

# Funções

# O que vamos ver hoje?

- O que são funções
- Declarando funções
- Parâmetros e argumentos
- Outros tipos de função
- Boas práticas

# Contextualização

# Calcular área de **um** retângulo



- Para calcular a área de um retângulo, temos a seguinte equação:

$$\text{area} = \text{altura} \times \text{largura}$$

- Se vamos escrever um código para calcular a área de um retângulo para a gente, podemos fazer algo assim:

```
1 const altura = 2
2 const largura = 3
3 const area = altura * largura
4 console.log(area)
```

# Calcular área de **dois** retângulos #

- Para calcular a área de dois retângulos, basta repetir a mesma lógica para ambos

```
1 // Retângulo 1
2 const altura1 = 2
3 const largura1 = 3
4 const area1 = altura1 * largura1
5 console.log(area1)
6
7 // Retângulo 2
8 const altura2 = 5
9 const largura2 = 2
10 const area2 = altura2 * largura2
11 console.log(area2)
```

# Calcular área de seis retângulos 🤯

- Para calcular a área de seis retângulos, basta repetir a mesma lógica para todos?
- Perceba quantas linhas de código!  
35 linhas com muitas repetições

```
1 // Retângulo 1
2 const altura1 = 2
3 const largura1 = 3
4 const area1 = altura1 * largura1
5 console.log(area1)
6
7 // Retângulo 2
8 const altura2 = 5
9 const largura2 = 2
10 const area2 = altura2 * largura2
11 console.log(area2)
12
13 // Retângulo 3
14 const altura3 = 1
15 const largura3 = 1
16 const area3 = altura3 * largura3
17 console.log(area3)
18
19 // Retângulo 4
20 const altura4 = 7
21 const largura4 = 8
22 const area4 = altura4 * largura4
23 console.log(area4)
24
25 // Retângulo 5
26 const altura5 = 3
27 const largura5 = 1
28 const area5 = altura5 * largura5
29 console.log(area5)
30
31 // Retângulo 6
32 const altura6 = 2
33 const largura6 = 7
34 const area6 = altura6 * largura6
35 console.log(area6)
```

# Problemas

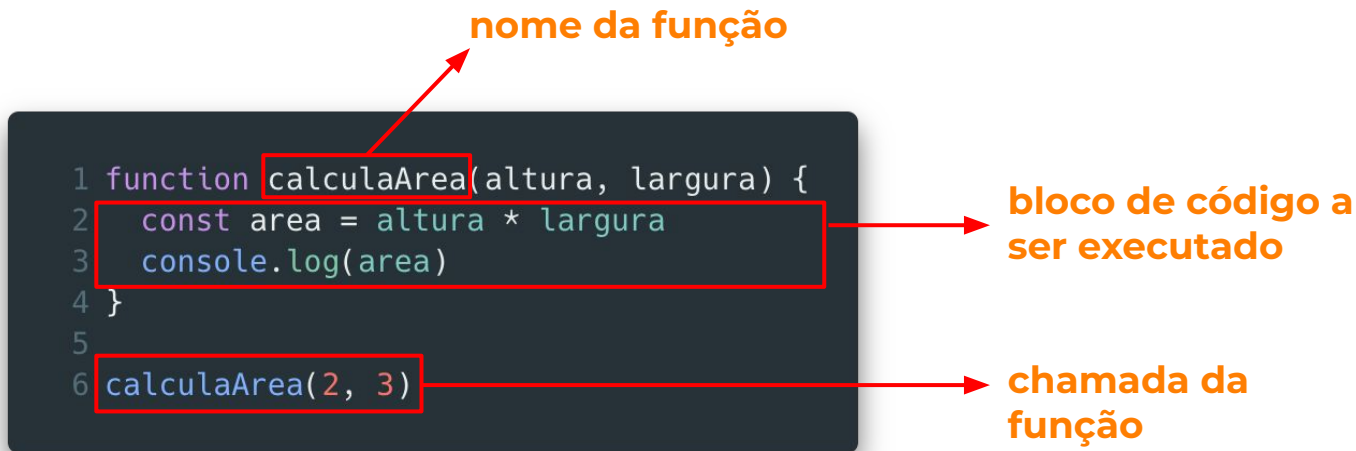
- Copiar e colar código é chato
- Código fica muito comprido e difícil de ler
- Nomes de variáveis não podem repetir
- Se precisarmos mudar a lógica, teremos que mudar em todos os lugares do código
- **Solução: Funções!**

