

JavaScript básico





<title>

AGENDA DO </> </> </> MÓDULO

</title>



Agenda do módulo



Anotações



{Introdução a funções}

{Trabalhando com parâmetros}

{Usando funções}



<title>

MISSÃO DO MÓDULO

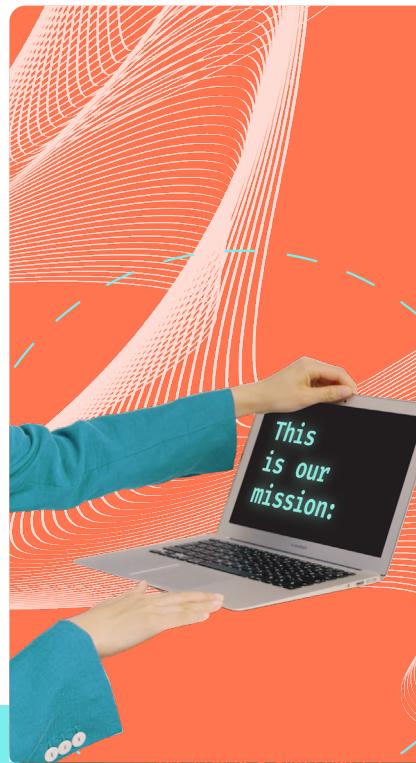
</title>

Usar as **funções** em Javascript de modo eficiente.

Usar funções



traz resultados!



```
let start = function() {  
    console.log("Usar função")  
};
```

```
let slogan = function
  console.log("Us")
```



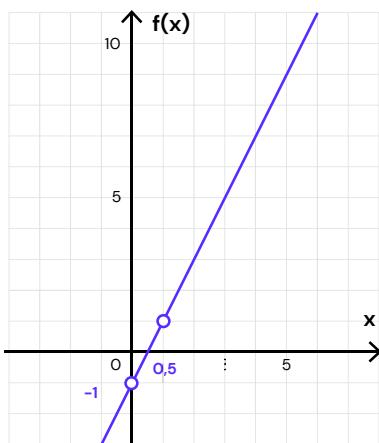


{01.}

Introdução a funções

Provavelmente você se lembra de estudar equações matemáticas na escola. Algo do tipo $f(x) = 2x - 1$, em que dependendo do valor de x e $f(x)$, chamadas de variáveis, a gente consegue desenhar um gráfico com os resultados dessas contas.

$$y = 2x - 1$$

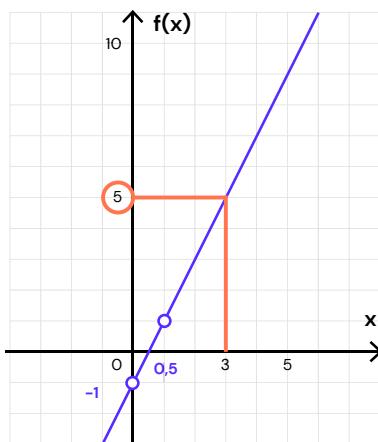




Para exemplificar melhor, vamos fingir que $f(x)$ é o total de xícaras de café que você toma enquanto programa e X é a quantidade de horas em que você trabalha.

Você sabe que hoje você vai ficar trabalhando por 3 horas, logo você pode já preparar 5 xícaras de café.

$$y = 2x - 1$$



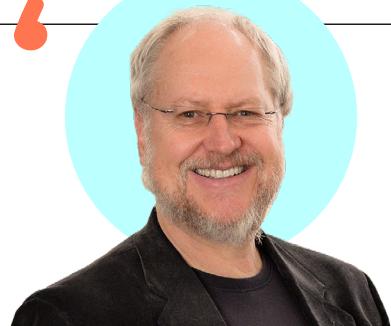
Cada função deve ser usada apenas para as variáveis para as quais foram criadas!

Da mesma forma que nas funções matemáticas, as funções em javascript relacionam e realizam operações com variáveis, utilizando parâmetros definidos por nós programadores.

Assim, uma função, ao receber dados de entrada, retornam um resultado que precisamos para o funcionamento do nosso script.



“



A melhor coisa a respeito do JavaScript é sua forma de implementar funções.

Douglas Crockford

A função é um pedaço de código que podemos reutilizar quantas vezes quisermos.
Basta “chamar” o nome da função.

