Funções

Labenu_

O que vamos ver hoje?

- O que são funções
- Como e por quê usá-las
- Formas de declarar
- Boas práticas

O que são funções

Labenu_

O que são funções?

- Estruturas que representam uma **ação** no código
- Usadas para dividir algoritmos grandes e complexos em blocos menores e mais simples
- Facilitam a manutenção e o reaproveitamento do código
- Assim como os próprios programas, podem possuir *entradas* (argumentos) e *saídas* (retornos)

Exemplo

```
const pergunta = "Quem é você?"

const resposta = prompt(pergunta)
```

nome da função (verbo)

retornos podem ser armazenados em variáveis

argumentos são passados entre parênteses

Mais Exemplos

```
const texto = "      Quem é você?    "

const textoMinusculo = texto.toLowerCase()

console.log(texto)

alert(textoMinusculo)

console.log(texto.trim())
```

Modelo Mental

Input

Output

Declarando funções

Labenu_

Declarando funções 💭

Além de todas as funções nativas que podemos utilizar, o javascript também nos permite criar novas funções!

É muito comum lançarmos mão desse recurso quando nosso código começa a ficar repetitivo.

Imagine, por exemplo, que você queira gerar logins a partir de uma lista de nomes de usuários. Os logins não podem conter letras maiúsculas ou espaços.

Declarando funções 

O código ao lado ilustra uma forma pouco prática de resolver o problema.

Em uma base de dados com milhares (ou até milhões) de nomes, ele seria completamente inviável

```
let nome1 = "Bill Gates  "
let nome2 = " Jeff Bezos"
let nome3 = "Elon Musk"

nome1 = nome1.toLowerCase()
nome1 = nome1.trim()
nome1 = nome1.replaceAll(" ", "-")

nome2 = nome2.toLowerCase()
nome2 = nome2.trim()
nome2 = nome2.replaceAll(" ", "-")

nome3 = nome3.toLowerCase()
nome3 = nome3.trim()
nome3 = nome1.replaceAll(" ", "-")
```

Declarando funções 

Além do mais, se quiséssemos alterar o algoritmo de formatação dos nomes (para remover acentos, por exemplo), teríamos que copiar e colar essas alterações para cada bloco destacado, com grandes chances de errar em algum ponto (como na última linha)

```
let nome1 = "Bill Gates  "
let nome2 = " Jeff Bezos"
let nome3 = "Elon Musk"

nome1 = nome1.toLowerCase()
nome1 = nome1.trim()
nome1 = nome1.replaceAll(" ", "-")

nome2 = nome2.toLowerCase()
nome2 = nome2.trim()
nome2 = nome2.replaceAll(" ", "-")

nome3 = nome3.toLowerCase()
nome3 = nome3.trim()
nome3 = nome1.replaceAll(" ", "-")
```

Declarando funções 💭

Caso tivéssemos uma função para ***formatar*** esses nomes, nosso código ficaria livre de toda a repetição.

Vamos, então, criá-la juntos!

```
let nome1 = formatar("Bill Gates ")
let nome2 = formatar(" Jeff Bezos")
let nome3 = formatar("Elon Musk")
```

Funções Nomeadas

Labenu_

Funções Nomeadas 💭

A sintaxe mais direta para criar funções é a **declaração nomeada**, que possui os seguintes elementos:

```
function formatar(nome) {
  nome = nome.toLowerCase()
  nome = nome.trim()
  nome = nome.replaceAll(" ", "-")

  return nome
}
```

Funções Nomeadas 💭

1. Palavra chave ***function***, indicando que estamos criando uma função;

```
function formatar(nome) {
  nome = nome.toLowerCase()
  nome = nome.trim()
  nome = nome.replaceAll(" ", "-")

  return nome
}
```

Funções Nomeadas 💭

2. O ***nome*** escolhido (de preferência, um verbo indicando o que ela faz)

```
function formatar(nome) {
  nome = nome.toLowerCase()
  nome = nome.trim()
  nome = nome.replaceAll(" ", "-")

  return nome
}
```

Funções Nomeadas 💭

3. Par de parênteses contendo os ***parâmetros*** dessa função, que são variáveis representando suas **entradas**. Caso não haja parâmetros, os parênteses ficam vazios.

```
function formatar(nome) {
  nome = nome.toLowerCase()
  nome = nome.trim()
  nome = nome.replaceAll(" ", "-")

  return nome
}
```

