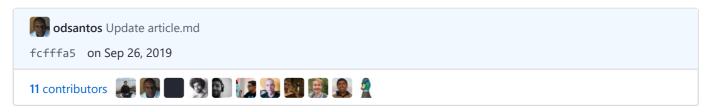
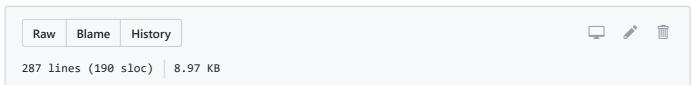
Tree: fcfffa5aad ▼

Find file

Copy path

#### pt.javascript.info / 1-js / 02-first-steps / 16-javascript-specials / article.md





# Especiais em JavaScript

Este capítulo brevemente revê as funcionalidades de JavaScript que aprendemos até agora, dando atenção especial a momentos subtis.

# Estrutura do código

Instruções são delimitadas por ponto-e-vírgula:

```
alert('Olá'); alert('Mundo');
```

Geralmente, uma quebra de linha é também tratada como um delimitador, assim o acima também funcionaria como:

```
alert('Olá')
alert('Mundo')
```

Isto chama-se "inserção automática de ponto-e-vírgula". Por vezes, não funciona, a exemplo:

```
alert("Haverá um erro depois desta mensagem")
[1, 2].forEach(alert)
```

A maioria dos guias de estilo-de-código concorda que deveríamos colocar um pontoe-vírgula após cada instrução. Pontos-e-vírgula não são necessários depois de blocos de código {...} e outras construções sintáticas que os utilizem, como laços (loops):

```
function f() {
} // nenhum ponto-e-vírgula necessário depois da declaração de função
for(;;) {
} // nenhum ponto-e-vírgula necessário depois do laço
```

...Mas mesmo que coloquemos um ponto-e-vírgula "extra" algures, isso não é um erro. Será ignorado.

Mais em: info:structure.

#### Modo strict

Para activar completamente todas as funcionalidades do JavaScript moderno, devemos começar programas (scripts) com "use strict".

```
'use strict';
```

A diretiva deve estar no topo de um script ou no início de uma função.

Sem "use strict", tudo ainda funciona, mas algumas funcionalidades comportam-se à forma antiga, no modo "compatível". Nós geralmente preferiríamos o comportamento moderno.

Algumas funcionalidades modernas da linguagem (como classes que estudaremos no futuro) ativam o modo strict implícitamente.

Mais em: info:strict-mode.

#### Variáveis

Podem ser declaradas usando:

- let
- const (constante, não pode ser alterada)
- var (estilo-antigo, veremos mais tarde)

O nome de uma varável pode incluir:

- Letras e dígitos, mas o primeiro caráter não pode ser um dígito.
- Carateres \$ e \_ são normais, on par às letras.
- Alfabetos não-latinos e hieróglifos também são permitidos, mas geralmente não utilizados.

Variáveis obtêm tipos dinâmicamente. Elas podem armazenar qualquer valor:

```
let x = 5;
x = "John";
```

Existem 7 tipos de dados:

- number para números, quer inteiros (*integer*) como em ponto-flutuante (*floating-point*),
- string para cadeias-de-carateres (strings),
- boolean para valores lógicos: true/false (verdadeiro/falso),
- null -- um tipo de dados com apenas um valor null , significando "vazio" ou "não existe",
- undefined -- um tipo de dados com apenas um valor undefined , significando "não atribuido",
- object and symbol -- para estruturas de dados complexas e identificadores únicos, que ainda não aprendemos.

O operador typeof retorna o tipo de um valor, com duas exceções:

```
typeof null == "object" // erro da linguagem
typeof function(){} == "function" // funções são tratadas especialmente
```

Mais em: info:variables e info:types.

# Interação

Estamos a utilizar um navegador (*browser*) como ambiente de trabalho, assim funções básicas de *UI* (Interface/interação com o Utilizador) serão:

```
prompt(question, [default]) : Faz uma question, e retorna o que o visitante inseriu
ou null se a pessoa "cancelou".
```

confirm(question) : Faz uma question e sugere que a pessoa escolha entre *Ok* e *Cancel*. A escolha é retornada como true/false (verdadeiro/falso).

```
alert(message) : Mostra message .
```

Todas estas funções são modal (modais), elas suspendem o código em execução e impedem o visitante de interagir com a página até obterem uma resposta.

Por exemplo:

```
let userName = prompt("Como se chama?", "Alice");
let isTeaWanted = confirm("Quer chá?");
alert( "Visitante: " + userName ); // Alice
alert( "Chá aceite: " + isTeaWanted ); // true
```

Mais em: info:alert-prompt-confirm.

