



# Fundamentos em Desenvolvimento Front End

## Capítulo 4. JavaScript Moderno

Prof. Raphael Gomide



## **Aula 4.1. Introdução ao JavaScript moderno**

## ☐ Introdução:

- ES6+.
- var x const x let.
- Arrow functions.
- Template literals.
- Default Parameters.

- ECMAScript em constante evolução.
- Navegadores acompanham de forma mais lenta.
- Risco de “quebra” da internet.
- A evolução do JavaScript é regulada pelo [TC39](#).
- Conseguimos utilizar ES6+ com transpiladores.
  - Babel.
  - TypeScript.

- **var** já foi visto anteriormente.
- **var** possui escopo amplo.
- É melhor utilizar **let**, que possui escopo reduzido.
- Utilizamos **const** para garantir imutabilidade.
- Com **const**, não é possível atribuir um novo valor à variável.
- Entretanto, é possível modificar os dados de **arrays** e **objetos**.
- Uma boa prática é sempre começar a declaração com **const** e trocar por **let** se for realmente necessário.
- **var** tem sido raramente utilizado atualmente.

# Arrow functions

- Sintaxe mais simples.
- Escrita mais declarativa e funcional.
- Exemplo:

```
const makeBeer = function beerFun(qty) {  
  return '🍺'.repeat(qty);  
}  
  
const makeWine = (qty) => '🍷'.repeat(qty);
```

Fonte: [fireship.io](https://fireship.io)

# Template literals

- Maneira mais simples e elegante de criar Strings a partir de expressões.
- Utiliza o símbolo ` (crase).
- Expressões JavaScript ficam entre \${}.

```
> const a = 1, b = 2, c = 3;  
> a = 1, b = 2, c = 3;  
> array = [1,2,3,4]  
> array3 = [...array1, ...array2];  
> array2 = [4, 5, 6];  
> array1 = [1, 2, 3];  
> array3 = [...array1, array2];  
> accounts = temp1.map(item => {return {id: ...  
> accounts = temp1.map(item => {id: item.id,...
```

- É possível atribuir valores padrão para parâmetros de funções.
- Em geral, parâmetros com valores padrão devem se situar no **final** da função.
- Acompanhe o professor para exemplos de:
  - var x const x let.
  - Arrow functions.
  - Template literals.
  - Default parameters.



- ☑ **var** deve ser evitado, pois é propenso a bugs de escopo.
- ☑ Utilize **const** por padrão para declarar variáveis.
- ☑ Utilize **let** caso seja necessário reatribuir um novo valor à variável.
- ☑ **Arrow functions** permitem escrita mais simples e funcional em comparação a funções.
- ☑ **Template literals** facilitam a escrita de Strings com expressões JavaScript.
- ☑ **Default parameters** flexibilizam a utilização de parâmetros em funções.

- ❑ Manipulação de arrays com ES6+.



## **Aula 4.2. Manipulação de arrays com ES6+**

## ☐ Principais métodos de arrays com ES6+:

- map
- filter
- forEach
- reduce
- find
- some
- every
- sort

- **map** → gera um novo array transformando os dados.
- **filter** → gera um novo array filtrando elementos com base em proposição.
- **forEach** → percorre todos os elementos do array, aplicando lógica.
- **reduce** → realiza cálculo iterativo com base nos elementos.
- **find** → encontra elementos com base em proposições.
- **some** → verifica se há pelo menos um elemento que atenda à proposição.
- **every** → verifica se todos os elementos atendem à proposição.
- **sort** → ordena os elementos com base em um critério.
- Acompanhe o professor.

- ☑ O JavaScript possui excelentes métodos para manipulação de arrays.
- ☑ Métodos podem ser customizáveis através de **callbacks**.
- ☑ É uma boa prática a utilização de arrow functions nesses métodos, para simplificar a escrita.

- ❑ Operador ... (rest/spread) e *destructuring*.



## **Aula 4.3. Operador ... e *destructuring***



- ❑ Operador ... (rest/spread).

- ❑ *Destructuring*.

# Operador ... (spread)

- Muito útil para trabalhar com arrays e objetos.
- Em arrays, este operador **espalha** os itens do array, que podem ser recuperados para compor outro array, por exemplo.
- Acompanhe o professor.

```
> const array|  
  > array3 = [...array1, ...array2];  
  > array2 = [4, 5, 6];  
  > array1 = [1, 2, 3];  
  > array3 = [...array1, array2];
```

# Operador ... (rest)

- Muito útil para trabalhar com arrays e objetos.
- Como **rest**, é comum a utilização em funções, **agrupando** os parâmetros em um array.
- Principal aplicação → permitir funções com número infinito de parâmetros.
- Acompanhe o professor.

```
> const super|
```

# Destructuring

- Facilita a escrita ao trabalhar com objetos.
- Torna o código mais claro.
- É também possível utilizar a técnica de *destructuring* com arrays, usando [].
- Acompanhe o professor.

```
> //sem des|  
← undefined
```

```
> //|  
← undefined
```

## ☑ Rest/spread:

- Espalha elementos de vetores (spread).
- Agrupa elementos em funções (rest).

## ☑ Destructuring:

- Permite uma melhor escrita e legibilidade de código.
- Compatível com arrays e objetos.

- ❑ Refatorando o projeto do Capítulo 3.



## **Aula 4.4. Refatorando o projeto do Capítulo 3**

- ❑ Refatoração do projeto do Capítulo 3.



# Refatoração do projeto do Capítulo 3

- Acompanhe o professor.
- O projeto será refatorado com ES6+.

# Conclusão

- ✓ Código **mais** organizado.
- ✓ Código **mais** elegante.
- ✓ Código **menos** compatível.

☐ Capítulo 5 – Programação Assíncrona com JavaScript.