

Funções

O que é uma Função?

Soluções Reutilizáveis

Você já deve ter percebido que, quando entramos no mundo da programação, nosso objetivo é encontrar as melhores soluções para problemas comuns e rotineiros através de linhas de código e usando muita lógica!

Qualquer ação do dia a dia que envolva resolver problemas, como a execução de operações bancárias, prestar algum tipo de serviço, como realizar o pedido de um prato em um restaurante, ou gerar um relatório de vendas, por exemplo, vai demandar que alguém " programe " essas tarefas.

Em algum momento, você perceberá que tarefas simples são realizadas através de pequenas rotinas que podem ser reaproveitadas em diversas situações diferentes, gerando **soluções reutilizáveis**.

*“Quando temos um pequeno trecho de código que realiza uma tarefa mais simples, chamamos comumente de **FUNÇÃO**.”*

A vantagem de utilizar funções dentro do seu código, é que você não precisa reescrever a rotina novamente, apenas “chamá-la” quando for necessário.



Vamos pensar sobre o funcionamento de um restaurante. Para que tudo funcione bem, é preciso que os setores estejam separados e tenham bem definidas suas respectivas funções, por exemplo:

- na cozinha serão preparados os pratos disponíveis no cardápio;
- os garçons farão o trabalho de anotar pedidos e servir as mesas;
- os entregadores ficam responsáveis pelos pedidos de delivery;
- a equipe de limpeza cuida para que o ambiente esteja sempre agradável;
- o caixa se encarrega de encerrar as contas, etc.



www.shutterstock.com · 718840747

Nesse modelo que descrevemos, tudo está bem **“separado”** e **“definido”** para otimizar o serviço prestado para os clientes.

Em programação, o uso de funções tem essa mesma pegada: **separar** cada ação que precisa ser executada e **definir** como, onde e quando essas funções devem acontecer. Em outras palavras, trabalhar com **funções é uma forma de encapsular um código**, para o utilizarmos mais facilmente, quantas vezes forem necessárias.

ATENÇÃO!

Em programação, é muito comum utilizarmos o termo “método” para citar o uso de “funções”

Funções

Construção e chamada

Criando minhas funções

Chegou o momento de aprendermos a criar nossas próprias funções. E, só pra lembrar, funções são um encapsulamento de determinadas instruções ao computador, visando sempre que necessário repeti-las!

A seguir, apresentamos a sintaxe de uma função:

```
function nome_da_funcao() {  
    comandos  
}
```

Sempre que formos construir uma função, precisamos utilizar a palavra chave **“function”** no início da linha, seguida do **nome da função** (que deve obedecer as regras de nomenclatura camelCase), seguida de parênteses **“()”** e os comandos que devem ser realizados entre as chaves **“{ }”**.

camelCase é uma **convenção de nomenclatura** usada por desenvolvedores para deixar o seu código mais legível para outros DEVs. Em resumo, as principais regras são:

- Iniciar sempre o nome de uma função com letra minúscula;
- Não iniciar o nome de uma função com números;
- Não utilizamos espaços entre nomes compostos, ao invés disso, usamos a primeira letra da segunda palavra em maiúsculo. Exemplo: minhaFuncao()
- Não utilizar acentos e/ou caracteres especiais no nome da função, exceto quando a linguagem permitir, por exemplo, em JavaScript é possível o uso dos caracteres “underline” (`_`) e “dólar” (`$`);



Vamos construir nossa primeira função, que realiza uma tarefa bem simples: Cumprimentar o usuário! Nosso código ficaria assim:

```
function saudacao(){  
    console.log("Olá tudo bem!")  
}
```

No código acima temos:

- a palavra reservada “function”;
- o nome da função “saudacao()”
- e o comando “console.log(“Olá tudo bem!”)” que está entre as chaves e imprime a saudação na tela!

Mas vale ressaltar que podemos fazer qualquer instrução dentro da função, como loops, condições e afins! Legal né?

Acabamos de criar a nossa função, mas isso não é o suficiente para vermos seu funcionamento, agora precisamos chamá-la para que algo aconteça!

Chamada de Função

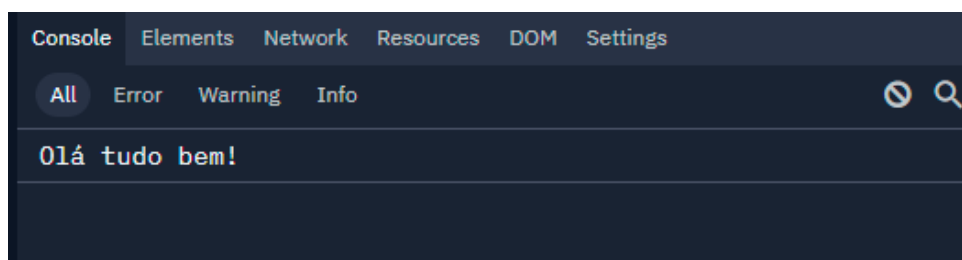
Para executar uma função, precisamos chamá-la pelo seu nome seguido pelo par de parênteses, também conhecida como “assinatura da função”.

