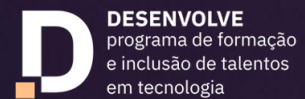



KORU





**FUNÇÕES,
SPREAD E
REST**









O QUE VAMOS APRENDER HOJE?

- Por que usar **Funções**?
- **Sintaxe Básica** e Tipos de Funções
- **Parâmetros** e **Argumentos**
- O Comando **return**
- Entendendo o **Escopo** (Global, Função, Bloco)
- **Parâmetros Padrão**
- **Rest** Parameters (...args)
- **Spread** Syntax (...) para Arrays e Objetos



**POR QUE
FUNÇÕES?**

FUNÇÕES SÃO PEQUENOS PROGRAMAS

```
function makeSandwich(, , ) {  
      
    let sandwich =  +  +  ;  
    return  ;  
}
```


POR QUE FUNÇÕES?

Problema: repetição de código.

Solução: funções

Funções são blocos de código que realizam uma tarefa específica e podem ser chamados (executados) várias vezes.

Benefícios:

Reutilização: Escreva uma vez, use em muitos lugares.

Organização: Quebre problemas grandes em partes menores e gerenciáveis.

Legibilidade: Nomeie seus blocos de código para entender o que fazem.

Manutenção: Corrija um erro ou adicione uma funcionalidade em apenas um lugar.



SINTAXE DAS FUNÇÕES

SINTAXE BÁSICA

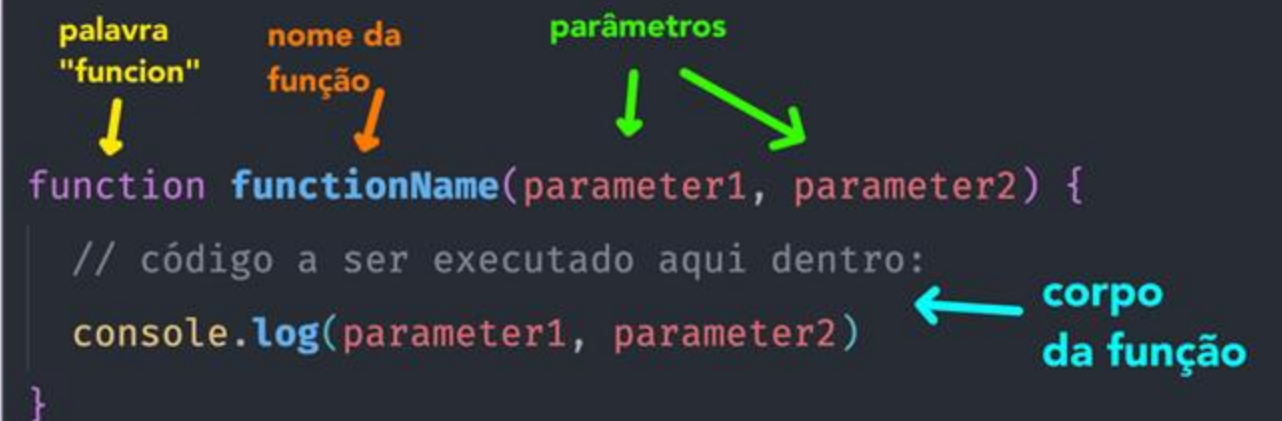
EXISTEM 2 MOMENTOS:

1. A **declaração** da função (onde dizemos como nossa função "funciona").

2. A **chamada** da função (onde queremos executar/rodar a nossa função)

SINTAXE BÁSICA

Declaração de uma função



The diagram illustrates the syntax of a JavaScript function declaration. It shows the following code snippet with labels and arrows:

```
function functionName(parameter1, parameter2) {  
  // código a ser executado aqui dentro:  
  console.log(parameter1, parameter2)  
}
```

Labels and arrows:

- palavra "function"** (yellow text) points to the word `function`.
- nome da função** (orange text) points to the identifier `functionName`.
- parâmetros** (green text) points to the parameters `parameter1` and `parameter2` inside the parentheses.
- corpo da função** (cyan text) points to the code block inside the curly braces, specifically to the `console.log` statement.

SINTAXE BÁSICA

Chamada de uma função

argumentos



```
functionName('lala', 'land');
```

SINTAXE BÁSICA

Parâmetros vs Argumentos

Parâmetros: Variáveis listadas na definição da função. Eles agem como placeholders para os valores que a função espera receber.

Argumentos: Os valores reais que você passa para a função quando a chama.

SINTAXE BÁSICA

O comando ``return``

Nem todas as funções precisam produzir um resultado "para fora", mas muitas precisam.

O comando **return** especifica o valor que uma função deve "**devolver**" para o código que a chamou.

Quando o JavaScript encontra um `return`, a função para de executar imediatamente e retorna o valor especificado.

Se uma função não tem um ``return`` explícito, ou tem um `return;` vazio, ela **retorna** **undefined** por padrão.