# **Operadores**

JavaScript suporta os seguintes operadores:

Aritmétricos : Regulares: \* + - / , e também % para o resto de uma divisão inteira, e \*\* para a potência de um número.

```
O operador binário mais `+` concatena *strings*. E se um dos operandos for uma
*string*, o outro também é convertido para *string*:
```js run
alert( '1' + 2 ); // '12', *string*
alert( 1 + '2' ); // '12', *string*
```

De atribuição: Existe uma atribuição simples: a = b e combinadas como a \*= 2.

Bit-a-bit: Operadores bit-a-bit (bitwise operators) trabalham com números inteiros ao nível do bit: veja em docs quando são necessários.

Ternários: O único operador com três parâmetros: condition? resultA: resultB. Se condition for verdadeira, retorna resultA, senão resultB.

Lógicos: Os lógicos AND (E) && e OR (OU) || executam evaluação em curto-circuito (short-circuit evaluation) e depois retornam o valor em que pararam. O lógico NOT (NÃO) ! converte o operando para o tipo boleano e retorna o valor inverso desse boleano.

Comparisons : O de verificação de igualdade == para valores de tipos diferentes, os converte para números (exceto null e undefined que igualam-se a si próprios, e a nada mais); assim os seguintes são similares:

```
```js run
alert( 0 == false ); // verdadeiro
alert( 0 == '' ); // verdadeiro
Em outras comparações também são convertidos para números.
O operador de igualdade exata (*strict equality*) `===` não executa a
conversão; para ele com tipos diferentes sempre são valores diferentes, assim:
Os valores `null` e `undefined` são especiais: eles são iguais `==` a si
próprios e a mais nenhum outro.
Comparações maior/menor, comparam *strings* (cadeias-de-carateres) caráter-
por-caráter, outros tipos são convertidos para número.
```

Outros operadores: Existem mais uns outros poucos, como o operador vírgula.

Mais em: info:operators, info:comparison, info:logical-operators.

### Laços

Vimos 3 tipos de laços:

```
// 1
while (condition) {
}
// 2
do {
} while (condition);
for(let i = 0; i < 10; i++) {
  . . .
}
```

- A variável declarada no laço for(let...) apenas é visível dentro do laço. Mas também podemos omitir let e reusar uma variável já existente.
- As diretivas break/continue permitem sair completamente-do-laço/da-actualiteração. Use etiquetas (labels) para sair (break) de laços aninhados (nested loops)..

Detalhes em: info:while-for.

Mais adiante estudaremos outros tipos de laços para lidar com objetos.

## A construção "switch"

A construção "switch" permite substituir múltiplas verificações if . Ela emprega === (igualdade exata - strict equality) nas comparações.

Por exemplo:

```
let age = prompt('Que idade tem?', 18);
switch (age) {
  case 18:
   alert("Não funcionará"); // o resultado de *prompt* é uma *string*, não um nú
  case "18":
   alert("Isto funciona!");
   break;
 default:
   alert("Qualquer valor não igual aos dos 'case' acima");
}
```

Detalhes em: info:switch.

# **Funções**

Vimos três formas de criar uma função em JavaScript:

1. Declaração de função: a função no fluxo principal do código

```
function sum(a, b) {
 let result = a + b;
  return result;
}
```

2. Expressão de função: a função no contexto de uma expressão

```
let sum = function(a, b) {
 let result = a + b;
  return result;
}
```

Expressões de função podem ter um nome, como sum = function name(a, b), mas esse name apenas é visível dentro da função.

3. Arrow functions:

```
// expressão no lado direito
let sum = (a, b) => a + b;
// ou em sintaxe multi-linha com { ... }, aqui precisa de return
let sum = (a, b) => {
 // ...
 return a + b;
}
// sem argumentos
let sayHi = () => alert("01á");
// com um único argumento
let double = n => n * 2;
```

- Funções podem ter variáveis locais: aquelas declaradas no seu corpo. Tais variáveis apenas são visíveis dentro da função.
- Parâmetros podem ter valores por defeito: function sum(a = 1, b = 2) {...}.
- Funções sempre retornam algo. Se não houver uma instrução return, então o resultado é undefined.

Declaração de Função	Expressão de Função
visível em todo o bloco de código	criada quando a execução a alcança
-	pode ter um nome, visível apenas dentro da função

Mais: veja info:function-basics, info:function-expressions-arrows.

#### Mais adiante

Esta foi uma breve lista de funcionalidades de JavaScript. Até agora apenas estudámos o básico. Mais adiante no tutorial encontrará mais funcionalidades especiais e avançadas de JavaScript.