Lembre-se de dar um nome que faça sentido ao que a função irá realizar. Em geral, você pode usar verbos para indicar o que a função faz. Ações como: **somar**, **multiplicar**, **consultarSaldo** e etc, são alguns exemplos.

Nosso código ficaria assim:

```
saudacao();
```

E o resultado será a impressão da frase no console:



Funções

O que é um Parâmetro?

Deixando as funções dinâmicas

Até esse ponto, aprendemos a criar funções de uma maneira super bacana, sempre se atentando ao uso da palavra reservada “function” seguida do nome da função junto ao par de parênteses.

```
// Lembre-se de sempre usar a palavra reservada function, sem ela o computador não saberá que é para criar uma função
```

```
function nomeDaFuncao() {  
    // seu código aqui  
}
```

Mas, nossas funções ainda não estão legais, porque ela sempre faz a mesma coisa. Você deve estar se perguntando: “não era esse o objetivo?” E a resposta é sim, porém, podemos ter alguns problemas com isso. Imagine a seguinte situação:

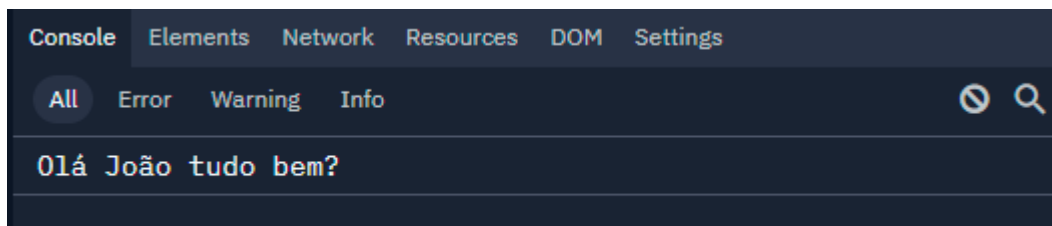
Vamos melhorar nossa função de saudação? Ao invés de simplesmente cumprimentar a pessoa, que tal se escrevêssemos o nome dela também? Para isso, precisamos escrever nossa função de forma que ela receba o nome de uma pessoa como “parâmetro”, como mostra o exemplo a seguir:

```
function saudacao(nome){  
    console.log("Olá " + nome + " tudo bem?")  
}
```

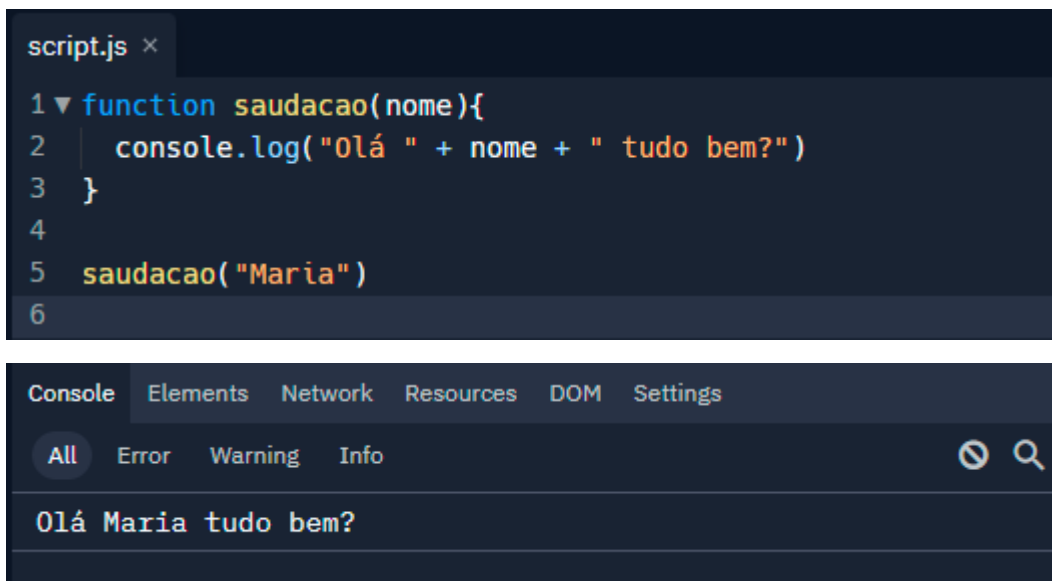
Lembre-se de que apenas criar a função, não é o suficiente para ver sua execução! Precisamos chamar a função também, mas desta vez, passando uma informação a mais como parâmetro, como segue o exemplo abaixo:

```
saudacao("João");
```