# O que são funções?



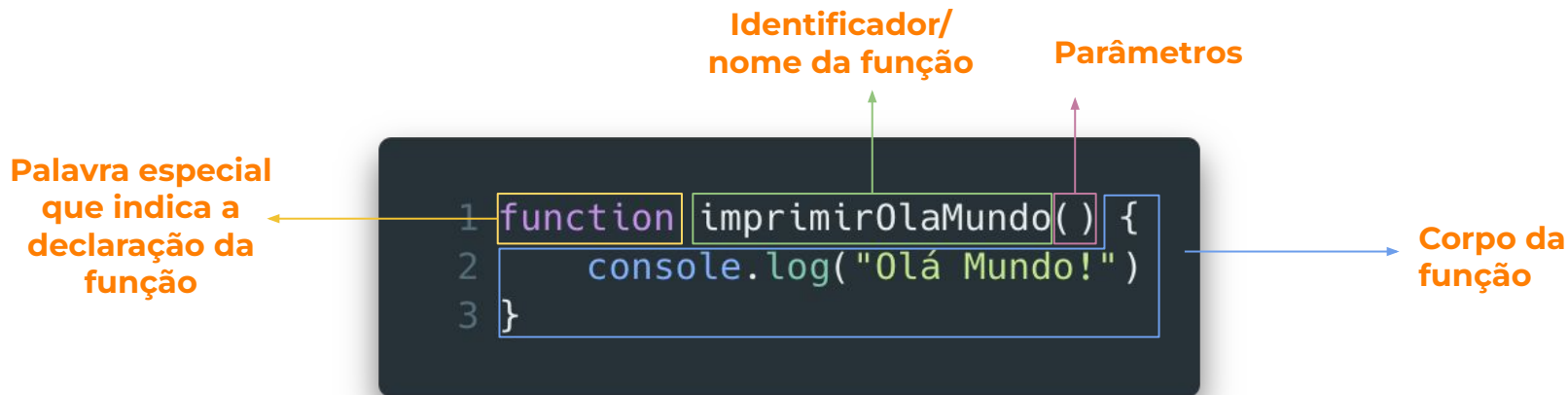
# O que é uma função?

- Uma função é um bloco de código que pode ser chamado (ou invocado) a partir do seu nome. Permite reutilizar variáveis.



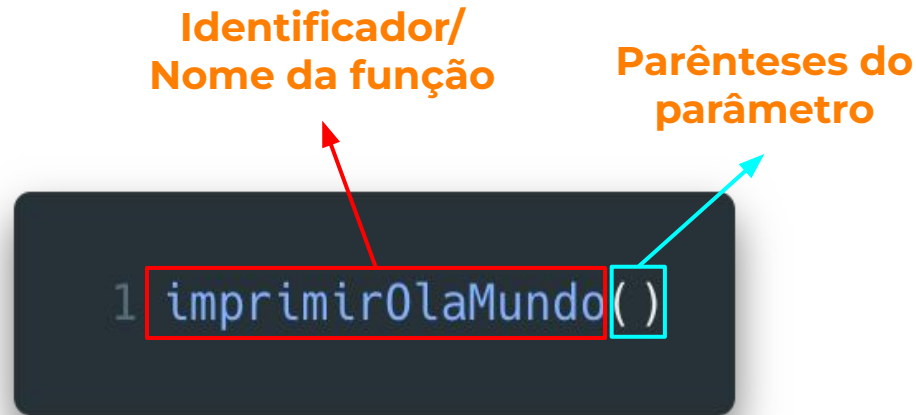
# Declarando uma função

- O primeiro passo para criar uma função é **declará-la**
- A declaração **atribui** um **bloco de código** à um **identificador** (ou um nome)



# Chamando uma função

- Podemos chamar, invocar ou executar uma função usando o seu identificador. Quando fazemos isso, o bloco de código definido na declaração é executado.



# Declaração **vs.** Execução 💡

- Só declarar a função **não executa** o código
- Você pode **chamar/invocar** e **executar** a função quantas vezes quiser
- O JavaScript permite executar a função **antes** da sua declaração. Porém, isso deixa o código confuso
- Priorize declarar a função primeiro, e posteriormente executa-lá

# Exemplo

## Declaração

```
1 function imprimirOlaMundo() {  
2     console.log("Olá Mundo!")  
3 }
```

## Execução

```
1 imprimirOlaMundo()
```

# Exercício 1

- Crie uma **função** que **imprima no console** a frase “Olá Mundo!”

