Algoritmia e Programação

Estruturas de Dados
Arrays, Objetos e Coleções

Marta Martinho 2025

Objetivos

- Compreender o que é uma estrutura de dados
- Utilizar arrays em JavaScript de forma correta e consciente
- Aplicar operações fundamentais sobre arrays
- Implementar algoritmos simples de ordenação e pesquisa
- Criar objetos e classes
- Trabalhar com arrays de objetos
- Reconhecer outras estruturas de dados e os seus contextos de uso
-

ARRAYS - O que é um Array?

- Um **array** é:
  - Uma estrutura de dados linear
  - Que armazena vários valores
  - Indexados a partir de 0
  - Um array em JavaScript é um objeto especial
-

```
const numeros = [5, 2, 9, 1];
```

```
const numeros = [];

numeros[0] = 5;
numeros[1] = 2;
numeros[2] = 9;
numeros[3] = 1;
```

ARRAYS - Particularidades dos Arrays em JavaScript

- Os **arrays** em JavaScript:
  - São dinâmicos (tamanho variável)
  - Podem conter tipos mistos
  - São objetos
  - São passados nas funções por referência
  - Encapsulam muitos métodos

```
const a = [1, "texto", true];
```

ARRAYS - Índices e Acesso

- Índices inválidos devolvem **undefined**
- Não gera erro

```
const numeros = [10, 20, 30];

console.log(numeros[0]); // 10
console.log(numeros[2]); // 30
```

ARRAYS - Tamanho do Array

- **length**:
    - Não é o último índice
    - Pode ser alterado (não recomendado)

```
numeros.length = 1; // corta o array
```

ARRAYS - Percorrer um Array

```
for (let i = 0; i < numeros.length; i++) {
  console.log(numeros[i]);
}
```

```
for (let n in numeros){
    console.log(numeros[n]);
}
```

```
for (let n of numeros) {
  console.log(n);
}
```

ARRAYS - Exercício 1

- **Criar um array de inteiros e calcular a soma dos elementos.**
-

```
const numeros = [2, 4, 6, 8];
let soma = 0;

for (let i = 0; i < numeros.length; i++) {
  soma += numeros[i];
}

console.log(soma);
```

ARRAYS - Exercício 2

- **Criar um programa que:**
    - Guarde 10 números inteiros num array
    - **Calcule:**
        - maior valor
        - menor valor
        - média

ARRAYS - Pesquisa de Elementos

- **Pesquisar significa:**
  - Verificar se um valor existe
  - Descobrir a sua posição
  - Exemplo - Pesquisa Sequencial (Linear)

- JavaScript oferece métodos,

mas vamos implementar o algoritmo

```
function pesquisaLinear(arr, valor) {
    let i = 0;

    while (i < arr.length && arr[i] != valor) {
        i++;
    }

    if (i >= arr.length) {
        return -1
    }

    return 1;
}
```

ARRAYS - Exercício 3

- **Modificar a pesquisa linear para:**
    - Contar quantas vezes um valor aparece
    - Não parar na primeira ocorrência

ARRAYS - ORDENAÇÃO

- **Ordenar significa:**
  - Reorganizar os dados segundo um critério
  - Ascendente ou descendente
- Ordenação por Seleção
  - Ideia:
    - Seleciona o menor/maior
    - Coloca-o na posição correta
    - Compara arr[i] com todos os restantes
-

```
function selectionSort(arr) {
  for (let i = 0; i < arr.length - 1; i++) {
    for (let j = i + 1; j < arr.length; j++) {
      if (arr[j] > arr[i]) {
        let temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
      }
    }
  }
}
```

ARRAYS - Métodos Essenciais

- **Métodos mais importantes**:
    - push, pop
    - shift, unshift
    - splice
    - slice
    - indexOf, includes, find

ARRAYS - Adicionar e Remover Elementos (métodos)

- **shift e unshift** são mais caros computacionalmente
- Removem / adicionam elementos no início do array.
- Características:
  - Alteram o array original
  - Todos os elementos são reposicionados
  - Custo elevado em arrays grandes
- **push e pop** adicionam e removem do fim respetivamente

```
const a = [];

a.push(10);     // adiciona no fim
a.pop();        // remove do fim

a.unshift(5);   // adiciona no início
a.shift();      // remove do início
```

Exercício rápido : Remover o primeiro elemento de um array apenas se o array não estiver vazio.

ARRAYS - Adicionar e Remover Elementos (métodos)

- **splice()**
- Permite:
  - remover
  - inserir
  - substituir elementos

```
let numeros = [10, 20, 30, 40];
numeros.splice(1, 2);
// remove 2 elementos a partir do índice 1
// resultado: [10, 40]

//inserir (0 no segundo argumento)
numeros.splice(1, 0, 15);
// [10, 15, 40]

//substituir (1 no segundo argumento)
numeros.splice(1, 1, 99);
// [10, 99, 40]
```

ARRAYS - Copiar array (métodos)

- **slice()**
  - Cria uma cópia parcial ou integral do array.

```
//copia integral
const original = [1, 2, 3];
const copia = original.slice();
```

```
const numeros = [10, 20, 30, 40];
const parte = numeros.slice(1, 3);//do indice 1 ao 3

console.log(parte);    // [20, 30]
console.log(numeros);  // não é alterado
```

ARRAYS - Pesquisa (métodos)

- **Indexof e includes**
  - devolve o índice da primeira ocorrência

  de um valor num array.
  - Se o elemento não existir, devolve -1.
  - Útil para verificar se um valor existe e saber onde está

```
const numeros = [5, 10, 15];

numeros.indexOf(10); // 1
numeros.indexOf(99); // -1

numeros.includes(15); // true
numeros.includes(99); // false
```

Exercício rápido : Verificar se um número existe no array sem usar includes().