O resultado deve exibir a frase concatenada com o nome que passamos:



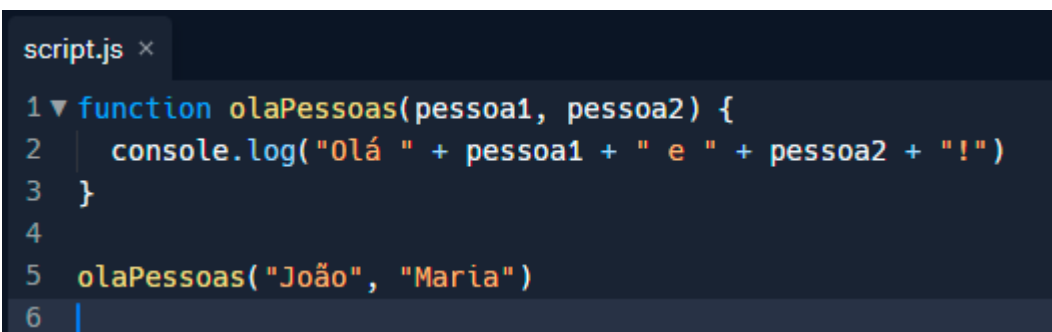
E se trocássemos esse parâmetro, executando novamente a função com outro nome, qual seria o resultado? Vamos testar com o nome “Maria”:



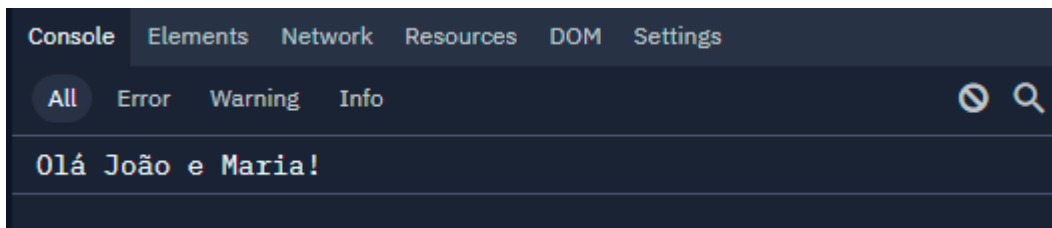
Você consegue perceber que utilizamos a mesma função para gerar resultados diferentes, apenas trocando a informação no parâmetro? Pois é, por isso chamamos essas funções de **função dinâmica**!

Função com mais de um parâmetro

As funções dinâmicas podem receber mais de um parâmetro em sua composição, para isso, precisamos organizar tudo sequencialmente, separando cada parâmetro por vírgula (,). Vejamos um exemplo:



E esse seria o nosso resultado no console:



Observe que a sequência que utilizamos na passagem dos parâmetros, na chamada da função, é a mesma utilizada para imprimir os resultados. Desta forma, “João” foi atribuído ao parâmetro “pessoa1” e “Maria” foi atribuído ao parâmetro “pessoa2”.

Situação Problema 1:

Recebemos o desafio de construir uma função que realize a soma de dois valores, por exemplo: 10 e 20. Como você faria para programar essa tarefa?

Talvez você pense em algo como:

```
function somar() {  
    10+20  
}
```

De início, pode fazer sentido, afinal, estamos cumprindo o requisito de somar dois números. Mas, por que exatamente 10 e 20? E se quiséssemos somar 60 + 40?

Observe que da forma que essa função está escrita, ela é estática, ou seja, realizará sempre o mesmo cálculo, com os mesmos valores. Se precisarmos mudar os valores, teríamos que recriar essa função alterando o código.

Como vimos anteriormente, podemos transformar essa função estática em uma função dinâmica, com o uso de **parâmetros**. Eles vão nos ajudar a resolver esse problema de uma forma bem simples.

Para isso, iremos definir que a função irá receber dois parâmetros, **numA** e **numB**, como mostrado a seguir:

```
function somar(numA, numB) {  
  
}
```

Esses parâmetros vão se comportar como variáveis dentro da nossa função. Eles receberão os valores que forem passados no momento da chamada da função, e então poderemos fazer o que quisermos com eles como, por exemplo, somá-los e imprimir o resultado pelo `console.log()`.

```
function somar(numA, numB) {  
    console.log(numA + numB) //vai imprimir 30  
}  
somar(10,20) //chamada da função passando dois números
```

Dessa forma, independente dos valores passados durante a chamada, a função irá executar a soma destes valores. E se quisermos somar outros números, basta substituir os números que estão sendo passados como parâmetros na chamada da função. A ideia é garantir uma melhor performance e aplicabilidade da nossa função!

