida ou não lembrar completamente como funciona esse algoritmo, basta voltar na aula de ordenação.

A ideia é ir fazendo o loop em um **while** enquanto a variável **trocado** for verdadeira. Em cada iteração a gente precisa percorrer todo o array de músicas com um **for** até o penúltimo elemento e ir comparando com o próximo (utilizando o **i + 1**. Iniciamos o **for** sempre partindo do princípio que não será troca de posição, então atribuímos falso para a variável **trocado**.

Como cada elemento do array é um JSON, então precisamos acessar a chave correta para fazer a comparação de quem é maior. Nesse caso, estamos ordenando pelo título da música (ou pela chave **nome**), então verificamos se o elemento atual (na variável **i**) é maior que o elemento seguinte (ou **i + 1**). Se for então temos que trocar essas músicas de posição. E controlar para que a variável **trocado** seja verdadeira.

Por fim usamos o console para mostrar que a playlist foi ordenada pelo nome da música.

Partimos então para a **ordenarPorReproducoes**. Nesse caso vamos usar o **Selection Sort**. Eu sei que nós já vimos isso, mas é preciso exercitar em vários contextos diferentes.

Só assim a gente fixa esse conhecimento e se sente mais seguro para implementar diferentes algoritmos.

Então para conseguirmos ordenar, vamos precisar utilizar dois `for` onde o primeiro vai do início do array de músicas até o penúltimo elemento. Nele vamos precisar antes armazenar qual foi o índice com o maior valor encontrado até o momento.

O segundo `for` (interno) iniciará justamente da posição a frente que temos na variável `i` (do primeiro `for`) até o último elemento. Lembra que precisamos percorrer os elementos ainda não verificados para saber quem irá ou não trocar de posição. Para isso, precisamos justamente verificar se a posição na variável `j` (utilizando a chave `reproducoes` do JSON) é maior que o maior índice encontrado até o momento e que está na variável `maxIndex`. Caso seja então temos então um novo índice para atualizar o `maxIndex`.

Voltando para o `for` mais externo, fazemos a substituição de posição do índice `i` pelo `maxIndex` e vice e versa. Quando isso terminar, teremos as músicas ordenadas pelo número de reproduções das músicas.

Para testar é bem simples, podemos adicionar ao código já que tínhamos a chamada ao `ordenarPorNome` e depois chamaos a função `mostrarPlaylist`. O mesmo fazemos com o `ordenarPorReproducoes` e depois por `mostrarPlaylist`.

Teste e execute o código, você vai ver como funciona magicamente!