Funções Nomeadas 💭

4. Par de chaves contendo a lógica e a ***saída*** dessa função, indicada pela palavra chave ***return***. Na ausência dessa palavra, o retorno será *undefined*.

```
function formatar(nome) {
  nome = nome.toLowerCase()
  nome = nome.trim()
  nome = nome.replaceAll(" ", "-")

  return nome
}
```

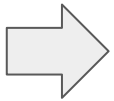
Reescrevendo o código:

```
let nome1 = "Bill Gates "
let nome2 = " Jeff Bezos"
let nome3 = "Elon Musk"

nome1 = nome1.toLowerCase()
nome1 = nome1.trim()
nome1 = nome1.replaceAll(" ", "-")

nome2 = nome2.toLowerCase()
nome2 = nome2.trim()
nome2 = nome2.replaceAll(" ", "-")

nome3 = nome3.toLowerCase()
nome3 = nome3.trim()
nome3 = nome1.replaceAll(" ", "-")
```



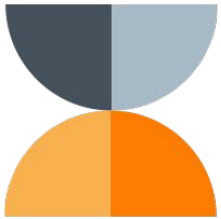
```
function formatar(nome) {
  nome = nome.toLowerCase()
  nome = nome.trim()
  nome = nome.replaceAll(" ", "-")

  return nome
}

let nome1 = formatar("Bill Gates ")
let nome2 = formatar(" Jeff Bezos")
let nome3 = formatar("Elon Musck")
```

Exercício 1

- Crie uma função que receba um nome e imprima no console a frase `Olá, [nome]!`. Invoque-a com 3 nomes diferentes



