Innominatum CONSULTORIA

Programação em JavaScript

ServiceNow - Deloitte

Rodrigo Costa

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parceiros para o Crescimento
PORTUGAL DIGITAL

Innominatum CONSULTORIA

JavaScript

Módulo 2

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

Manipulação de Dados Complexos

Sessão 8

Objetivo geral

Capacitar os formandos na utilização de técnicas avançadas de manipulação de coleções de dados, promovendo a escrita de código limpo, declarativo e performante através de High Order Functions. A sessão foca na transição para o paradigma da imutabilidade, ensinando a transformar e extrair dados sem comprometer a integridade das fontes originais, utilizando as funcionalidades de ES6+ (Spread, Rest e Destructuring).

Objetivos específicos

Ao final da sessão, os formandos deverão ser capazes de:

- Aplicar métodos de array (map, filter, reduce, find, some, every) para transformar e consultar coleções de dados;
- Explicar o conceito de imutabilidade e evitar a alteração direta de arrays e objetos;
- Utilizar os operadores spread (...) e rest para copiar, combinar e manipular arrays e objetos;
- Aplicar destructuring para extrair dados de arrays e objetos de forma clara e eficiente.

Innominatum CONSULTORIA

Métodos de Array

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

O que é uma função de alta ordem?

Uma **função de alta ordem (High Order Function)** é definida como uma **função que opera sobre outras funções**, seja recebendo uma ou mais funções como argumentos ou retornando uma nova função.

Métodos do array

Definição: É uma função interna (built-in) que já vem "embutida" em todos os Arrays no JavaScript.

- Propósito: Serve para realizar operações comuns — como adicionar, remover, transformar ou filtrar dados — sem que o programador tenha de reconstruir a lógica do zero;
- Sintaxe: É acedido através da notação de ponto (ex: meuArray.metodo());
- Conexão com HOFs: Muitos destes métodos são Funções de Alta Ordem, pois permitem passar uma função personalizada para decidir como os dados devem ser processados.

Os principais métodos do array

- forEach(): O "Executador" – Percorre a lista e executa uma ação para cada item;
- map(): O "Transformador" – Cria um novo array com os dados alterados;
- filter(): O "Filtro" – Cria um novo array apenas com os itens que passam numa regra;
- find(): O "Buscador" – Encontra e devolve o primeiro item que satisfaz uma condição;
- Reduce(): O "Acumulador" – Reduz o array inteiro a um único valor (ex: uma soma total).

Método forEach

Executa uma função (callback) para cada elemento do array. Utilizamos o forEach quando queremos realizar uma ação (como imprimir no console ou guardar numa base de dados), mas não precisamos de criar um novo array. Ele entrega três coisas à tua função em cada volta (laço):

1. O item atual (ex: o nome do produto);
2. O índice (a posição 0,1,2…);
3. O array completo (opcional).

```
// 1. Usando uma função anônima
frutas.forEach(function(fruta) {
    console.log("For Each Anônima: " + fruta);
});
```

Método map

Tal como o forEach, ele percorre o array e executa uma callback para cada item. O map() espera um retorno da callback e guarda esse valor num novo array. O array original permanece intacto, o map() entrega-te uma nova lista transformada.

```
const numeros = [1, 2, 3];
const resultado =
    numeros.map(n => n * 2);
```

Método filter

Este método tal como os anteriores irá percorrer todo o array, ele deve devolver sempre um valor booleano (true ou false), se o valor for true, é inserido no novo array, caso contrário, será descartado.

```
const pares = numeros
    .filter(n => n % 2 === 0);
```

Método find

Sua missão é procurar no array o elemento desejado e devolver true quando ele o encontrar, como resultado, será devolvido o primeiro elemento que satisfazer a condição, parando a execução do loop, caso não encontre nada, é devolvido um undefined.

```
const encontrado = numeros
    .find(n => n > 2);
```

Método some e every

Ambos retornam um valor booleano:

- every(): Retorna true se, e somente se, todos os elementos do array retornarem true;
- some(): Retorna true se existir pelo menos um elemento no array que retorne true. Ambos interrompem a iteração assim que o resultado (verdadeiro ou falso) é determinado.

```
numeros.some(n => n > 3);   // true
numeros.every(n => n > 0); // true
```

Innominatum CONSULTORIA

Introdução ao Reduce

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

O que é reduce?

O **reduce()** é frequentemente considerado o "canivete suíço" dos métodos de **array** em JavaScript. Ele é o mais poderoso (e complexo) porque, ao contrário do **map** ou **filter**, ele não retorna necessariamente um array; ele pode retornar um único valor de qualquer tipo (um número, uma string, um objeto ou até um novo array). O objetivo do reduce é reduzir uma coleção de valores a um único resultado, através de uma função acumuladora.

Sintaxe básica - reduce

```
array.reduce((acumulador, valorAtual) => {
  return novoValor;
}, valorInicial);
```

Os dois principais parâmetros do reduce são o:

- Acumulador (acc): Onde o valor vai sendo “guardado” de volta em cada volta;
- Valor atual (cur): O item que está a ser processado no momento;
- Indice e Array completam a lista, mas são opcionais.