Mas calma que queremos te mostrar uma forma de resolver esse tipo de situação de outra forma!
Olha só!

```
// Função para criar uma nova música em formato JSON
function criarMusica(nome, artista, tempo) {
    return {
        nome: nome,
        artista: artista,
        reproducoes: 0,
        tempo: tempo
    };
}

// Estrutura da Playlist como um Objeto Literal
const playlist = {
    musicas: [],

    // Adiciona uma música ao início da playlist
    adicionarMusica: function(nome, artista, tempo) {
        const novaMusica = criarMusica(nome, artista, tempo);

        // Criamos um novo array e deslocamos os elementos
manualmente
        for (let i = this.musicas.length; i > 0; i--) {
            this.musicas[i] = this.musicas[i - 1];
        }
        this.musicas[0] = novaMusica;
        console.log(`🎶 Música "${nome}" adicionada à
playlist!`);
    },

    // Ordenação por Nome usando `.sort()`
    ordenarPorNomeSort: function() {
      this.musicas.sort((a, b) => {
        let nomeA = a.nome.toLowerCase();
        let nomeB = b.nome.toLowerCase();

        if (nomeA < nomeB) return -1;
        if (nomeA > nomeB) return 1;
        return 0;
      });

        console.log("🔤 Playlist ordenada por Nome (usando
sort).");
    },

    // Ordenação por Número de Reproduções usando `.sort()`
    ordenarPorReproducoesSort: function() {
        this.musicas.sort((a, b) => b.reproducoes -
a.reproducoes);
```

```
    console.log("🔢 Playlist Ordenada por Número de
Reproduções (usando sort).");
    },

    // Simula tocar uma música e aumentar a contagem de
reproduções
    tocarMusica: function(nome) {
        for (let i = 0; i < this.musicas.length; i++) {
            if (this.musicas[i].nome === nome) {
                this.musicas[i].reproducoes++;
                console.log(`🎵 Tocando:
${this.musicas[i].nome} - ${this.musicas[i].artista}
(${this.musicas[i].tempo})`);
                return;
            }
        }
        console.log(`⚠️ Música "${nome}" não encontrada.`);
    },

    // Exibe a playlist
    mostrarPlaylist: function() {
        if (this.musicas.length === 0) {
            console.log("📭 A playlist está vazia.");
        } else {
            console.log("🎶 Playlist Atual:");
            for (let i = 0; i < this.musicas.length; i++) {
                console.log(`${i + 1}. ${this.musicas[i].nome}
- ${this.musicas[i].artista} | Reproduções:
${this.musicas[i].reproducoes}`);
            }
        }
    }
};

// 🎧 Testando a playlist
playlist.adicionarMusica("Bohemian Rhapsody", "Queen",
"5:55");
playlist.adicionarMusica("Shape of You", "Ed Sheeran",
"3:53");
playlist.adicionarMusica("Blinding Lights", "The Weeknd",
"3:22");
playlist.adicionarMusica("Hotel California", "Eagles",
"6:30");
playlist.adicionarMusica("Imagine", "John Lennon", "3:07");

// Mostrar playlist antes da ordenação
playlist.mostrarPlaylist();

// Tocar algumas músicas para modificar a contagem de
reproduções
playlist.tocarMusica("Shape of You");
playlist.tocarMusica("Shape of You");
playlist.tocarMusica("Imagine");
playlist.tocarMusica("Bohemian Rhapsody");
playlist.tocarMusica("Bohemian Rhapsody");
playlist.tocarMusica("Bohemian Rhapsody");

// Mostrar playlist com as contagens atualizadas
playlist.mostrarPlaylist();

// Ordenar por Nome e mostrar (usando sort)
playlist.ordenarPorNomeSort();
```

```
playlist.mostrarPlaylist();

// Ordenar por Reproduções e mostrar (usando sort)
playlist.ordenarPorReproducoesSort();
playlist.mostrarPlaylist();
```

No JavaScript, assim como em outras linguagens, nós já temos muitos desses algoritmos implementados, então é possível executar o mesmo processo utilizando essas funções internas. Nesse caso, em todo array no JavaScript, já tem disponível um método chamado **`sort()`**.

Nesse caso, implementamos a função **`ordenarPorNomeSort()`** e o que ela faz é simplesmente utilizar o método **`sort()`** do array **`musicas`**. Como parâmetro para essa função, temos que enviar uma outra função. Podemos utilizar esse tipo de chamada, conhecida como **`arrow function`**, onde adicionamos parêntesis com os parâmetros e o **`=>`**. Como teremos mais de uma linha de código, então vamos precisar usar também chaves.

A ideia do **`sort()`** é simplesmente uma comparação entre **`a`** e **`b`**, caso um seja maior então é necessário retornar **`-1`**, senão **`1`**. Se for o caso de não ser nenhum dos casos (por exemplo igual) retornamos **`0`**. Isso, por si só já será o suficiente para ordenar. Nesse caso, veja que criamos as variáveis **`nomeA`** e **`nomeB`** que são respectivamente a chave nome do elemento passado, porém forçamos o nome da música a ficar com letras minúsculas. Assim na hora de comparar não importa se é maiúsculo ou não, ele irá ordenar.

Veja que para implementar o `ordenarPorReproducoesSort` o princípio é o mesmo, mas nós só precisamos diminuir a chave `reproducoes` de `b` com o `reproducoes` de `a`. Isso fará já retornará o resultado equivalente e a ordenação será feita. Nesse caso não precisamos usar o `return` pois como não temos chaves nessa `arrow function` a operação já é de retorno.

DESAFIO LÓGICA AFIADA 21 dias

by CódigoFonte

desafiologicaafiada.com.br