# Erros comuns do 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_best_practices.asp&usg=ALkJrhiDo8wuZin5yRpQ0vbcM8j6iWn0Rg)[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_performance.asp&usg=ALkJrhg3voC3ongNM5hGUEAicXUanfzz-Q)

Este capítulo aponta alguns erros de JavaScript comuns.

## Usando acidentalmente o Operador de Atribuição

Programas de JavaScript podem gerar resultados inesperados se um programador acidentalmente usa um operador de atribuição (=), em vez de um operador de comparação (==) em uma instrução if.

Esta declaração **if** retorna **falsa** (como esperado) porque x não é igual a 10:

var x = 0;  
if (x == 10)

[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_mistakes_equal_1&usg=ALkJrhhi4CQPUh-TbXiYXM--5REtvUg12Q)

Esta declaração **if** retorna **verdadeira** (talvez não conforme o esperado), porque 10 é verdadeira:

var x = 0;  
if (x = 10)

[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_mistakes_equal_2&usg=ALkJrhgyXHWfktkSvUxCVxp4RdXKGnIv7w)

Esta declaração **if** retorna **falsa** (talvez não conforme o esperado), porque 0 é falso:

var x = 0;  
if (x = 0)

[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_mistakes_equal_3&usg=ALkJrhjHISTVSL-pLBmxQtgpXXFZx9D1FA)

Uma atribuição sempre retorna o valor da tarefa.

## Esperando a comparação frouxa

Na comparação regular, o tipo de dados não importa. Esta declaração if retorna true:

var x = 10;  
var y = "10";  
if (x == y)

[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_mistakes_loose_1&usg=ALkJrhj0NRXDMjERwst0mJBEbrt9OPVGIg)

Em comparação rigorosa, o tipo de dados é importante. Esta declaração if retorna falsa:

var x = 10;  
var y = "10";  
if (x === y)

[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_mistakes_loose_2&usg=ALkJrhjDb28eQTg8U6IYpCG_v1SyAhb0sg)

É um erro comum esquecer que as declarações de mudança usam uma comparação rigorosa:

Este interruptor de caixa exibirá um alerta:

var x = 10;  
switch(x) {  
    case 10: alert("Hello");  
}

[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_mistakes_loose_3&usg=ALkJrhjBFb7L1E7O7OR_hGMkubEHXuRUHg)

Este interruptor de caixa não exibirá um alerta:

var x = 10;  
switch(x) {  
    case "10": alert("Hello");  
}

[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_mistakes_loose_4&usg=ALkJrhgG9NzQ21BWUFbZPtyI30yFEq7VUA)

## Adição e concatenação confusas

**A adição** é sobre a adição de **números** .

**Concatenação** é sobre como adicionar **strings** .

Em JavaScript, ambas as operações usam o mesmo operador +.

Por causa disso, adicionar um número como um número produzirá um resultado diferente de adicionar um número como uma string:

var x = 10 + 5;          // the result in x is 15  
var x = 10 + "5";        // the result in x is "105"

[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_mistakes_add_1&usg=ALkJrhi-hb-xT2g-y2c-QHyr97usdGvepw)

Ao adicionar duas variáveis, pode ser difícil antecipar o resultado:

var x = 10;  
var y = 5;  
var z = x + y;           // the result in z is 15  
  
var x = 10;  
var y = "5";  
var z = x + y;           // the result in z is "105"

[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_mistakes_add_2&usg=ALkJrhiYV_w1ehI1Ca1YOF0MRY3SnpW1eQ)

## Flutuadores de incompreensão

Todos os números em JavaScript são armazenados como **números de ponto flutuante de** 64 bits (flutuadores).

Todas as linguagens de programação, incluindo o JavaScript, têm dificuldades com valores de ponto flutuante precisos:

var x = 0.1;  
var y = 0.2;  
var z = x + y            // the result in z will not be 0.3

[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_mistakes_floats&usg=ALkJrhiqWwp7dL5DZMe0s9DITl_4T1N2yA)

Para resolver o problema acima, ajuda a multiplicar e dividir:

### Exemplo

var z = (x \* 10 + y \* 10) / 10;       // z will be 0.3

[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_mistakes_floats_ok&usg=ALkJrhhCdFtTThxcTI5fgW6lr9trUzMwAw)

## Quebrando uma Cadeia de JavaScript

O JavaScript permitirá que você quebre uma declaração em duas linhas:

### Exemplo 1

var x =  
"Hello World!";

[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_mistakes_string_1&usg=ALkJrhgWQnw-3FbnrRPHN1d2A2OZyWhReQ)

Mas, interromper uma declaração no meio de uma string não funcionará:

### Exemplo 2

var x = "Hello  
World!";

[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_mistakes_string_2&usg=ALkJrhiESSHe-yfO31TaOU2XPbl_jFzHFQ)

Você deve usar uma "barra invertida" se você deve quebrar uma declaração em uma string:

### Exemplo 3

var x = "Hello \  
World!";

[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_mistakes_string_3&usg=ALkJrhgog0Rcsthonq8O91np1VsHx52TIw)

## Ponto-e-malha

Por causa de um ponto e vírgula mal colocado, esse bloco de código será executado independentemente do valor de x:

if (x == 19);  
{  
    // code block    
}

[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_mistakes_semicolon&usg=ALkJrhh2uiSYBADzL--Due4BOJbm4u972A)

## Quebrando uma declaração de retorno

É um comportamento de JavaScript padrão para fechar uma declaração automaticamente no final de uma linha.