SINTAXE BÁSICA

```
function add(a, b) {  
  let sum = a + b;  
  return sum; // Retorna o valor da soma  
}  
  
let result = add(5, 3); // result agora é 8  
result = add(10, 3) // result agora é 13
```


A group of diverse, smiling people, including a man with a beard and glasses, a woman with glasses, and a man with a beard and glasses, are shown in a close-up shot. The image is overlaid with a semi-transparent purple filter. The text "ARROW FUNCTIONS" is written in large, bold, white capital letters on the right side of the image.

ARROW FUNCTIONS

ARROW FUNCTIONS

Sintaxe mais curta
(introduzida
posteriormente), popular e
moderna

```
const cube = (number) => {  
  return number * number * number;  
};
```



PRÁTICA 01. CRIANDO FUNÇÕES

EXERCÍCIOS

- Crie uma função chamada **welcomeMessage** que não recebe parâmetros e imprime "Bem-vindo ao curso!" no console. Chame esta função.
- Crie uma função chamada **calculateArea** que recebe dois parâmetros: width e height. Ela deve calcular a área (largura * altura) e retornar o resultado. Armazene o resultado em uma variável e imprima no console.
- Crie uma função **convertToCelsius** que recebe uma temperatura em Fahrenheit e retorna o valor equivalente em Celsius usando a fórmula: $(F - 32) * 5 / 9$. Teste com alguns valores.
- Crie as funções **calculateArea** e **convertToCelsius** usando a sintaxe de Function Expression e Arrow Function.

A group of diverse young people, including men and women of various ethnicities, are smiling and waving. The image is overlaid with a purple gradient. The word "ESCOPO" is written in white, bold, sans-serif capital letters on the right side.

ESCOPO

ESCOPO

- **Escopo:** Refere-se à acessibilidade de variáveis, funções e objetos em alguma parte do seu código. Basicamente, onde você pode usar um nome (variável, função) que você definiu.
- **Por que é importante?** Ajuda a evitar conflitos de nomes e entender onde e como as variáveis estarão disponíveis no código

ESCOPO

Escopo Global

Variáveis declaradas fora de qualquer função ou bloco.

Podem ser acessadas de qualquer lugar no seu código (dentro de funções, blocos, etc.).

Exemplo: `let`, `const` declarados no nível superior de um script

```
let globalVariable = 'Eu sou uma variável global!';

function accessGlobal() {
  console.log(globalVariable); // Posso acessar aqui!
}

accessGlobal();
console.log(globalVariable); // E aqui também!
```

ESCOPO

Escopo de Função

Variáveis declaradas dentro de uma função (usando *let* ou *const*).

Só podem ser acessadas de dentro da função onde foram declaradas.

Variáveis de uma função não são acessíveis fora dela.

```
function functionScopeExample() {  
  let functionVariable = "Eu vivo só dentro desta função";  
  console.log(functionVariable);  
}  
  
functionScopeExample();  
console.log(functionVariable); // Erro! functionVariable  
não está definida aqui.
```

ESCOPO

Escopo de Bloco

Variáveis declaradas **dentro de um bloco {}** (como dentro de if, for, while, ou simplesmente um par de chaves) usando let ou const. Só podem ser acessadas de dentro do bloco onde foram declaradas.

```
if (true) {  
  let blockVariableLet = "Eu sou de bloco com let";  
  const blockVariableConst = "Eu sou de bloco com const";  
  console.log(blockVariableLet);  
  console.log(blockVariableConst);  
}  
  
console.log(blockVariableLet); // Erro!  
console.log(blockVariableConst); // Erro!
```



PARÂMETROS PADRÃO (DEFAULT PARAMETERS)

DEFAULT PARAMETERS

Default Parameters

Permitem que você defina **valores padrão** para os parâmetros de uma função caso nenhum argumento (ou undefined) seja passado para eles na chamada da função.

Torna as funções mais flexíveis e evita erros quando argumentos opcionais não são fornecidos.

```
function functionName(parameter1 = 'lala', parameter2 = 'land') {  
  console.log(parameter1 + parameter2) // 'lalaland'  
}
```

DEFAULT PARAMETERS

```
function greetUser(name = 'Usuário Anônimo') {  
  console.log('Hello, ' + name + '!');  
}
```

```
greetUser('Maria'); // Hello, Maria!
```

```
greetUser(); // Hello, Usuário Anônimo!
```

```
greetUser(undefined); // Hello, Usuário Anônimo!
```

```
greetUser(null); // Hello, null! (null é um valor, não undefined)
```



REST PARAMETERS

REST PARAMETERS

Permitem que uma função aceite um número indefinido de argumentos como um array.

Útil quando você não sabe quantos argumentos serão passados para a função, mas quer lidar com eles como uma coleção.

São indicados por três pontos (...) seguidos pelo nome do parâmetro (geralmente args ou algo descritivo).