E, mesmo sem você perceber, você já tem usado as funções

Lembra quando apresentamos a biblioteca Math?
Alguns dos comandos que apresentamos como o Math.floor e o Math.ceil são funções!

Input	Math.floor	Math.ceil	Math.round	Math.trunc
3.1	3	4	3	3
3.6	3	4	4	3
-1.1	-2	-1	-1	-1
-1.6	-2	-1	-2	-1



Pronto para criar suas próprias funções?

De uma maneira simplificada, para declarar uma função começamos com a keyword “function” (uma das palavras reservadas do javascript).

Em seguida, atribuímos um nome que a ela e entre parênteses incluímos um ou mais parâmetros separados por vírgula.

O código a ser executado fica dentro de um par de chaves, que chamamos de corpo da função.

```
function nome(parâmetro) {  
    corpo da função  
}
```

A função que demos o nome de “aoQuadrado”, recebe como argumento um número qualquer e sua função é calcular o quadrado desse número multiplicando ele por ele mesmo.

```
function aoQuadrado(numero) {  
    return numero * numero;  
}
```

Para que a função seja executada, precisamos chamar essa função. Para isso, digitamos seu nome com o argumento entre parênteses.

```
function aoQuadrado(numero) {  
    return numero * numero;  
}
```

```
aoQuadrado(4)
```



Também podemos usar o resultado dessa função posteriormente. Para isso, atribuímos a função e seu argumento para uma variável, como essa que está aparecendo na sua tela. O resultado fica salvo na variável “res” e pode ser utilizado em outras partes do código.

```
function aoQuadrado(numero) {  
    return numero * numero;  
}  
  
let res = aoQuadrado(4)
```

Esse resultado chamamos de retorno.

As funções podem receber mais de um parâmetro como pode também receber nenhum.

No exemplo abaixo, quando a função curso for chamada, ela criará um alerta na página com a frase “Eu estudo Javascript”. Como ela não possui variáveis, ela não necessita de um argumento.

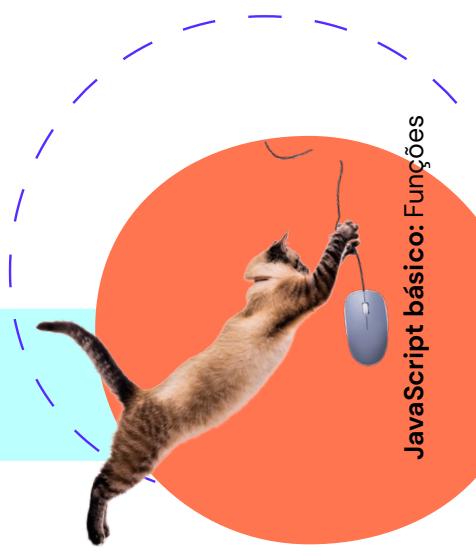
```
function curso() {  
    alert("Eu estudo javascript");  
}
```

```
<title>
```

PULO DO GATO

```
</title>
```

- Não confunda **funções** com **procedimentos**.





No geral, procedimentos são sequências de instruções. Uma função retorna um valor e um procedimento executa comandos.

Em Javascript ambos têm a mesma sintaxe, mas é uma boa prática que todas as funções recebam entradas e retornem saídas.

```
<title>
```

MÃO + NA MASSA

```
</title>
```

Altere a função “`curso()`” para que a palavra “`javascript`” mude (dependendo do argumento passado).



Para a função retornar um resultado, use a palavra “`return`”.

Também podemos criar funções para fazer outras coisas, como criar um alerta na tela (com o uso da keyword “`alert`”), por exemplo.

- › Crie uma função para converter temperaturas em Celsius para Fahrenheit.
- › Use como argumento o valor em Celsius.
- › Chame `console.log` para ver a conversão no inspetor.
- › Use a equação:

$$T(^{\circ}\text{F}) = T(^{\circ}\text{C}) \times 1,8 + 32;$$

Caso você tenha ficado com alguma dúvida, acesse o gabarito no gitbub e o passo a passo em vídeo no seu material de apoio.



{02.}

Trabalhando com parâmetros

Na programação existem duas palavras muito usadas – que as pessoas confundem bastante também: Parâmetros e argumentos. Apesar de serem usadas como sinônimos elas possuem valores semânticos diferentes e é preciso se atentar a isso.