# Parâmetros e Argumentos



Funções podem receber **entradas**, e se receberem, devem ser usadas no bloco do código dentro da função

```
1 function calculaArea(altura, largura) {  
2   const area = altura * largura  
3   console.log(area)  
4 }  
5  
6 calculaArea(2, 3)
```

parâmetros

parâmetros sendo utilizados  
dentro do bloco de código

argumentos

# Parâmetros e Argumentos

- **Parâmetros** são como **variáveis** criadas na declaração da função, onde podemos guardar os argumentos (valores) a serem enviados para a função
- **Argumentos** são os **valores** (strings, numbers, booleanos) passados na chamada da função. Cada parâmetro recebe seu valor dos argumentos, seguindo a mesma ordem



## Exercício 2

- Crie uma função que **receba por parâmetro um nome** e imprima no console a mensagem:  
`Olá \${nome}`
- **Invoque** esta função passando **3 argumentos** (nomes) diferentes

# Fixação

- Uma função é um **bloco de código** que é executado a partir da sua invocação
- Podem receber **entradas**, que devem ser usadas no meio do código (parâmetros e argumentos)

Programa

**3000 TALENTOS TI**

# Escopo

# Escopo { }

O escopo determina quais variáveis serão acessíveis ao rodarmos o código.

# Escopo { }

- No Javascript temos dois tipos de escopo:
  - **Escopo Global:** variáveis no escopo global podem ser acessadas de qualquer lugar do código.
  - **Escopo Local:** variáveis no escopo local somente podem ser acessadas dentro do escopo em que foram declaradas.
- As variáveis definidas dentro de uma **função** possuem **escopo local**

# Escopo {}

## escopo global

**pai** de todos os escopos (compartilha suas variáveis com todos)

```
function funcao1() {
```

## escopo local #1

**pai** do escopo local #2 (compartilha suas variáveis com o **filho**)

```
function funcao2() {
```

## escopo local #2

**filho** do escopo local #1

```
}
```

```
}
```

# Escopo {}

Global

```
const a = 1
```

Declaração da variável **a** no **escopo global**

Local

```
function imprimeVariavel () {  
  const b = 2  
  console.log('Variável a', a)  
  console.log('Variável b', b)  
}
```

Declaração da variável **b** no **escopo local**

```
imprimeVariavel()
```

```
console.log('Variável a', a)  
console.log('Variável b', b)
```

# Escopo {}

Global

```
const a = 1
```

```
function imprimeVariavel () {
```

Local

```
  const b = 2
```

```
  console.log('Variável a', a)
```

```
  console.log('Variável b', b)
```

```
}
```

```
imprimeVariavel()
```

```
console.log('Variável a', a)
```

```
console.log('Variável b', b)
```

Acessando variáveis **a** e **b**  
dentro do escopo local  
É possível acessar ambas  
variáveis

Acessando variáveis **a** e **b**  
dentro do escopo global  
Não é possível acessar  
variável **b**



Como **utilizar** o resultado da  
função **sem usar** o  
**console.log()**?

# Retorno

# Retorno

Funções podem gerar **saídas**, que podem ser acessadas após a execução

```
1 function calculaArea(altura, largura) {  
2     const area = altura * largura  
3     return area  
4 }  
5  
6 // Atribui retorno à uma variável  
7 const areaCalculada = calculaArea(2, 3)  
8  
9 // Imprime retorno no console  
10 console.log(calculaArea(2, 3))
```

retorno da  
função

chamadas

# Retorno

- O retorno acontece usando a palavra chave **return**, seguida pela variável/valor a ser retornado
- Uma função só pode retornar **um valor**
- Quando a função retorna algo, sua **execução é interrompida**
  - Ou seja, o código escrito após o **return** não é executado

# Imprimir **vs.** Retornar

- Quando pede-se para imprimir algo, utilizamos o **console.log()**
- Quando pede-se para retornar algo, utilizamos o **return**

## Exercício 3

- Crie uma **função** que **receba** dois números e **retorne** a soma entre eles
- Guarde o **retorno** dessa função em uma variável e imprima no console

# Funções - modelo mental 🤔

- Funciona como uma caixa preta que pode receber **valores de entrada** (input/parâmetros/argumentos) e pode devolver **valores de saída** (output/resultado)



# Resumindo

Entrada  
(Input)



Saída  
(Output)



# Exercício 4

Crie uma **função** que:

- **Receba** um array de números;
- **Retorne** um **novo array** com dois elementos:
  - o **último** e o **primeiro** número do array recebido divididos por dois.