Situação Problema 2:

Vamos criar um programa em JavaScript que funcione como uma calculadora, que deve executar as operações de soma, subtração, multiplicação e divisão, recebendo dois números como parâmetros! É importante que esse programa utilize funções para realizar os cálculos e exibi-los no console.

```
function somar(num1, num2) {  
    console.log("Resultado da soma: ", num1 + num2);  
}  
function subtrair(num1, num2) {  
    console.log("Resultado da subtração: ", num1 - num2);  
}  
function multiplicar(num1, num2) {  
    console.log("Resultado da multiplicação: ", num1 * num2);  
}  
function dividir(num1, num2) {  
    if (num2 != 0) {  
        console.log("Resultado da divisão: ", num1 / num2);  
    } else {  
        console.log("Impossível dividir por 0");  
    }  
}  
somar(2, 4);  
dividir(8, 0);  
dividir(8, 2);  
subtrair(5, 0);
```


Funções

O que é um Retorno?

Vamos resgatar o exemplo que trabalhamos há pouco, que busca receber dois valores numéricos e imprimir a soma deles:

```
function somar(numA, numB) {  
    console.log(numA + numB) //vai imprimir 30  
}  
somar(10,20) //chamada da função passando dois números
```

No exemplo acima, imprimimos o valor da soma com o `console.log()`, mas se precisarmos utilizar esse valor posteriormente, como faríamos?

Nesse caso, precisamos informar para nossa função que o resultado da soma precisa ser **retornado** e armazenado em uma variável para ser utilizado posteriormente em outros trechos do código.

Surge então, uma outra palavra reservada que irá nos ajudar: **return** ou em português, **retornar**. Como o nome já diz, ela irá nos retornar um valor, seja ele número, string, array, etc.

Vejamos como ficaria nosso código utilizando o retorno:

```
function somar(numA, numB) {  
    return numA + numB  
}
```

Nesse caso, a função que estamos usando irá retornar o resultado da soma. Mas retornar para onde? Podemos responder a essa pergunta observando a chamada a seguir:

```
var resultado = somar(10, 20);
```

Desta forma, estamos armazenando o “retorno” da função `somar()` dentro da variável “resultado” e a partir daqui, podemos fazer quaisquer outras operações com ela, como por exemplo, imprimir “resultado” ou utilizar a variável para novos cálculos.

Vamos ver um exemplo disso a seguir:

Situação Problema 3:

Vamos reescrever o exemplo da calculadora que efetua as quatro operações (somar, subtrair, multiplicar e dividir), mas desta vez, utilizaremos “**return**” para tratar as informações:

```
function somar(num1, num2) {
    return num1 + num2;
}
function subtrair(num1, num2) {
    return num1 - num2;
}
function multiplicar(num1, num2) {
    return num1 * num2;
}
function dividir(num1, num2) {
    if (num2 !== 0) {
        return num1 / num2;
    } else {
        return "Impossível dividir por 0";
    }
}

var resultadoSoma = somar(2, 4);
var resultadoSubtracao = subtrair(5, 0);

console.log("Soma= ", resultadoSoma);
console.log("Subtração= ", resultadoSubtracao);
console.log("Soma + Subtração= ", resultadoSoma + resultadoSubtracao);
```

Vamos só esclarecer alguns pontos importantes:

Em uma função, não é obrigatório o uso de return. Você precisa entender primeiro o que sua função precisa fazer, para definir se ela irá apenas realizar uma ação, como por exemplo exibir um console, ou se ela irá devolver algo para ser usado em algum momento do nosso código.

E, por último, **o return é SEMPRE a última linha executada por uma função.** Sempre que o computador interpreta essa linha, ele entende que a função acabou ali. Qualquer código após o return de uma função será desconsiderado:

```
function dizerOla(){  
    return "olá"  
    // o console logo a baixo não será executado devido ao return acima!  
    console.log('Oi também!')  
}
```

