Algoritmia e Programação

Funções

Marta Martinho 2025

Exercício 0 — “Simulador de avaliações”

- Pretende-se um programa que:
  - peça ao utilizador 3 notas (0–20);
  - valide cada nota;
  - calcule a média;
  - indique se o aluno está aprovado ou reprovado.

```
let nota;
let soma = 0;

do {
  nota = Number(prompt("NUMERO [0-20]"));
} while (nota < 0 || nota > 20);

soma += nota;

do {
  nota = Number(prompt("NUMERO [0-20]"));
} while (nota < 0 || nota > 20);

soma += nota;

do {
  nota = Number(prompt("NUMERO [0-20]"));
} while (nota < 0 || nota > 20);

soma += nota;

let media = soma / 3;

if (media >= 10) {
  console.log("Aprovado");
}
else {
  console.log("Reprovado");
}
```

Exercício 0 — “Simulador de avaliações”

- código longo,
- repetição de validações,
- decisões misturadas.

- Onde está a validação?
- Onde está o cálculo?
- Se mudasse a regra de aprovação, onde mexíamos?

Organização da apresentação

- Motivação: porque usamos funções
- Conceito de função em JavaScript
- Tipos de funções
- Parâmetros, argumentos e valores de retorno
- Escopo e return
- Funções como valores (callbacks e funções de ordem superior)
- Funções recursivas
- Funções assíncronas – introdução
- Exercícios

Vantagens no uso de funções

- Reutilização de código
- Organização: dividir um problema em partes mais pequenas
- Redução de código duplicado
- Maior legibilidade e manutenção
- Facilita trabalho em equipa
- Permite esconder detalhes de implementação (abstração)

```
function saudacao() {
  console.log("Olá!");
}

saudacao();
saudacao();
```

Conceito básico de função em JavaScript

O que é uma função?

- Bloco de código que tem um nome (opcional, se for anónima)
- É definida uma vez e pode ser executada várias vezes
- Pode receber dados (parâmetros)
- Pode devolver um valor com **return**

Estrutura básica (declaração)

- **function** – palavra reservada
- n**omeDaFuncao** – identificador
- (**param1, param2)** – lista de parâmetros
- **return** – devolve um valor à instrução de chamada

```
function nomeDaFuncao(param1, param2) {
  // corpo da função
  // instruções
  return resultado; // opcional
}
```

Exercício 1 — Primeira função

- Criar uma função saudacao() que escreve “Olá, mundo!” na consola e chamá-la duas vezes.

```
function saudacao() {
  console.log("Olá mundo!");
}

saudacao();
saudacao();
```

Exercício 2 — Separação de responsabilidades

- Reorganiza o programa anterior criando:
  - lerNumero(min, max, msg)
  - media(n1, n2, n3)
  - resultadoFinal(media
- Nenhuma função pode:
  - ler e decidir ao mesmo tempo,
  - decidir e calcular ao mesmo tempo.

```
const MIN = 0;
const MAX = 20;

function lerNumero(min, max, msg) {
  let numero;
  do {
    numero = Number(prompt(msg));
  } while (nota < min || nota > max);
  return numero;
}

let soma = 0;

nota = lerNumero(MIN, MAX, "NUMERO [0-20]");

soma += nota;

let nota = lerNumero(MIN, MAX, "NUMERO [0-20]") ;

soma += nota;

nota = lerNumero(MIN, MAX, "NUMERO [0-20]");

soma += nota;
```

Exercício 3 — Função que não pode usar prompt

- Implementa a função:

```
function classificacao(media)
```

que devolve:

- "Aprovado" se média ≥ 9.5
- "Reprovado" caso contrário

Regra:

- a função não pode ler dados nem escrever na consola
- apenas recebe valores e devolve um resultado

Tipos de funções em JavaScript

- Function Declaration (declaração clássica)
- Function Expression (função atribuída a uma variável)
- Arrow Function (sintaxe curta ES6+)
- Métodos (funções dentro de objetos)
- Funções anónimas (sem nome explícito)

Function Declaration

- (**hoisted**): pode ser usada antes da declaração

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

const resultado = soma(3, 5);
```

Function Expression

- Função anónima - sem nome - guardada numa variável
- Não é **hoisted** da mesma forma que a declaration

```
const soma = function (a, b) {
  return a + b;
};

const resultado = soma(3, 5);
```

Arrow Functions

- Sintaxe mais compacta

```
const soma = (a, b) => {
  return a + b;
};

// forma ainda mais curta
const soma2 = (a, b) => a + b;
```

Exercício 4 — Escolha da estratégia de cálculo

- Implementa duas versões da média:
    - média simples
    - média ponderada (pesos 0.3, 0.3, 0.4)
- Aqui entram:
    - function expressions
    - passar funções como argumento
    - funções como valores

Métodos (funções em objetos)

- Funções associadas a objetos dizem-se métodos

```
const pessoa = {
  nome: "Ana",
  cumprimentar() {
    console.log("Olá, eu sou a " + this.nome);
  },
};

pessoa.cumprimentar();
```

Parâmetros vs argumentos

- **Parâmetros** – nomes definidos na função
- **Argumentos** – valores passados quando a função é chamada