# Fixação

- As variáveis definidas dentro de uma **função** possuem **escopo local**
- As funções podem **retornar** valores usando **return**

## Atenção

**Todos os conceitos importantes sobre funções já foram passados.**

Daqui para frente, veremos outras **sintaxes** e algumas **terminologias**, apenas a nível de informação e conhecimento.

# Expressões de funções

# Expressões de funções

- Expressões de funções são uma forma **diferente** (mas bem parecida) de se declarar funções
- Deve ser atribuída a uma **variável** e é **invocada** da mesma forma que a declaração, mas usando o nome da variável atribuída

```
1 const calculaArea = function(altura, largura) {  
2   const area = altura * largura  
3   return area  
4 }  
5  
6 const areaCalculada = calculaArea(2, 3)
```

# Arrow Functions

# Arrow Functions

- Tipo de **expressão de função** com **sintaxe simplificada**
- Por ser uma expressão, deve ser atribuída a uma variável para ser invocada
- Invocação continua a mesma

```
1 const calculaArea = (altura, largura) => {  
2   const area = altura * largura  
3   return area  
4 }  
5  
6 const areaCalculada = calculaArea(2, 3)
```

# Comparação



# Comparação 🙌

## Declaração de função

```
1 function somaNumeros (num1, num2) {  
2     return num1 + num2  
3 }
```

## Expressões de função

```
1 let somaNumeros = function(num1, num2) {  
2     return num1 + num2  
3 }
```

```
1 let somaNumeros = (num1, num2) => {  
2     return num1 + num2  
3 }
```

# Comparação 🙌

- A **expressão** de função só pode ser invocada **depois da sua declaração** (const, let)
- A **declaração** de função pode ser chamada de **qualquer parte do código**, mesmo antes de sua declaração efetiva (function) 📍
- **Mas evite usar coisas fora da ordem!** O código fica bem mais confuso

# Exercício 5

- Refaça o exercício 2 com a sintaxe de expressão de função
- Refaça o exercício 3 com a sintaxe de Arrow Function

# Extra: funções anônimas

- Uma outra terminologia que existe é a de “funções anônimas” ou “funções não-nomeadas”
- É uma outra forma de denominar expressões de funções, por elas não terem um nome diretamente associado à função
- O nome é o da variável, não o da função em si
- Não tem implicações práticas

# Boas Práticas

# Boas práticas 👍

- Assim como nas variáveis, as funções devem ter **nomes significativos**.
  - Verbos no infinitivo
  - camelCase
- Cada função deve, idealmente, realizar **uma única tarefa**.
- Se sua função tiver muitas responsabilidades, você deve fazer uma função para cada uma dessas

# Resumo

# Resumo

Uma função é um **bloco de código** que pode ser **chamado (ou invocado)** a partir de um nome

```
1 function calculaArea(altura, largura) {  
2   const area = altura * largura  
3   console.log(area)  
4 }  
5  
6 calculaArea(2, 3)
```

bloco de código

chamada  
(invocação)



# Resumo

Funções podem receber **entradas**, que podem ser usadas no meio do código

```
1 function calculaArea(altura, largura) {  
2   const area = altura * largura  
3   console.log(area)  
4 }  
5  
6 calculaArea(2, 3)
```

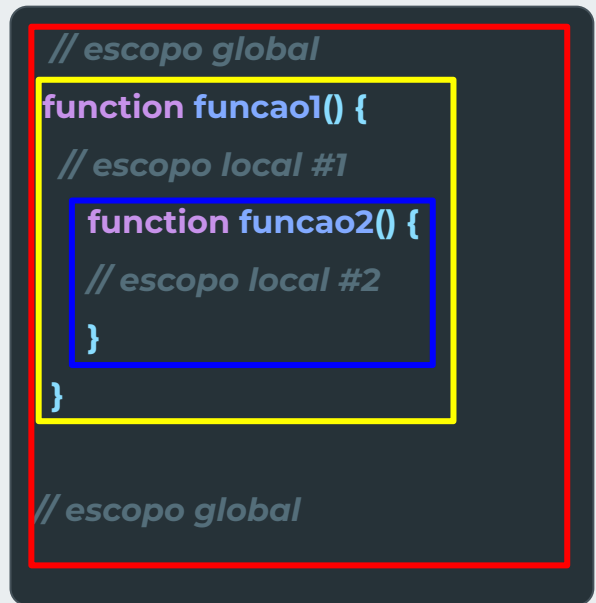
→ parâmetros

→ argumentos

# Resumo



As variáveis criadas dentro das funções possuem **escopo local**, ou seja, só podem ser acessadas de dentro destas.



