



Funções




que são funções em JavaScript

Neste módulo, vamos explorar um dos conceitos fundamentais em JavaScript: as funções. Elas são essenciais não apenas para esta linguagem, mas para a programação em geral. Entender o que são funções e como utilizá-las é crucial para escrever código eficiente, organizado e fácil de manter.

Em termos simples, funções em JavaScript são blocos de código reutilizáveis que executam tarefas específicas. Essas tarefas podem variar desde cálculos simples até operações mais complexas, como implementar regras de negócio. Por exemplo, imagine que estamos desenvolvendo um aplicativo de compras de bebidas. Podemos criar uma função que valida se a idade do usuário é maior que 18 anos, permitindo ou negando a compra de bebidas alcoólicas. Esse é um exemplo de como as funções podem ser aplicadas para verificar condições específicas dentro de um programa.

O grande benefício de usar funções é a modularização do código. Isso significa que, ao invés de repetir o mesmo bloco de código em várias partes do programa, podemos criar uma função e chamá-la sempre que necessário. Isso não só torna o código mais limpo e legível, mas também facilita sua manutenção. Quando precisamos atualizar a lógica de uma operação, como uma soma ou validação de idade, podemos simplesmente modificar a função correspondente, e a mudança será refletida em todas as partes do código onde essa função é utilizada.

Agora, como definimos uma função em JavaScript? A definição de uma função é bastante simples. Começamos com a palavra reservada `function`, seguida pelo nome da função que desejamos criar. Em seguida, entre parênteses, declaramos os

parâmetros de entrada, que são as variáveis que a função receberá para processar. Dentro das chaves `{ }` que seguem, inserimos o bloco de código que define o que a função deve fazer com esses parâmetros. 


Por exemplo, podemos criar uma função para somar dois números. Nessa função, os parâmetros seriam os números que queremos somar, e o bloco de código realizaria a soma e retornaria o resultado. No caso de uma validação de idade, o parâmetro seria a idade do usuário, e o bloco de código verificaria se essa idade é maior que um valor específico, como 18.

O uso de funções é uma prática comum e altamente recomendada na programação, pois permite que o código seja menos repetitivo e mais fácil de manter. Quando precisamos fazer uma alteração, ajustamos a função, e todas as suas instâncias no código serão atualizadas automaticamente. Isso economiza tempo e reduz a chance de erros.

Entender o conceito de funções e como utilizá-las corretamente em JavaScript é fundamental para quem deseja desenvolver software de maneira eficiente e organizada. Funções não são apenas blocos de código; elas são uma ferramenta poderosa para estruturar e simplificar o desenvolvimento de aplicações. Ao longo da aula, exploraremos outros aspectos importantes das funções em JavaScript, incluindo funções anônimas e Arrow Functions, que trazem ainda mais flexibilidade e poder à linguagem.

Importância das funções


Nesta segunda parte, discutiremos a importância das funções em JavaScript e por que elas são fundamentais no desenvolvimento de software. Compreender o valor das funções vai além de saber como declará-las; é crucial entender por que devemos utilizá-las sempre que possível e como elas podem otimizar nosso código.

Primeiramente, as funções promovem a reutilização de código, o que é essencial para a modularidade. Modularidade significa que podemos dividir nosso código em partes menores e reutilizáveis, facilitando tanto a escrita quanto a manutenção do software. Por exemplo, considere uma função que valida datas. Podemos criar uma função que recebe uma data como parâmetro e verifica se essa data não é anterior à data atual. Uma vez criada essa função, podemos reutilizá-la em qualquer lugar do nosso código onde for necessário fazer essa validação, sem precisar reescrever a lógica. 

Além da reutilização, as funções também são importantes para a abstração. A abstração nos permite esconder detalhes complexos do código, expondo apenas o que é essencial. Ao criar uma função, podemos encapsular toda a lógica complexa dentro dela, deixando visível apenas uma chamada de função com um nome sugestivo. Isso torna o código mais simples e legível. Por exemplo, se criarmos uma função chamada `validarData`, sabemos imediatamente que essa função lida com a validação de datas, sem precisar nos preocupar com os detalhes de implementação toda vez que a utilizamos.