ARRAYS - Pesquisa (métodos)

- **filter**
  - Cria um novo array com todos os elementos que passam no teste indicado.
  - Se nenhum elemento cumprir, devolve um **array vazio []**
- **find**
  - devolve o primeiro elemento que satisfaz uma condição lógica.
  - Se nenhum cumprir, devolve undefined.

```
const numeros = [3, 7, 10, 15, 8, 20];

const maioresQue10 = numeros.filter((n) => n > 10);

console.log(maioresQue10); //[15, 20]
```

```
const numeros = [3, 7, 10, 15, 8, 20];

const resultado = numeros.find((n) => n > 12);

console.log(resultado); // 15
```

ARRAYS - map() (métodos)

- Aplica uma função a cada elemento do array e devolve um novo array transformado..
- Não altera o array original.
- É ideal para alterar ou formatar dados.
- Quando usar map
  - Transformação 1 → 1
  - Mesmo tamanho
  - Sem efeitos colaterais

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


console.log(quadrados); // [1, 4, 9]
```

ARRAYS - reduce() (métodos)

- Reduz o array a um único valor.
  - acc → acumulador
  - n → valor
  - 0 → valor inicial
- Usos de reduce
  - soma
  - produto
  - máximo / mínimo
  - contagem
  - Agregações

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

const soma = numeros.reduce((acc, n) => acc + n, 0);

console.log(soma); // 10
```

ARRAYS - reduce() (métodos)

- Calcular o maior valor de um array.

```
const numeros = [4, 7, 2, 9];

const max = numeros.reduce((m, n) => (n > m ? n : m));

console.log(max);
```

Objetos - Problema inicial

- Arrays funcionam bem para listas simples, mas:
    - Não atribuem significado aos dados
    - Dependem da posição
    - Não representam entidades reais

```
const aluno = ["Ana", 12345, 14];
```

Objetos - Introdução aos Objetos

- Agrupam dados relacionados
- Cada valor tem um significado (chave)
- Melhor legibilidade e manutenção

```
const aluno = {
  nome: "Ana",
  numero: 12345,
  nota: 14,
};
```

```
console.log(aluno.nome);
console.log(aluno.numero);
console.log(aluno.nota);
console.log(aluno);
```

Objetos - Exercício 1

- Criar um objeto livro com:
    - título
    - autor
    - ano
- Mostrar os dados na consola.

Objetos - Métodos em Objetos

- Funções associadas a um objeto
- Operam sobre os seus próprios dados
- Uso de this

```
const aluno = {
  nome: "Ana",
  nota: 14,
  aprovado() {
    return this.nota >= 10;
  },
};
```

Objetos - Exercício 2

- Criar um objeto contaBancaria com:
    - titular
    - saldo
- Método levantar(valor) que só permite levantar se houver saldo suficiente.

Objetos - Coleções de Entidades

- Muito comuns em aplicações reais
- Permitem listagens, pesquisas e filtros

```
const alunos = [
  { nome: "Ana", nota: 14 },
  { nome: "Bruno", nota: 9 },
];
```

```
for (const aluno of alunos) {
  if (aluno.nota >= 10) {
    console.log(aluno.nome, "aprovado");
  }
}
```

Objetos - Percorrer e Filtrar

```
for (const aluno of alunos) {
  if (aluno.nota >= 10) {
    console.log(aluno.nome, "aprovado");
  }
}
```

Objetos - Exercício 3

- Criar um array de objetos produtos e listar apenas os produtos com preço superior a 10€.

Classes - Porque Usar Classes?

- Criar moldes reutilizáveis
- Evitar duplicação
- Código mais organizado

- Instância de objetos

```
const a1 = new Aluno("Ana", 14);
const a2 = new Aluno("Bruno", 9);

console.log(a1.aprovado());
```

- Definição de class

```
class Aluno {
  constructor(nome, nota) {
    this.nome = nome;
    this.nota = nota;
  }

  aprovado() {
    return this.nota >= 10;
  }
}
```

Classes - Exercício 4

- Criar uma classe Produto com:
    - nome
    - preco
    - stock
- Método temStock().

Classes - Combinar Estruturas

```
const turma = [];
turma.push(new Aluno("Ana", 14));
turma.push(new Aluno("Bruno", 9));
```

Classes - Exercício 5

- Adicionar vários produtos a um array e listar apenas os que têm stock disponível.

Coleções - Estrutura Map()

- Chaves únicas
- Qualquer tipo de chave
- Acesso rápido por identificador

```
const notas = new Map();
notas.set(123, 14);
notas.set(456, 9);

console.log(notas.get(123));
```

Coleções - Exercício 6

- Usar um Map para associar:
    - código de produto → preço

Coleções - Estrutura Set()

- Não permite duplicados
- Útil para validação e controlo

```
const tecnologias = new Set();
```

```
const numeros = new Set([1, 2, 2, 3, 3]);
console.log(numeros.size);
```

Coleções - Exercício 7

- Criar um Set com os nomes dos alunos presentes numa aula, evitando repetições.

Coleções - Pilha (Stack)

- Criar um Set com os nomes dos alunos presentes numa aula, evitando repetições.

```
const stack = [];
stack.push(1);
stack.push(2);
stack.pop();
```

```
const historico = [];
historico.push("A");
historico.push("B");
historico.pop();
```

Coleções - Exercício 8

- Simular uma fila de atendimento usando um array (push e shift).

Algoritmia e Programação

Estruturas de Dados
Arrays, Objetos e Coleções

Marta Martinho 2025