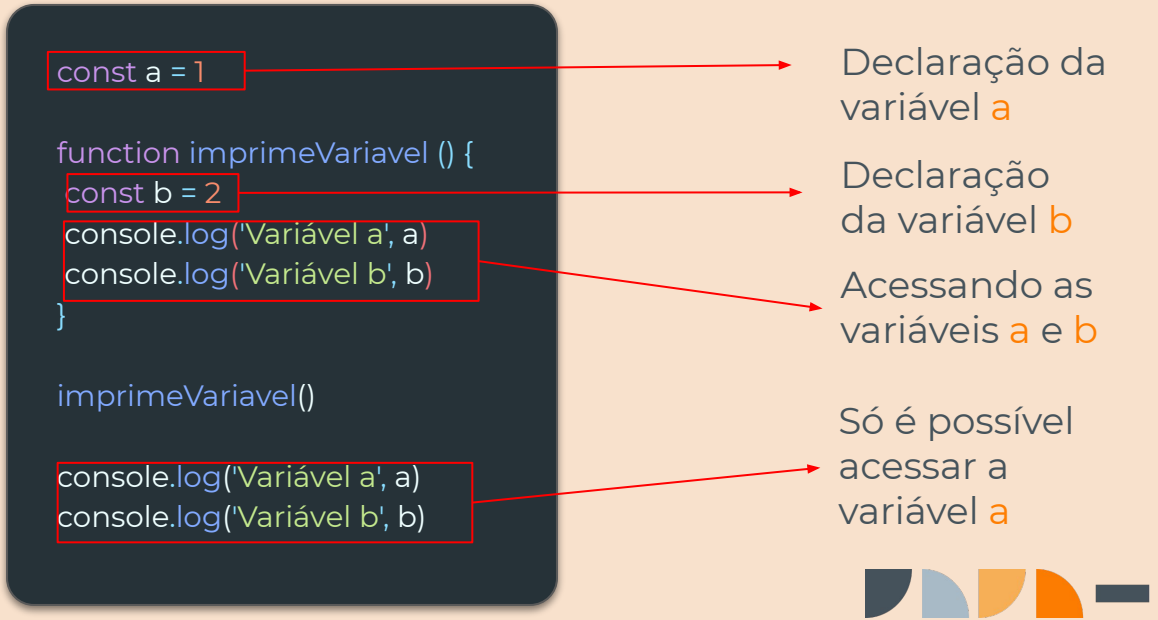
Pausa na função

- Uma função é um bloco de código com um nome que representa uma ação específica

- Pode receber **entradas** e retornar **saídas**

Escopo

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

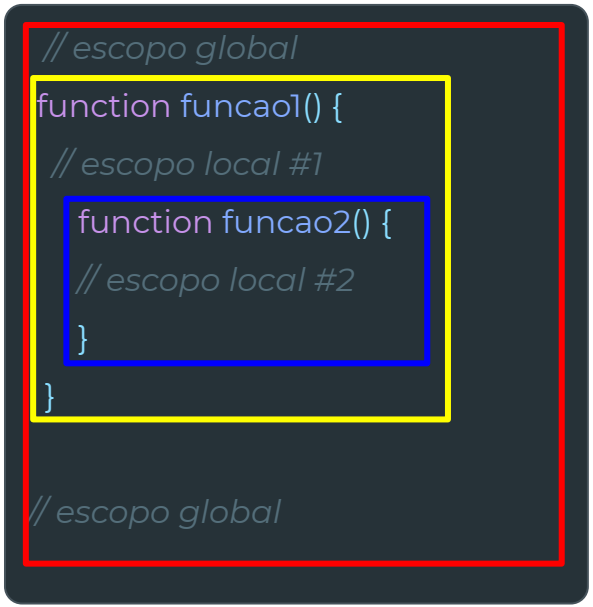


Escopo { }

- 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**

Ver Exemplo

Escopo { }



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

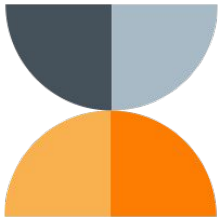
Escopo local #1 - **pai** do escopo local #2 (compartilha suas variáveis com o **filho**)

Escopo local #2

Exercício 2

- Crie uma função que receba dois números e retorne a soma entre eles
- Crie uma variável ***soma*** no escopo da sua função e tente acessá-la fora desse escopo

Pausa para relaxar 😴



- Variáveis definidas dentro de uma **função** possuem **escopo local**, ajudando a esconder a complexidade do código

Funções Anônimas

Labenu_

Funções Anônimas

- São somente uma forma **diferente** (mas bem parecida) de se declarar funções

```
const dizerOla = function (nome) {
 console.log("Olá, " + nome + "!")
}


dizerOla("Arnold") // Olá, Arnold!
```

Arrow Functions

Labenu_

Arrow Functions ➡️

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

Função anônima com sintaxe simplificada

```
const imprimeNome = nome => {
   return nome
}
```

Quando possui um único parâmetro, permite omitir os **parênteses ( )**

Arrow Functions ➡️

- Quando já começa com a palavra ***return***, é possível omitir essa palavra juntamente com as **chaves { }**

```
const somar = (a, b) => a + b
```

Resumindo

Função nomeada

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

Funções anônimas

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

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

Exercício 3

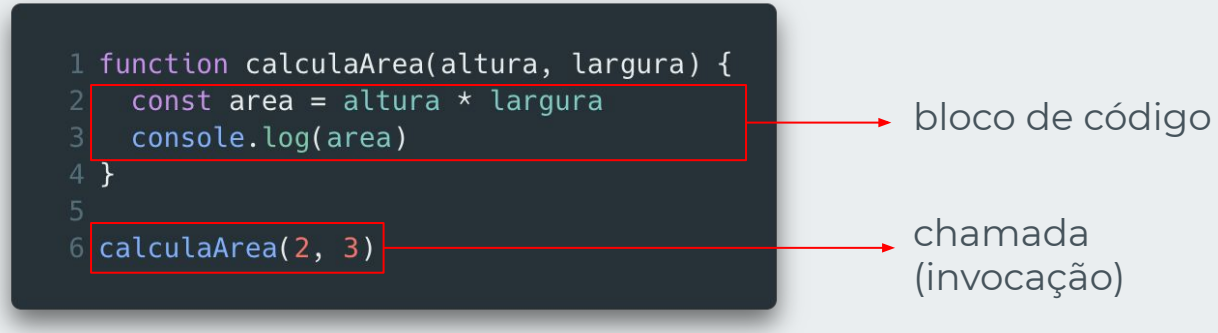
- Refaça o exercício 1 com a sintaxe de função anônima
- Refaça o exercício 2 com a sintaxe de Arrow Function