Os parâmetros são as variáveis definidas dentro dos parênteses da função e que irão no futuro receber os valores de entrada: os argumentos.

```
function aoQuadrado(numero) {  
    return numero * numero;  
}  
  
aoQuadrado(4)
```

Veja esse exemplo:

```
function apresentacao(nome, cidade, filmeFavorito) {  
    console.log("Meu nome é ${nome}, moro em ${cidade}, e meu  
    filme favorito é ${filmeFavorito}`)`)  
}  
  
apresentacao("Caio", "Curitiba", "Curtindo a vida adoidado")
```

Essa função tem como parâmetro o nome, a cidade e o filme favorito da pessoa que está se apresentando.

Podemos chamar a função passando como argumento os dados dessa pessoa.



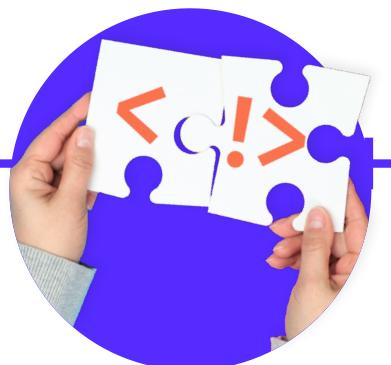
Ou seja, nós passamos **argumentos** para a função e não parâmetros.

Nós deixamos a função parametrizada para valores que ela receberá futuramente e executamos a função com os argumentos.

```
<title>
```

MÃO + + NA MASSA

```
</title>
```



Crie uma função de adição com dois parâmetros que receberá dois números como argumento.

No corpo da função, crie uma lógica com a adição desses dois números e retorne o resultado.

Em javascript, se você não passar argumentos para algum dos parâmetros da função ele retornará o valor de `undefined`, que em português quer dizer **indefinido**.

Em alguns casos é interessante que a função já seja definida com parâmetros padrão para que não precisem ser passadas posteriormente.

Imagine a seguinte situação: Uma empresa de ônibus fretado sempre define o preço da passagem de acordo com o número de passageiros. Para isso foi criada a seguinte função.



```
function price(passengers, totalPrice = 3000) {  
  let pricePerPassenger = totalPrice / passengers;  
  console.log(`O preço da viagem por passageiros  
  será de ${pricePerPassenger}`);  
}
```

Nessa função é necessário que se entre apenas com o argumento do número de passageiros da viagem, já que o argumento do preço já foi.

Como você acha que ficaria a sua função se você tivesse uma lista de **10 parâmetros**?

Nesse exemplo, temos uma função que recebe e imprime 10 números

```
function (numero1, numero2, numero3, numero4, numero5,  
numero6, numero7, numero8, numero9, numero10) {  
  console.log(numero1);  
  console.log(numero2);  
  console.log(numero3);  
  console.log(numero4);  
  console.log(numero5);  
  console.log(numero6);  
  console.log(numero7);  
  console.log(numero8);  
  console.log(numero9);  
  console.log(numero10);  
}
```

Veja quantos parâmetros, ficou muito extenso não foi?

Mas não se preocupe, vou te mostrar um recurso que vai te ajudar a reduzir drasticamente tudo isso.



Para fazer isso da melhor maneira, podemos usar o rest operator e o spread operator, que agrupa parâmetros em uma única variável.

Veja como o código ficou menor e mais simples:

```
function exibeNumeros ( ... numeros ) {  
  console.log( ... numeros );  
}
```

REST OPERATOR

SPREAD OPERATOR

Basicamente trocamos todos os parâmetros para um único parâmetro. Tanto na declaração quanto no uso.

Se adicionarmos o array dentro do outro array apenas especificando seu nome dentro no novoArray, isso injetará o array dentro e o resultado seria **[1, 2, [3, 4, 5, 6, 7, 8, 9, 10, 11, 12]]**.

Ou seja, criamos um tipo de matriz. Mas queremos combinar os números para termos um array único, com os números totalmente desagrupados.

É nesse momento que entra o spread operator que nada mais é do que adicionar esses três pontos na frente do nome do arrayPrincipal. Dessa forma o spread operator “desempacotará” os números no novoArray. Olha só como ficaria:

```
const arrayPrincipal = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12]  
const novoArray = [1, 2, ...arrayPrincipal]
```

Console +

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```



O Spread operator não precisa ser usado apenas com números. Imagine uma lista de pessoas que estão inscritas num curso.

```
const list = ["Bruno", "Gabriel", "Júlia", "Manuela"];
```

A partir do momento que novas pessoas se inscreverem nesse curso, podemos utilizar o spread operator para criar uma nova lista de inscritos.