```
function media(n1, n2, n3) { // parâmetros
  return (n1 + n2 + n3) / 3;
}

const resultado = media(10, 12, 8); // argumentos
```

Parâmetros opcionais e valores por omissão

- **Introduz param = valorPadrao**

```
function saudacao(nome = "visitante") {
  console.log("Olá, " + nome + "!");
}

saudacao("Marta");   // Olá, Marta!
saudacao();          // Olá, visitante!
```

Escopo de função

- Variáveis declaradas com let ou const dentro da função só existem nessa função
- Não são visíveis fora da função
- Evita “poluir” o escopo global

```
let x = 15
function exemplo() {
  let y = 10;
  console.log(y); // 10
}

console.log(x); //15
console.log(y); // erro: x is not defined
```

Exercício 5 — Bug realista

- corrigir sem mudar o objetivo:

- Perguntas:
  - Porque é que isto é perigoso?
  - Onde está o problema de escopo?
  - Como resolver sem variáveis globais?

```
let total = 0;

function adicionar(valor) {
  total += valor;
}

function media(qtd) {
  return total / qtd;
}
```

Funções que retornam valor vs sem retorno

Funções que retornam um valor:

- usadas em expressões
- O resultado da execução é guardado em variáveis

Funções que não retornam (sem return ou return;):

- executam ações (ex.: escrever na consola, alterar o DOM)

```
function soma(a, b) {
  return a + b; // devolve um valor
}

function mostraSoma(a, b) {
  console.log(a + b); // apenas mostra
}
```

Exercício 6 — API mal desenhada

- Problema:
  - não pode ser reutilizada
  - não pode ser testada
  - não pode ser composta com outras funções
- Desafio:
  - redesenhar a função
  - justificar a decisão
-

```
function calculaDesconto(preco) {
  console.log(preco * 0.9);
}
```

Funções como valores (higher-order)

Podem ser:

- guardadas em variáveis
- passadas como argumentos
- devolvidas por outras funções

Funções são “cidadãos de primeira classe”

Callbacks (funções passadas como argumento)

```
function cumprimentar(nome, callback) {
  console.log("Olá, " + nome);
  callback();
}

function despedir() {
  console.log("Até já!");
}

cumprimentar("João", despedir);
```

Funções recursivas

O que é recursividade?

- Função que chama a si própria
- Precisa sempre de:
  - caso base (onde pára)
  - chamada recursiva (que se aproxima do caso base)

```
function contagem(n) {
  if (n === 0) {
    console.log("Terminado!");
    return;
  }

  console.log(n);
  contagem(n - 1);
}

contagem(5);
```

Recursividade

$$fact(n) = \begin{cases} 1 & \text{se} \quad n = 0 \\ n \times fact(n-1) & \text{se} \quad n \geq 1 \end{cases}$$

$$\begin{aligned}
\texttt{fact}(5) &= 5 \times \texttt{fact}(4) \\
&= 5 \times (4 \times \texttt{fact}(3)) \\
&= 5 \times (4 \times (3 \times \texttt{fact}(2))) \\
&= 5 \times (4 \times (3 \times (2 \times \texttt{fact}(1)))) \\
&= 5 \times (4 \times (3 \times (2 \times (1 \times \underline{\texttt{fact}(0)})))) \\
&= 5 \times (4 \times (3 \times (2 \times (1 \times 1)))) \\
&= 5 \times (4 \times (3 \times (2 \times 1))) \\
&= 5 \times (4 \times (3 \times 2)) \\
&= 5 \times (4 \times 6) \\
&= 5 \times 24 \\
&= 120
\end{aligned}$$

Exercício

Potência de um Número

A potência de um valor inteiro pode ser calculado usando uma definição recursiva, baseada no facto de que $a^0 = 1$, $a^1 = 1 \times a$, $a^2 = 1 \times a \times a$,...

$$pot(base, exp) = \begin{cases} 1 & \text{se } exp = 0 \\ base \times pot(base, exp - 1) & \text{se } exp \geq 1 \end{cases}$$

Implemente uma função recursiva em C que calcule a potência de um número inteiro. Implemente também uma versão iterativa equivalente.

Funções síncronas vs assíncronas

- Síncronas:
  - executam passo a passo; cada instrução espera pela anterior
- Assíncronas:
  - permitem “aguardar” operações demoradas (pedidos HTTP, timers) sem bloquear o programa

```
console.log("Antes");

setTimeout(() => {
  console.log("Dentro do timeout");
}, 2000);

console.log("Depois");
```

Exercício 7 — Mini-sistema (30 min)

- Desenvolve um programa que:
  - peça temperaturas até o utilizador escrever -1;
  - valide cada valor;
  - converta todas para Kelvin;
  - calcule a média;
  - indique quantas estavam acima da média.
- Criar pelo menos estas funções:
  - lerTemperatura()
  - celsiusToKelvin(temp)
  - media(total, quantidade)
  - acimaDaMedia(valor, media)

Algoritmia e Programação

Funções

Marta Martinho 2025