Funções

Resolução de Problemas com Funções em Javascript

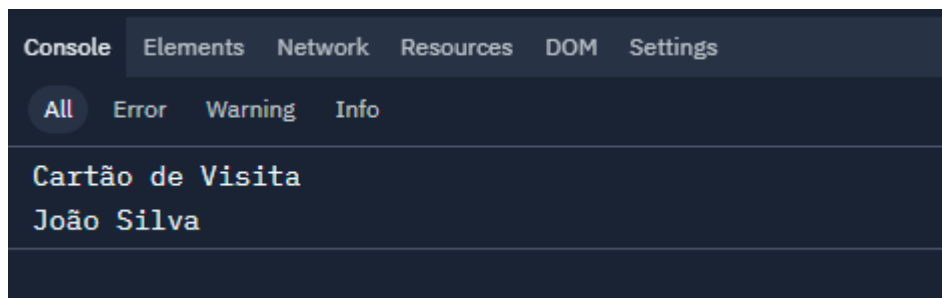
Situação Problema 4:

Vamos criar uma função chamada **cartaoDeVisita**. Essa função deverá receber dois parâmetros: um nome e um sobrenome de uma pessoa. Ao executarmos essa função, teremos a impressão do cartão contendo o nome completo da pessoa.

```
script.js x
1 function cartaoDeVisita(nome, sobrenome){
2   console.log("Cartão de Visita\n" + nome + " " + sobrenome)
3 }
4
5 cartaoDeVisita("João", "Silva")
6
```

Para essa resolução, primeiro declaramos a função com a palavra reservada **function**. Depois demos os parâmetros **nome** e **sobrenome** para identificar os dados que a função irá receber e por fim, dentro da função, concatenamos os parâmetros com outros textos utilizando o operador de concatenação **+**.

O resultado deve ser algo similar a este:



```
Console Elements Network Resources DOM Settings
All Error Warning Info
Cartão de Visita
João Silva
```

Situação Problema 5:

Os engenheiros de uma montadora estão projetando o computador de bordo de um carro. Eles precisam de uma função que possa calcular a autonomia atual do automóvel, em outras palavras, quantos quilômetros ele consegue andar com a quantidade de combustível atual.

O cálculo da autonomia se dá pela divisão da distância percorrida pela quantidade de combustível consumida. Por exemplo: um carro que gasta 8 litros a cada 100 km tem como média 12,5 km/litro.

Vamos criar uma função chamada **autonomia**. Essa função deve receber dois parâmetros:

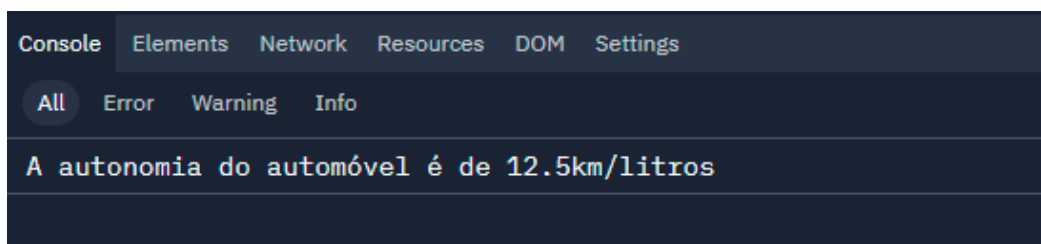
- O primeiro, que representa a quantidade em **litros** de combustível gasto;
- O segundo, que representa a **distância** percorrida pelo automóvel

A função deve retornar o valor que representa a autonomia do automóvel em km/litro.

```
script.js x
1 function autonomia (litros, distancia){
2   return distancia / litros
3 }
4
5 var resultado = autonomia (8, 100)
6
7 console.log("A autonomia do automóvel é de " + resultado + "km/litros")
8
```

Para esse problema, criamos dois parâmetros para a função, um que irá receber a quantidade de combustível em litros e outro para a distância percorrida pelo carro em quilômetros, e por fim retornamos a operação dividindo a distância pelos litros de combustível gastos.

Definimos uma variável chamada “resultado” que recebe o que vier de **retorno** da função autonomia, passando como parâmetros: 8 litros de combustível e 100 km percorridos. O resultado é algo como esse:



Esse cálculo foi realizado dentro da função e retornado para ser armazenado na variável “resultado”, e por fim, utilizamos o `console.log()` para imprimir na tela a autonomia do automóvel.

Exercícios

1. Você deve criar uma função chamada **metade** que irá receber um número como parâmetro e deve retornar a metade desse número.
2. Uma lavanderia lava roupa pelo peso. Ela cobra de seus clientes R\$ 5,00 por cada quilo de roupa suja. Atualmente, eles usam um caderno e uma calculadora para fazer o cálculo de quanto cada cliente precisa pagar.

Nosso desafio é automatizar esse cálculo, vamos lá?

Crie uma função que recebe como parâmetro o peso de roupa suja e deve retornar o valor a ser cobrado ao cliente.