Outro ponto importante é a organização e manutenção do código. Funções bem nomeadas e organizadas facilitam a leitura e a compreensão do código, não apenas para quem o escreveu, mas também para qualquer outra pessoa que precise dar manutenção no futuro. Imagine que você ou outra pessoa precise revisar, ou modificar o código meses, ou anos depois de ele ter sido escrito. Se as funções tiverem nomes claros e precisos, a compreensão do que cada parte do código faz será muito mais rápida, reduzindo a necessidade de ler linha por linha.

Ao criar funções, é importante escolher nomes sugestivos que descrevam claramente o que a função faz. Esses nomes não devem ser nem muito longos, nem muito curtos; o ideal é encontrar um equilíbrio que seja suficientemente descritivo. Funções com nomes bem escolhidos ajudam a manter o código organizado e facilitam a manutenção futura, evitando confusões desnecessárias.

Em resumo, as funções são essenciais para criar código reutilizável, abstrair complexidades, organizar o projeto e facilitar a manutenção. Portanto, sempre  for necessário e possível, utilize funções no seu desenvolvimento. Isso não apenas tornará o seu código mais eficiente e organizado, mas também ajudará a evitar problemas futuros quando for necessário revisar ou atualizar o sistema.

Funções anônimas

Nesta terceira parte da nossa aula, vamos abordar um conceito interessante e muito utilizado em JavaScript: as funções anônimas. Vamos entender em quais cenários elas são aplicadas, como funcionam e por que são uma ferramenta valiosa no desenvolvimento.

Para começar, é importante destacar que funções anônimas, como o nome sugere, não possuem um nome explicitamente declarado. Isso as diferencia das funções convencionais, que exigem um nome ao serem declaradas. As funções anônimas são frequentemente utilizadas em situações específicas, como quando são passadas como argumentos para outras funções, o que é bastante comum em JavaScript.

Um exemplo clássico do uso de funções anônimas é como funções de callback. Um callback é uma função passada como argumento para outra função, sendo executada em um momento específico, geralmente após a conclusão de uma tarefa. As funções anônimas são úteis aqui porque, em muitos casos, a função callback não precisa ser reutilizada em outros pontos do código; ela é necessária apenas naquele contexto específico. No exemplo a seguir, temos a função `funcaoCallBack`, que utiliza uma função anônima dentro do método `setTimeout`:

```
function funcaoCallBack() {
```

```
console.log("Esta é uma função com callback.");
```



```
setTimeout(function() {
```

```
    console.log("Função do time out.");
```

```
}, 2000);
```

```
}
```

```
funcaoCallBack();
```

Neste exemplo, a função anônima é executada após um intervalo de 2 segundos, exibindo a mensagem “Função do time out” no console. Esse tipo de uso é comum quando queremos que um bloco de código seja executado após um certo delay. Sua visualização dentro do console é a seguinte:

Figura 1 - Visualização da função anônima implementada

The screenshot shows a web application interface with a table of products and a registration form. The browser console on the right displays the following log messages:

- Esta é uma função anônima. (timestamp: 11/11/2023, 11:11:11)
- Esta é uma função com callback. (timestamp: 11/11/2023, 11:11:11)
- Função do time out. (timestamp: 11/11/2023, 11:11:13)

The table of products is as follows:

Código de Produto	Descrição do Produto	Preço (R\$)	Quantidade
PROD0001	Camisa Básica Preta	80,00	8
PROD0002	Calça Jeans Masculina	150,00	10
PROD0003	Tênis Esportivo	400,00	3
PROD0004	Mochila Escolar	350,00	10

The registration form (Cadastro de Produtos) includes the following fields and buttons:

- Código do Produto:
- Descrição do Produto:
- Preço:
- Quantidade:
- Buttons:



Outro cenário comum é o uso de funções anônimas como métodos de array. Quando trabalhamos com arrays em JavaScript, é comum utilizarmos métodos como `map`, `filter` e `reduce`. Esses métodos geralmente requerem uma função como argumento para processar os elementos do array. Ao invés de declarar uma função nomeada para isso, podemos usar uma função anônima, que executa a lógica necessária diretamente dentro do método. Veja o exemplo abaixo:

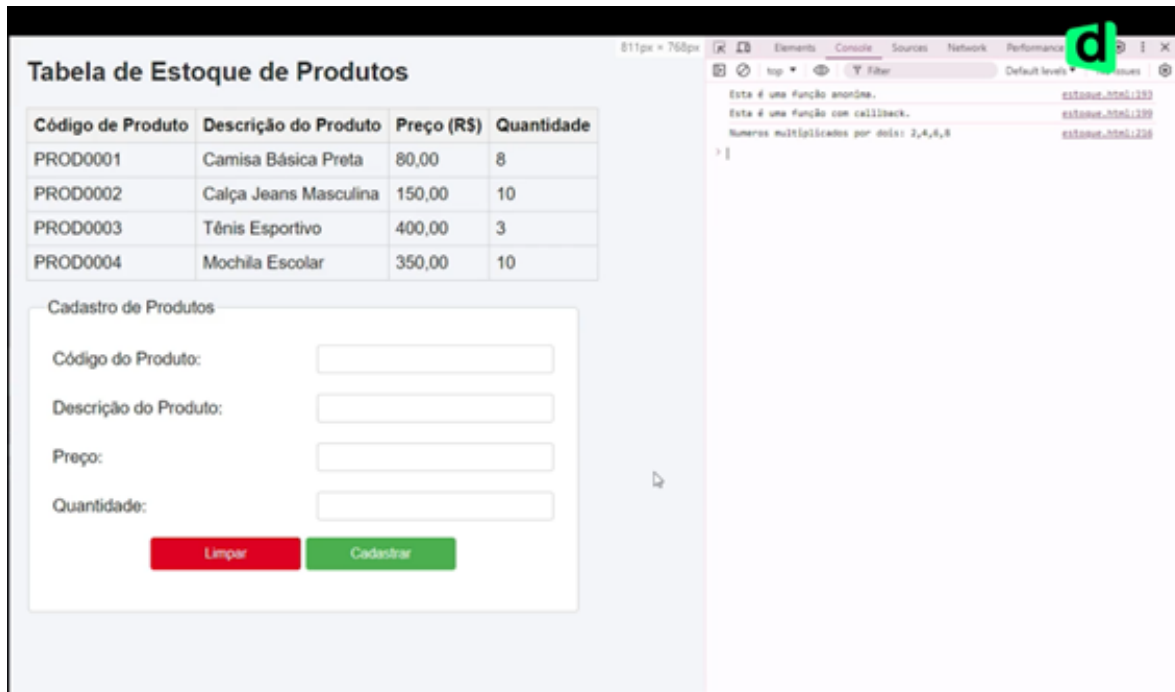
```
function funcaoAnonimaArray() {  
  
    let number = [1, 2, 3, 4];  
  
    let funcaoArray = number.map(function(num) {  
  
        return 2 * num;  
  
    });  
  
    console.log("Numeros multiplicados por dois: " + funcaoArray);  
  
}  
  
funcaoAnonimaArray();
```

Neste exemplo, a função anônima é passada diretamente para o método `map`, que itera sobre cada elemento do array `number`, multiplica-o por dois e retorna um novo array com os valores resultantes. No console, veremos a saída “Numeros

multiplicados por dois: [2, 4, 6, 8]”, confirmando que a função foi executada corretamente. Observe-o a seguir:



Figura 2 - Uso da função anônima como método de array



Fonte: elaborado pelo autor

As funções anônimas oferecem flexibilidade e são muito práticas, especialmente em casos onde uma função é necessária apenas temporariamente ou em um contexto limitado. Elas permitem que o código seja mais conciso e eliminam a necessidade de criar funções nomeadas para tarefas que não serão reutilizadas.

Em resumo, as funções anônimas são uma poderosa ferramenta em JavaScript, amplamente utilizadas tanto para callbacks quanto para métodos de arrays, entre outras aplicações. Elas tornam o código mais enxuto e direto, facilitando a escrita de funções simples e pontuais. Na próxima parte da nossa aula, continuaremos explorando mais aspectos das funções em JavaScript, aprofundando nosso conhecimento sobre esta importante característica da linguagem.

Arrow functions



Nesta última parte da nossa aula, exploraremos um tipo específico de função: as Arrow Functions. Conhecidas por oferecerem uma maneira mais curta e simplificada de escrever funções, as Arrow Functions são uma adição relativamente recente à linguagem, mas têm ganhado muita popularidade devido à sua sintaxe concisa e fácil de usar.

As Arrow Functions, ou funções de seta, são assim chamadas devido ao símbolo “=>” que utilizamos para declará-las. Elas permitem que você escreva funções sem precisar usar a palavra-chave `function`, tornando o código mais limpo e direto. A sintaxe é bastante simples: você declara os parâmetros da função entre parênteses, seguido pelo símbolo “=>” e, em seguida, escreve o corpo da função. Vamos ver como isso funciona na prática.