Assim o resultado da nova lista vai receber o nome dos que já estavam na lista e dos novos inscritos.

```
const newList = ["Geovana", "Tamara", ...list]
```

Console

```
const list = ["Geovanna", "Tamara", "Bruno",  
"Gabriel", "Júlia", "Manuela"];
```

O spread operator nos ajuda a manter o código mais limpo, pois põe em prática o conceito de **imutabilidade**, o que permite ter mais controle e segurança sobre nosso código.

E como isso fica dentro de uma função?

Costumamos usar o rest operator, os três pontinhos, na entrada da função, onde declaramos o parâmetro.

Assim, você consegue trabalhar com os valores descompactados usando posteriormente o spread operator. Veja esse exemplo:

```
function sum(...values) {  
  let total = 0;  
  
  for (let i = 0; i < values.length; i++) {  
    total = total + values[i];  
  }  
  
  console.log(`A soma é ${total}`);  
}
```



<title>

PULO DO GATO

</title>

- Em JavaScript, não há exigência de que o número de argumentos corresponda ao número de parâmetros.

Se você passar **mais argumentos** do que os parâmetros declarados para recebê-los, os valores passarão **perfeitamente**.

Se você passar **menos argumentos** do que os parâmetros declarados, cada parâmetro sem correspondência será tratado como uma variável "**undefined**".



Caso precise saber quantos argumentos foram passados para a função, utilize: **.length**. Pode ser útil caso sua função execute pedaços de código a depender do número de parâmetros passados.



<title>

MÃO + + NA MASSA

</title>

Math e Spread Operator

Na biblioteca **Math** temos a função para descobrir o maior número em uma lista. Você recebeu o seguinte array e precisa retornar seu maior valor:

arr = [1, 3, 54, 23, 89, 8, 7, 0, 1, 44, 13, 11]

Utilizando o **Math.max** e o array acima como argumento, retorne seu maior valor. Lembre-se, se ele for passado como array, o retorno será NaN.

Caso você tenha ficado com alguma dúvida, acesse o gabarito no gitbub e o passo a passo em vídeo no seu material de apoio.



{03.}

Usando funções

Funções anônimas são funções que não estão associadas a um nome, porque nem sempre elas vão precisar receber um nome.



No exemplo abaixo, foi criado uma função anônima e associei seu resultado a uma variável, assim, caso seja preciso utilizar seu resultado em outro lugar, eu consigo.

```
let mensagem = function() {  
  | console.log("Essa função é anônima")  
}
```

Uma função anônima que está relacionada a uma variável só terá seu retorno acessível onde a variável pode ser chamada.

Assim como uma função nomeada, as funções anônimas também podem receber argumentos.

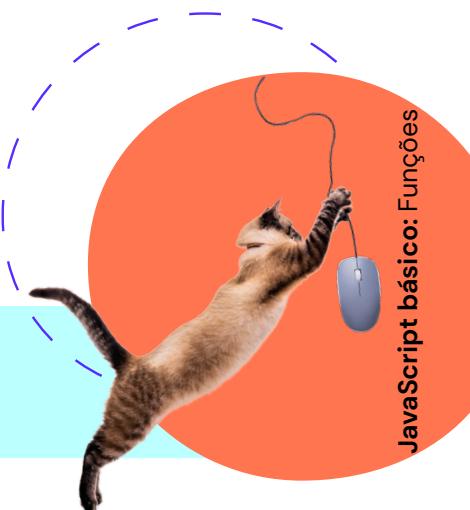
```
ARGUMENTO  
let hello = function (platform) {  
  | console.log("Seja bem vindo a " + platform);  
};  
  
hello("Conquer!");
```

```
<title>
```

PULO DO GATO

```
</title>
```

- O navegador às vezes infere o nome de uma função anônima, quando ela é atribuída a uma variável.