Escopo global - **pai** de todos os escopos (compartilha suas variáveis com todos)



Escopo local #1 - **pai** do escopo local #2 (compartilha suas variáveis com o **filho**)



Escopo local #2

# Resumo

Funções podem gerar **saídas**, que podem ser acessadas após a execução

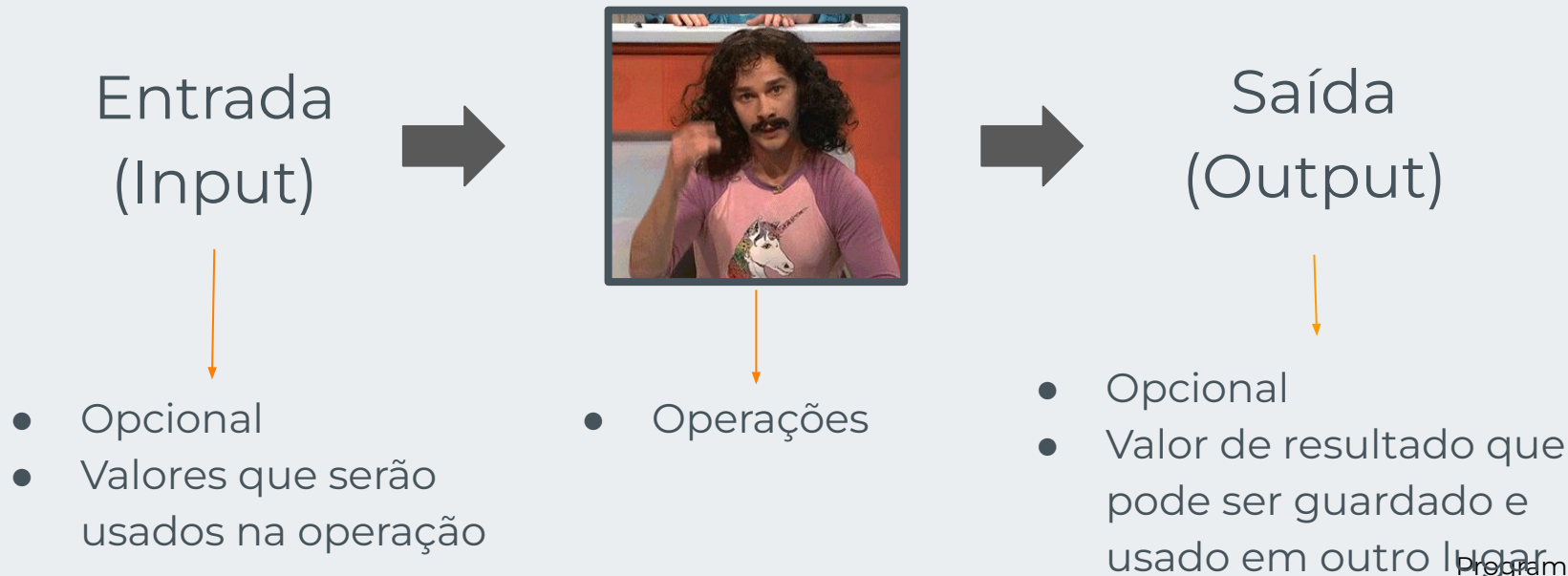
```
1 function calculaArea(altura, largura) {  
2     const area = altura * largura  
3     return area  
4 }  
5  
6 // Atribui retorno à uma variável  
7 const areaCalculada = calculaArea(2, 3)  
8  
9 // Imprime retorno no console  
10 console.log(calculaArea(2, 3))
```

retorno da função

chamadas

# Resumo

- Funções são estruturas que permitem isolar uma parte do nosso código e reaproveitá-lo depois



# Resumo

Em Javascript, existem algumas formas de declarar funções

## Declaração de função

```
1 function somaNumeros (num1, num2) {  
2   return num1 + num2  
3 }
```

## Expressões de função

```
1 let somaNumeros = function(num1, num2) {  
2   return num1 + num2  
3 }
```

```
1 let somaNumeros = (num1, num2) => {  
2   return num1 + num2  
3 }
```

# Dúvidas?



Programa  
**3000 TALENTOS TI**  
Obrigado(a)!