Primeiro, vamos declarar uma Arrow Function que realiza uma operação de soma. Em vez de usar `let` para declarar uma variável, desta vez usaremos `const`, pois não queremos que o valor dessa função seja alterado. Assim, declaramos uma constante chamada `somar`, e a ela atribuiremos uma Arrow Function:

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

Nesta declaração, `a` e `b` são os parâmetros da função, e a seta “=>” indica que estamos iniciando o corpo da função. O retorno dessa função é a soma de `a` e `b`. É importante notar que, se a função contiver apenas uma expressão, como no caso acima, podemos simplificá-la ainda mais, removendo as chaves `{ }` e a palavra `return`:



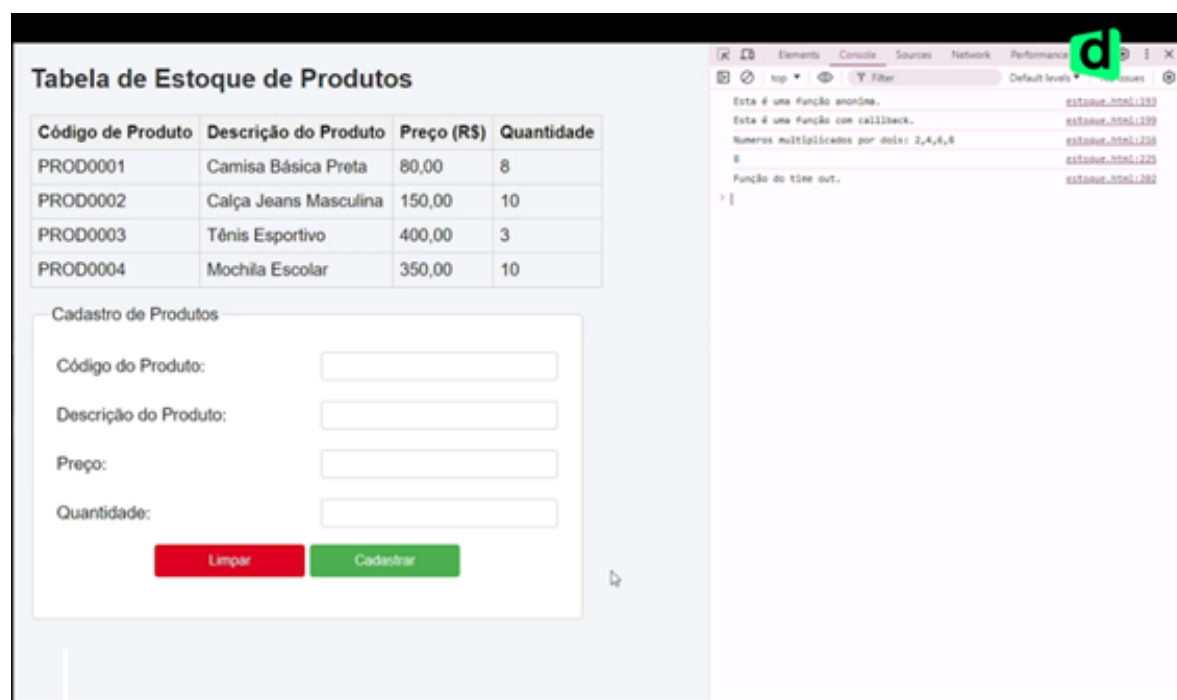
```
const somar = (a, b) => a + b;
```

Essa sintaxe mais compacta é um dos grandes benefícios das Arrow Functions, especialmente quando queremos escrever funções curtas e simples.

Agora, vamos chamar essa função e verificar seu funcionamento:


```
console.log(somar(5, 3)); // Saída esperada: 8
```

Figura 3 - Funcionamento da função 'somar'