Exemplo - reduce

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

const soma = numeros.reduce((total, n) => {
  return total + n;
}, 0);
// soma = 10
```

```
// Com retorno implícito
const soma = numeros
.reduce((total, n) => total + n, 0);
// soma = 10
```

1. total = 0, n = 1 → total = 1;
2. total = 1, n = 2 → total = 3;
3. total = 3, n = 3 → total = 6;
4. total = 6, n = 4 → total = 10.

Reduce não é só para somar

O **reduce** possui outros usos além de somar, como:

- Contar elementos;
- Calcular médias;
- Concatenar strings.

```
// Contar elementos em um array
const nomes = ["Ana", "João", "Ana"];

const total = nomes
    .reduce((contagem) => contagem + 1, 0);
// total = 3
```

Innominatum CONSULTORIA

Imutabilidade

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

O que é imutabilidade?

A **imutabilidade** refere-se à característica de valores que não podem ser alterados após a sua criação. Então o princípio da imutabilidade tem como objetivo:

- Não alterar o dado original;
- Criar novos dados a partir dos existentes.

O problema de alterar dados

Em JS arrays e objetos são referências e alterá-los diretamente afeta todas as referências e cria efeitos colaterais, o resultado são bugs difíceis de resolver. Isto é um problema por que:

- Dados partilhados mudam sem aviso;
- Funções deixam de ser previsíveis;
- Dificulta o debug.

Objeto mutável x imutável

```
const numeros = [1, 2, 3];
numeros.push(4);
// O array orignial foi alterado
```

```
const numeros = [1, 2, 3];
const novoArray = [...numeros, 4];
// O array original não foi alterado
```

Métodos imutáveis x mutáveis

Métodos seguros:

- map;
- filter;
- reduce;
- concat.

Métodos não seguros:

- push;
- pop;
- shifit;
- splice;
- sort.

Imutabilidade e previsibilidade

Uma Função Pura é o pilar central da imutabilidade e da programação funcional. Em termos simples, é uma função "previsível": ela sempre produzirá o mesmo resultado para os mesmos argumentos e não causará nenhuma alteração no mundo exterior.

Para ser considerada pura, uma função deve cumprir dois requisitos rigorosos:

1. Determinismo (Previsibilidade Total);
2. Ausência de Efeitos Secundários (Side Effects).

Spread e Rest Operators

Spread Operator – definição

O Spread Operator (Operador de Espalhamento) é representado por três pontos consecutivos ... e serve para "desempacotar" ou espalhar os elementos de um iterável (como um Array ou um Objeto) dentro de outro lugar. É uma ferramenta técnica que permite a cópia e fusão de dados de forma elegante e imutável.

Spread operator em arrays

```
// Espalhar elementos
const numeros = [1, 2, 3];
const copia = [...numeros];
// Cria uma cópia do array
// O original não é alterado
```

```
// Combinar arrays com spread
const a = [1, 2];
const b = [3, 4];

const combinado = [...a, ...b];
// Resultado = [1, 2, 3, 4]
```

Spread operator em objetos

```
// Copiar objetos
const pessoa = { nome: "Ana", idade: 30 };
const copia = { ...pessoa };
// Cria um novo objeto com as mesmas propriedades.
```

Destructuring (Array e Objetos)

Innominatum CONSULTORIA
IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parcerias para o Crescimento
PORTUGAL DIGITAL

O que é destructuring?

O Destructuring (Desestruturação) é uma funcionalidade introduzida no ES6 que permite "extrair" dados de dentro de arrays ou propriedades de objetos e atribuí-los a variáveis individuais de uma forma muito mais limpa e rápida.

Destructuring em objetos

```
const utilizador = {
  nome: "Ricardo",
  idade: 28,
  pais: "Portugal"
};
// Forma antiga:
// const nome = utilizador.nome;
// const idade = utilizador.idade;
// Com Destructuring:
const { nome, idade } = utilizador;
console.log(nome);  // "Ricardo"
console.log(idade); // 28
```

Em vez de acederes a cada propriedade usando o ponto (objeto.propriedade), "retiras" o que precisas usando chaves { }.

Destructuring em arrays

```
const cores = ["Azul", "Vermelho", "Verde"];
// Extraímos os dois primeiros elementos
const [primeira, segunda] = cores;
console.log(primeira); // "Azul"
console.log(segunda);  // "Vermelho"
```

Aqui, a ordem é o que importa.

Usamos colchetes [ ] para atribuir

nomes aos elementos com base

na sua posição.

Ignorar valores no arrays

```
const [ , segundo ] = numeros;
```

É possível pular elementos do array, é bastante útil quando só interessa uma parte do array.

Destructuring + spread

```
const [primeiro, ...resto] = numeros;
```

Extrai primeiro e guarda o resto

Destructuring em parâmetros de função

```
function mostrarNome({ nome }) {
  console.log(nome);
}
```

Muito usado em callbacks e APIs.

Rest operator em funções

```
// Recolher parâmetros
function somar(...valores) {
  return valores.reduce((t, n) => t + n, 0);
}
// Todos os argumentos viram um array
```

Síntese

- Métodos de Array (High Order Functions);
- Método reduce;
- Imutabilidade;
- Spread & Rest Operators (...);
- Destructuring.

Conclusão

Innominatum CONSULTORIA

Programação em JavaScript

ServiceNow - Deloitte

Rodrigo Costa

IPCA POLITÉCNICO DO CÁVADO E DO AVE
INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU
IAPMEI Parceiros para o Crescimento
PORTUGAL DIGITAL