Importante: Os Rest Parameters devem ser o último parâmetro na lista de parâmetros da função.

```
function functionName(param1, param2, ...restOfArgs) {  
  // restOfArgs será um array contendo o 3º argumento em diante  
}
```




PRÁTICA 02. **REST PARAMETERS**

REST PARAMETERS

- Crie uma função **sumAll**.
- Esta função poderá receber quantos argumentos forem necessários. E ela deverá retornar o valor da soma de todos os argumentos.



SPREAD SYNTAX

SPREAD SYNTAX

Indicado por **três pontos** (...), mas usado em um local diferente dos Rest Parameters.

Enquanto Rest coleta elementos em um array, **Spread expande (espalha) elementos de um iterável (como um array)** ou propriedades de um objeto.

Útil para criar cópias, combinar arrays/objetos, ou passar elementos de um array como argumentos individuais para uma função.

SPREAD SYNTAX

```
const newObj = {...object} // Espalha propriedades de um objeto em outro objeto  
const newArr = [...array] // Espalha elementos de um array em outro array
```

SPREAD SYNTAX COM ARRAYS

Copiar Arrays: Cria uma cópia rasa (shallow copy) de um array.

```
const originalArray = [1, 2, 3];  
const copiedArray = [...originalArray]; // copiedArray é [1, 2, 3] mas é um novo array  
console.log(copiedArray);  
console.log(originalArray === copiedArray); // false
```

SPREAD SYNTAX COM ARRAYS

Combinar/Concatenar Arrays: Uma alternativa concisa ao método `concat()`.

```
const arr1 = [1, 2];
const arr2 = [3, 4];
const combinedArray = [...arr1, ...arr2]; // combinedArray é [1, 2, 3, 4]
const anotherCombination = [0, ...arr1, 5, ...arr2, 6]; // [0, 1, 2, 5, 3, 4, 6]
console.log(combinedArray);
console.log(anotherCombination);
```


SPREAD SYNTAX COM OBJETOS

Copiar Objetos: Cria uma cópia rasa (shallow copy) de um objeto.

```
const originalObject = { a: 1, b: 2 };  
const copiedObject = { ...originalObject }; // copiedObject é { a: 1, b: 2 } mas é um novo objeto  
console.log(copiedObject);  
console.log(originalObject === copiedObject); // false
```

SPREAD SYNTAX COM OBJETOS

Combinar/Concatenar Objetos:

```
const obj1 = { a: 1, b: 2 };  
const obj2 = { c: 3, d: 4 };  
const combinedObject = { ...obj1, ...obj2 }; // { a: 1, b: 2, c: 3, d: 4 }  
console.log(combinedObject);  
  
const objWithOverride = { ...obj1, b: 20, e: 5 }; // { a: 1, b: 20, e: 5 }  
console.log(objWithOverride);
```



SPREAD VS REST

SPREAD VS REST

Mesma sintaxe (...), mas papéis opostos!

Rest: Usado na definição da função, no último parâmetro. Coleta argumentos em um array.

Spread: Usado ao chamar a função ou ao criar literais de array/objeto. Expande elementos/propriedades.



PRÁTICA 03

PRÁTICA

- Crie uma função chamada findMax que recebe um número variável de argumentos (números) usando Rest Parameters e retorna o maior número entre eles. Use um loop (for...of) para iterar sobre o array de argumentos. Teste com findMax(10, 5, 20, 8) e findMax(2, 7).
- Crie dois arrays: frontendSkills = ['HTML', 'CSS', 'JavaScript'] e backendSkills = ['Node.js', 'Databases', 'APIs']. Use Spread Syntax para criar um novo array fullstackSkills que combine frontendSkills e backendSkills. Imprima fullstackSkills.
- Crie um objeto person = { name: 'Ana', age: 25 } e um objeto address = { city: 'Recife', country: 'Brazil' }. Use Spread Syntax para criar um novo objeto personWithAddress que combine as propriedades de person e address. Imprima personWithAddress.
- Crie um array de números. Use a função findMax que você criou e a Spread Syntax para encontrar o maior número no array.

A group of diverse young people, including men and women of various ethnicities, are smiling and waving at the camera. The image is overlaid with a semi-transparent purple gradient. The word "RESUMO" is written in large, bold, white capital letters on the right side of the image.

RESUMO

RESUMO

Funções: Essenciais para reutilização, organização e legibilidade do código.

Declaramos funções usando function ou Arrow Functions (=>).

return permite que funções devolvam um valor.

Escopo: Define onde as variáveis são acessíveis (Global, Função, Bloco). let/const introduziram o escopo de bloco.

Parâmetros Padrão: Dão valores default aos parâmetros, tornando funções mais flexíveis.

Rest Parameters (... na definição): Coletam múltiplos argumentos em um array.

Spread Syntax (... na chamada/literal): Expandem elementos de arrays ou propriedades de objetos



Vamos
avaliar o
encontro?

KORU