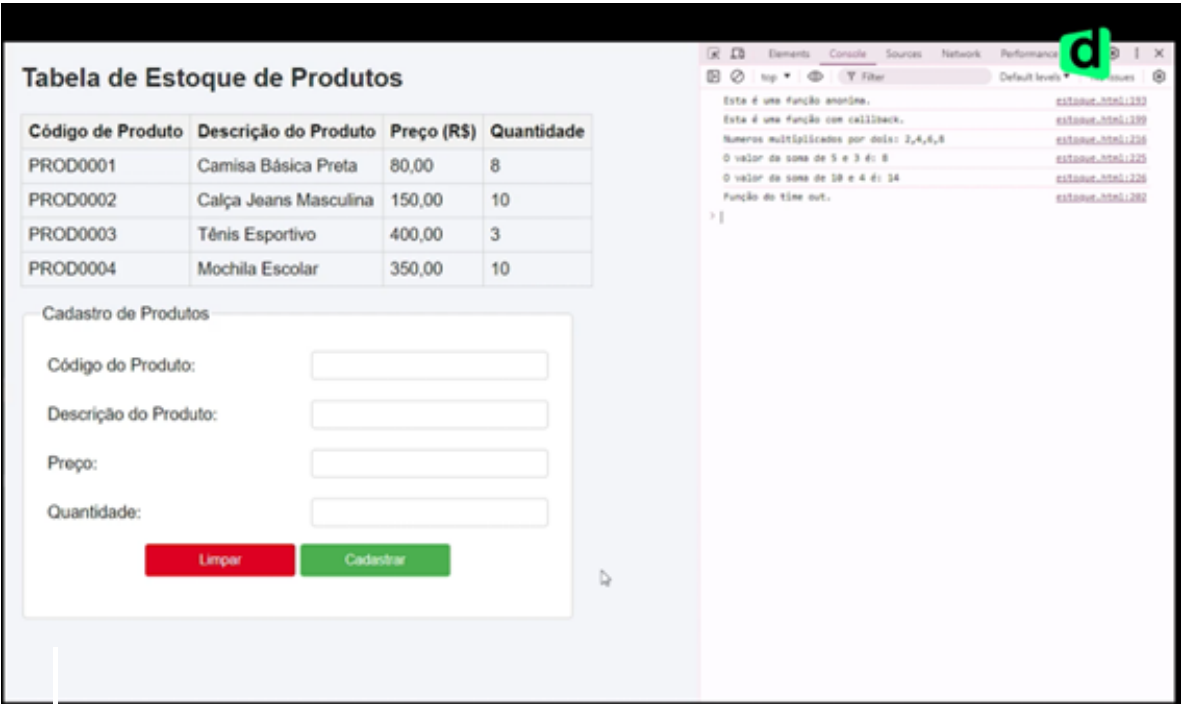
Fonte: elaborado pelo autor

Fonte: elaborado pelo autor

Neste exemplo, estamos passando os valores 5 e 3 como argumentos para a função somar, e a saída esperada é 8, que será exibida no console. Se quisermos real.  outra operação de soma, basta reutilizar a função somar com diferentes valores:

```
console.log(somar(10, 4)); // Saída esperada: 14
```

Figura 4 - Funcionamento da função ‘somar’ com uma nova operação



Fonte: elaborado pelo autor

Fonte: elaborado pelo autor

A reutilização de código, como vimos anteriormente, é uma das grandes vantagens de se trabalhar com funções, e as Arrow Functions tornam esse processo ainda mais eficiente.

Além disso, como o resultado pode ser observado nas imagens, a legibilidade dos logs pode ser melhorada, adicionando descrições mais detalhadas ao código:

```
console.log("O valor da soma de 5 e 3 é: " + somar(5, 3)); // Saída: O valor da soma de 5 e 3 é: 8
```



```
console.log("O valor da soma de 10 e 4 é: " + somar(10, 4)); // Saída: O valor da soma de 10 e 4 é: 14
```

Dessa forma, o console mostrará não apenas o resultado, mas também o que está sendo calculado, tornando mais fácil entender o que está acontecendo no código.

As Arrow Functions são especialmente úteis em situações onde precisamos escrever funções curtas e queremos evitar a verbosidade da sintaxe tradicional. Elas também têm algumas características especiais, como o fato de não terem seu próprio `this`, o que pode ser vantajoso em certos contextos, como ao trabalhar com métodos de objetos e classes.

Em resumo, as Arrow Functions oferecem uma maneira elegante e eficiente de escrever funções em JavaScript. Elas são uma excelente opção quando precisamos de simplicidade e clareza, sem comprometer a funcionalidade do nosso código.

Com isso, encerramos a nossa aula sobre funções em JavaScript. Espero que vocês tenham gostado de aprender sobre as diferentes formas de declarar e utilizar funções, desde as convencionais até as anônimas e Arrow Functions. Essas ferramentas são essenciais para o desenvolvimento moderno e, dominando-as, você estará mais preparado para escrever código limpo, organizado e eficiente.