Por causa disso, esses dois exemplos retornarão o mesmo resultado:

### Exemplo 1

function myFunction(a) {  
    var power = 10    
    return a \* power  
}

[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_mistakes_return_1&usg=ALkJrhjO1Q610pb_eq_UT6SPwZm-SazPNw)

### Exemplo 2

function myFunction(a) {  
    var power = 10;  
    return a \* power;  
}

[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_mistakes_return_2&usg=ALkJrhj5dKy_C6tu7FAKS4_SeLBVzDKSXw)

O JavaScript também permitirá que você quebre uma declaração em duas linhas.

Por causa disso, o exemplo 3 também retornará o mesmo resultado:

### Exemplo 3

function myFunction(a) {  
    var  
    power = 10;    
    return a \* power;  
}

[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_mistakes_return_3&usg=ALkJrhjDvpy3wCn31Kt11RRmr16EJLl3MQ)

Mas, o que acontecerá se você quebrar a declaração de retorno em duas linhas como esta:

### Exemplo 4

function myFunction(a) {  
    var  
    power = 10;    
    return  
    a \* power;  
}

[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_mistakes_return_4&usg=ALkJrhg0CbzUkyZlIAq_nztzuGQkdMEoSw)

A função retornará indefinida!

Por quê? Porque o JavaScript pensa que você quis dizer:

### Exemplo 5

function myFunction(a) {  
    var  
    power = 10;    
    return;  
    a \* power;  
}

[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_mistakes_return_5&usg=ALkJrhjrRWZxQIBCCfHkvdO1O8sfS4y9pQ)

## Explicação

Se uma declaração estiver incompleta, como:

var

O JavaScript tentará completar a declaração lendo a próxima linha:

power = 10;

Mas, uma vez que esta declaração está completa:

return

O JavaScript irá fechá-lo automaticamente assim:

return;

Isso acontece porque as instruções de fechamento (final) com ponto e vírgula são opcionais em JavaScript.

O JavaScript encerrará a declaração de retorno no final da linha, porque é uma declaração completa.

Nunca quebre uma declaração de retorno.

## Acessando matrizes com índices nomeados

Muitas linguagens de programação suportam arrays com índices nomeados.

Arrays com índices nomeados são chamados matrizes associativas (ou hashes).

O JavaScript **não** suporta arrays com índices nomeados.

Em JavaScript, os **arrays** usam **índices numerados** :

### Exemplo

var person = [];  
person[0] = "John";  
person[1] = "Doe";  
person[2] = 46;  
var x = person.length;         // person.length will return 3  
var y = person[0];             // person[0] 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_array_associative_1&usg=ALkJrhjugd7ZST_SNKSOVWn1Iq3GuHqViA)

Em JavaScript, os **objetos** usam **índices nomeados** .

Se você usar um índice nomeado, ao acessar uma matriz, o JavaScript redefinirá a matriz para um objeto padrão.

Após a redefinição automática, os métodos e propriedades da matriz produzirão resultados indefinidos ou incorretos:

### Exemplo:

var person = [];  
person["firstName"] = "John";  
person["lastName"] = "Doe";  
person["age"] = 46;  
var x = person.length;         // person.length will return 0  
var y = person[0];             // person[0] will return undefined

[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_array_associative_2&usg=ALkJrhgOVPed5H5PCkMluTt7end65odVvA)

## Definições finais com uma vírgula

As vírgulas na definição do objeto e da matriz são legais no ECMAScript 5.

### Exemplo de objeto:

person = {firstName:"John", lastName:"Doe", age:46,}

### Exemplo de Array:

points = [40, 100, 1, 5, 25, 10,];

ATENÇÃO !!

O Internet Explorer 8 falhará.

JSON não permite as vírgulas à direita.

### JSON:

person = {firstName:"John", lastName:"Doe", age:46}

### JSON:

points = [40, 100, 1, 5, 25, 10];

## Indeterminado não é nulo

Com JavaScript, **null** é para objetos, **indefinido** é para variáveis, propriedades e métodos.

Para ser nulo, um objeto deve ser definido, caso contrário, ele será indefinido.

Se você quer testar se um objeto existe, isso provocará um erro se o objeto for indefinido:

### Incorreta:

if (myObj !== null && typeof myObj !== "undefined")

Por isso, você deve testar typeof () primeiro:

### Corrigir:

if (typeof myObj !== "undefined" && myObj !== null)

## Esperando alcance do nível de bloco

O JavaScript **não** cria um novo escopo para cada bloco de código.

É verdade em muitas linguagens de programação, mas **não é verdade** em JavaScript.

Este código exibirá o valor de i (10), mesmo FORA do bloco de loop for:

### Exemplo

for (var i = 0; i < 10; i++) {  
    // some code  
}  
return i;