

Invocação de função JavaScript

< Anterior

Próximo >

O código dentro de um JavaScript `function` será executado quando "algo" o invocar.

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 **chamada** .

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 **invoke** , porque uma função JavaScript pode ser chamada sem ser chamada.

Invocando uma função como uma função

Exemplo

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

Tente você mesmo "

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, portanto, a função acima "pertence" à página HTML.

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

myFunction() and window.myFunction() is the same function:

Example

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

Try it Yourself >

This is a common way to invoke a JavaScript function, but not a very good practice. Global variables, methods, or functions can easily create name conflicts and bugs in the global object.

The **this** Keyword

In JavaScript, the thing called **this** , is the object that "owns" the current code.

The value of **this** , when used in a function, is the object that "owns" the function.

Note that **this** is not a variable. It is a keyword. You cannot change the value of **this** .

Tip: Read more about the **this** keyword at [JS this Keyword](#).

The Global Object

When a function is called without an owner object, the value of **this** becomes the global object.

In a web browser the global object is the browser window.

This example returns the window object as the value of **this** :

Example

```
let x = myFunction();           // x will be the window object  
  
function myFunction() {  
  return this;  
}
```

Try it Yourself >

Invoking a function as a global function, causes the value of **this** to be the global object. Using the window object as a variable can easily crash your program.

Invoking a Function as a Method

In JavaScript you can define functions as object methods.

The following example creates an object (**myObject**), with two properties (**firstName** and **lastName**), and a method (**fullName**):

Example

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

Try it Yourself >

The **fullName** method is a function. The function belongs to the object. **myObject** is the owner of the function.

The thing called **this** , is the object that "owns" the JavaScript code. In this case the value of **this** is **myObject**.

Test it! Change the **fullName** method to return the value of **this** :

Example

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

Try it Yourself >

Invoking a function as an object method, causes the value of **this** to be the object itself.

Invoking a Function with a Function Constructor

If a function invocation is preceded with the **new** keyword, it is a constructor invocation.

It looks like you create a new function, but since JavaScript functions are objects you actually create a new object:

Example

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

Try it Yourself >

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

A **this** palavra-chave no construtor não tem valor. O valor de **this** será o novo objeto criado quando a função for chamada.

< Anterior

Próximo >

Reportar erro

Fórum

Cerca de

Comprar

Principais tutoriais

Tutorial HTML Tutorial
CSS Tutorial
JavaScript
Como fazer Tutorial
SQL Tutorial
Python Tutorial
W3.CSS Tutorial
Bootstrap Tutorial
PHP Tutorial
Java Tutorial
C ++ Tutorial
jQuery Tutorial

Referências principais

HTML Reference
CSS Reference
JavaScript Reference
SQL Reference
Python Reference
W3.CSS Reference
Bootstrap Reference
PHP Reference
HTML Colors
Java Reference
Angular Reference
jQuery Reference

Top Examples

HTML Examples
CSS Examples
JavaScript Examples
How To Examples
SQL Examples
Python Examples
W3.CSS Examples
Bootstrap Examples
PHP Examples
Java Examples
XML Examples
jQuery Examples

Web Courses

Curso de HTML
Curso de CSS
Curso de JavaScript
Curso de front end
Curso de SQL
Curso de Python
Curso de PHP
Curso de jQuery
Curso de Java
Curso de C ++
Curso de C #
Curso de XML

Obter certificação "

COLOR PICKER



COMO NÓS



Obtenha a certificação completando um curso hoje!



iniciar

JOGO DE CÓDIGOS



Jogar um jogo