Resumo

Labenu_

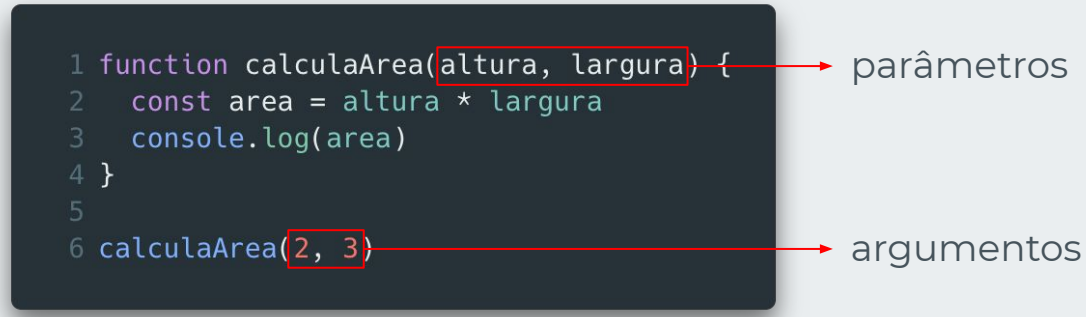
Resumo 📓

Uma função é um bloco de código representando uma **ação** específica



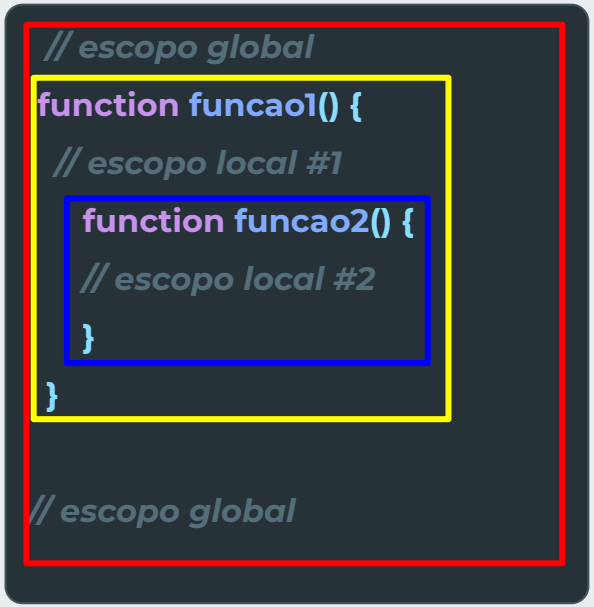
Resumo 📓

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



Resumo 📓

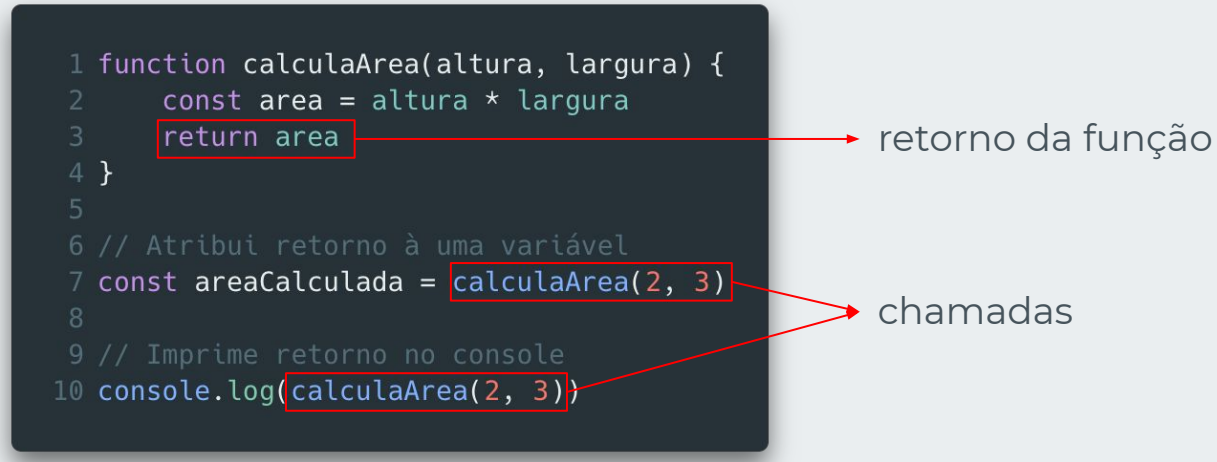
As variáveis criadas dentro das funções possuem **escopo local**, ou seja, só podem ser acessadas de dentro destas.



- 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



Resumo 📓

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

Entrada (Input) ➡

- Opcional
- Valores que serão usados na operação

- Operações

➡ Saída (Output)

- Opcional
- Valor de resultado que pode ser guardado e usado em outro lugar

Resumo 📓

Em Javascript, existem algumas formas de declarar funções

Função nomeada

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

Funções anônimas

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

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

Dúvidas? 

Labenu_

Labenu_

Obrigado(a)!