Como o JavaScript consegue lidar com funções de alta-ordem (função que recebe ou retorna uma função) podemos passar funções anônimas como parâmetros de outras funções. Assim como o exemplo abaixo:

```
function() é parâmetro  
para setTimeout  
  
setTimeOut(function () {  
    console.log("Estou aprendendo a passar  
    funções como parâmetro de funções");  
, 4000);
```

Um conceito interessante quando falamos de funções anônimas são as **IIFEs** (Immediately Invoked Function Expression), que são funções executadas imediatamente após a sua declaração. Como você pode ver no exemplo abaixo:

```
let multiplicationResult = (function(x,y) {  
    return x * y;  
) (3, 4);  
  
console.log(multiplicationResult);
```

Quando criamos funções anônimas que serão usadas apenas uma vez, usamos de maneira desnecessária a memória. Além disso, podemos causar confusão entre os nomes delas e gerar erros no nosso código.

Para evitar que isso ocorra podemos usar as IIFE como no exemplo acima. Ou seja, criamos a função de multiplicação multiplicationResult e logo após sua criação a chamamos passando os argumentos "3" e "4".

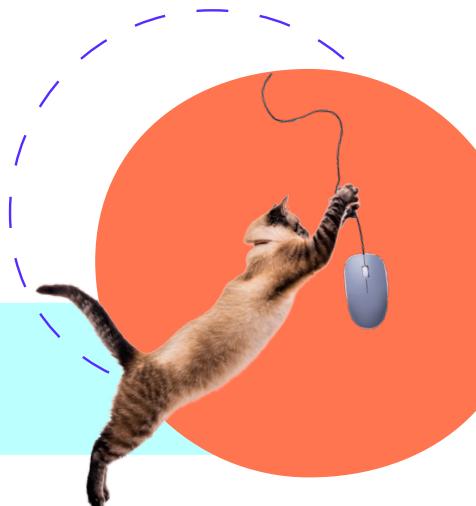


<title>

PULO DO GATO

</title>

- Use funções anônimas com moderação!



Mas como vimos na maioria das vezes é uma boa prática nomear a função para poder referenciá-la com mais praticidade e deixar o código mais legível.

Funções nomeadas também são muito melhores quando o assunto é encontrar um erro no código, caso ele apareça no console do navegador.

Seja sempre objetivo na hora de nomear sua função, essa é uma boa prática que deixa o código mais fácil de ler e não se perder caso ele venha a crescer e contar com a cooperação de outros programadores. Se o nome da função for composto por mais de uma palavra, use a convenção de nomenclatura camelCase, como usamos para variáveis.

- Por exemplo “aoQuadrado”, “idadeDoAluno”.

E apesar de não ser uma regra, é considerada uma boa prática nomear as funções em inglês.



+

+

+

+

+

+

Um nome longo e descritivo é melhor do que um nome curto e enigmático. Um nome longo e descritivo é melhor que um comentário longo e descritivo.

Robert C. Martin

<title>

MÃO + + + NA MASSA

</title>



Crie uma função anônima para calcular o IMC de uma pessoa. Associe essa função a uma variável com o nome da pessoa em que você está calculando o IMC.

O cálculo de IMC é feito pela seguinte expressão:

IMC = peso / altura² (em metros)

Caso você tenha ficado com alguma dúvida, acesse o gabarito no gitbub e o passo a passo em vídeo no seu material de apoio.



<title>

DESAFIO CONQUER

</title>



<body>

Crie funções para cada botão que uma calculadora simples tem:

- › Função de adição.
- › Função de subtração.
- › Função de multiplicação.
- › Função de divisão.

Crie uma variável vazia chamada **firstNumber**, uma chamada **secondNumber** e uma chamada **result**. e teste os valores no console. log.

</body>



<title>

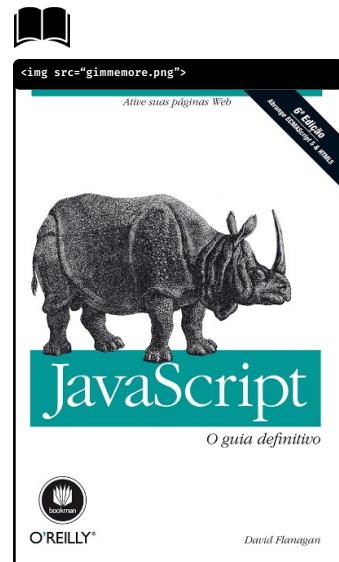
QUERO MAIS

</title>

👉 You don't know JS yet

👉 Documentação Mozilla

👉 Aprender JavaScript com um jogo



Livro: **JavaScript: O Guia Definitivo**
David Flanagan

```
if (interested)
// gimmemore
```



Anotações

-
-
-
-
-
-

