# Invocação de função JavaScript

[❮ Anterior](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/js_function_parameters.asp&usg=ALkJrhhzuzz8YYe20pQ2DXPHBjosjXulfw)[Próximo ❯](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/js_function_call.asp&usg=ALkJrhiJwrZp6_RDnpz_OAB50yBbE3L3Qw)

O código dentro de uma função JavaScript será executado quando "algo" o invoca.

## Invocando uma função JavaScript

O código dentro de uma função não é executado quando a função é **definida** .

O código dentro de uma função é executado quando a função é **invocada** .

É comum usar o termo " **chamar uma função** " em vez de " **invocar uma função** ".

Também é comum dizer "chamar uma função", "iniciar uma função" ou "executar uma função".

Neste tutorial, usaremos **invocar** , porque uma função JavaScript pode ser invocada sem ser chamada.

## Invocando uma Função como Função

### Exemplo

function myFunction(a, b) {  
    return a \* b;  
}  
myFunction(10, 2);           // Will return 20

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_invoke_function&usg=ALkJrhjbfuiSnQCUiAGuSZfi4FYFxTyFyw)

A função acima não pertence a nenhum objeto. Mas, em JavaScript, há sempre um objeto global padrão.

Em HTML, o objeto global padrão é a própria página HTML, então a função acima "pertence" à página HTML.

Em um navegador, o objeto da página é a janela do navegador. A função acima torna-se automaticamente uma função de janela.

myFunction () e window.myFunction () é a mesma função:

### Exemplo

function myFunction(a, b) {  
    return a \* b;  
}  
window.myFunction(10, 2);    // Will also return 20

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_invoke_function_2&usg=ALkJrhg28Toj81p94aY_XuEJ8vdb8dqajg)

Esta é uma maneira comum de invocar uma função JavaScript, mas não uma prática muito boa.   
Variáveis, métodos ou funções globais podem facilmente criar conflitos de nomes e erros no objeto global.

## A palavra chave

Em JavaScript, o objeto chamado **isso** é o objeto que "possui" o código atual.

O valor disto, quando usado em uma função, é o objeto que "possui" a função.

Observe que **esta** não é uma variável. É uma palavra-chave. Você não pode alterar o valor **disso** .

## O Objeto Global

Quando uma função é chamada sem um objeto proprietário, o valor **disso** se torna o objeto global.

Em um navegador web, o objeto global é a janela do navegador.

Este exemplo retorna o objeto da janela como o valor **desta** :

### Exemplo

function myFunction() {  
    return this;  
}  
myFunction();                // Will return the window object

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_invoke_function_3&usg=ALkJrhii50BpQr2GvXR_M2rLrDQS9EAzkg)

Invocar uma função como uma função global, faz com que o valor **desse** objeto seja global.   
Usar o objeto da janela como uma variável pode facilmente travar seu programa.

## Invocando uma função como método

Em JavaScript, você pode definir a função como métodos de objeto.

O exemplo a seguir cria um objeto ( **myObject** ), com duas propriedades ( **firstName** e **lastName** ) e um método ( **fullName** ):

### Exemplo

var myObject = {  
    firstName:"John",  
    lastName: "Doe",  
    fullName: function () {  
        return this.firstName + " " + this.lastName;  
    }  
}  
myObject.fullName();         // Will return "John Doe"

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_invoke_method&usg=ALkJrhjOPqqes1jg6t0rsSERj7_jd2eLaQ)

O método **fullName** é uma função. A função pertence ao objeto. **myObject** é o proprietário da função.

A coisa chamada **isso** , é o objeto que "possui" o código JavaScript. Nesse caso, o valor **disso** é **myObject** .

Teste-o! Altere o método **fullName** para retornar o valor **desta** :

### Exemplo

var myObject = {  
    firstName:"John",  
    lastName: "Doe",  
    fullName: function () {  
        return this;  
    }  
}  
myObject.fullName();          // Will return [object Object] (the owner object)

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_invoke_method_2&usg=ALkJrhiVJ8qgWvhUGOB5p8rJMd9T7VSN4A)

Invocar uma função como um método de objeto, faz com que o valor **desse** objeto seja o próprio objeto.

## Invocando uma Função com um Construtor de Função

Se uma invocação de função for precedida pela **nova** palavra-chave, é uma invocação de construtor.

Parece que você cria uma nova função, mas, como as funções de JavaScript são objetos, você realmente cria um novo objeto:

### Exemplo

// This is a function constructor:  
function myFunction(arg1, arg2) {  
    this.firstName = arg1;  
    this.lastName  = arg2;  
}  
  
// This creates a new object  
var x = new myFunction("John", "Doe");  
x.firstName;                             // Will return "John"

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_invoke_constructor&usg=ALkJrhhXyrULCuzmaLyJIOV3fsw66f2Lzw)

Uma invocação de construtor cria um novo objeto. O novo objeto herda as propriedades e os métodos do seu construtor.