Conteúdo Bônus

Para aprofundar seus conhecimentos sobre funções em JavaScript, sugiro que você assista ao vídeo intitulado **“FUNÇÕES EM JAVASCRIPT - Programador br - EP.90”**. Este conteúdo é apresentado por Igor Oliveira no canal Programador BR no YouTube. O vídeo oferece uma explicação prática sobre o uso de funções em

JavaScript, abordando conceitos importantes e demonstrando exemplos que complementam o que vimos em aula.



Referência Bibliográfica

ASCENCIO, A. F. G.; CAMPOS, E. A. V. de. **Fundamentos da programação de computadores**. 3.ed. Pearson: 2012.

BRAGA, P. H. **Teste de software**. Pearson: 2016.

GALLOTTI, G. M. A. (Org.). **Arquitetura de software**. Pearson: 2017.

GALLOTTI, G. M. A. **Qualidade de software**. Pearson: 2015.

MEDEIROS, E. **Desenvolvendo software com UML 2.0 definitivo**. Pearson: 2004.

PFLEEGER, S. L. **Engenharia de software: teoria e prática**. 2.ed. Pearson: 2003.

SOMMERVILLE, I. **Engenharia de software**. 10.ed. Pearson: 2019.

Prática Integradora Desenvolvimento de Software

Atividade Prática 8 - Explorando Funções em JavaScript

Objetivos:

- Compreender a definição e a importância das funções em JavaScript.
- Aplicar o conceito de funções anônimas e Arrow Functions em exemplos práticos.
- Desenvolver habilidades para criar funções reutilizáveis e concisas.

Materiais, Métodos e Ferramentas:



- Computador com um editor de código (Visual Studio Code, Sublime Text, etc.)
- Navegador web (para executar e testar o código JavaScript)
- Acesso à documentação do JavaScript (MDN Web Docs ou similar)

Atividade Prática

Primeiramente, leia atentamente o texto a seguir:

As funções em JavaScript são blocos de código que realizam tarefas específicas e podem ser reutilizadas em diferentes partes do seu código. Funções podem ser nomeadas ou anônimas, e o JavaScript oferece a sintaxe de Arrow Functions para tornar a escrita de funções mais concisa. Neste exercício, você explorará a definição de funções, funções anônimas e Arrow Functions, e aplicará esses conceitos em exemplos práticos.

Agora, vamos praticar!

PASSO A PASSO DETALHADO DA ATIVIDADE:

- **Definição de Funções:**
 - Crie um arquivo HTML e inclua um arquivo JavaScript.
 - No arquivo JavaScript, defina uma função chamada `calcularAreaRetangulo` que recebe dois parâmetros: `largura` e `altura`. A função deve calcular e retornar a área do retângulo.
 - Adicione um `console.log` para exibir a área calculada, chamando a função com valores de sua escolha.

- **Funções Anônimas:**



- No mesmo arquivo JavaScript, crie uma função chamada `exibirMensagem` que utiliza uma função anônima como callback. A função `exibirMensagem` deve receber um argumento de função (callback) e um parâmetro de mensagem, e chamar a função callback com a mensagem.
- Utilize o método `setTimeout` para passar a função anônima como callback e exibir a mensagem após 2 segundos.


- **Arrow Functions:**

- Defina uma Arrow Function chamada `duplicarValores` que recebe um array de números e retorna um novo array com todos os valores multiplicados por 2.
- Utilize a Arrow Function `duplicarValores` em conjunto com o método `map` para processar um array de números e exibir o resultado no console.

- **Refatoração e Teste:**

- Refatore qualquer função que possa ser simplificada utilizando Arrow Functions.
- Teste todas as funções no console do navegador para verificar se estão funcionando conforme o esperado.

Gabarito Esperado

- **Definição de Funções:** A função `calcularAreaRetangulo` deve calcular corretamente a área do retângulo e exibir o resultado no console. 
- **Funções Anônimas:** A função `exibirMensagem` deve chamar corretamente a função anônima após 2 segundos e exibir a mensagem no console.
- **Arrow Functions:** A Arrow Function `duplicarValores` deve multiplicar corretamente todos os valores do array por 2 e exibir o novo array no console.

Certifique-se de que cada parte da atividade esteja implementada e testada conforme as instruções para garantir que o código está funcionando corretamente.

[Ir para exercício](#)