Âmbito JavaScript

❮ Anterior Próximo ❯

O escopo determina a acessibilidade (visibilidade) das variáveis.

JavaScript tem 3 tipos de escopo:

- Escopo do bloco
- Escopo de função
- Âmbito global

Escopo do Bloco

Antes do ES6 (2015), JavaScript tinha apenas **escopo global** e **escopo de função** .

ES6 introduziu duas novas palavras-chave JavaScript importantes: `let` e `const`.

Essas duas palavras-chave fornecem **Block Scope** em JavaScript.

Variables declared inside a { } block cannot be accessed from outside the block:

Example

```
{
  let x = 2;
}
// x can NOT be used here
```

Variables declared with the `var` keyword can NOT have block scope.

Variables declared inside a { } block can be accessed from outside the block.

Example

```
{
  var x = 2;
}
// x CAN be used here
```

Local Scope

Variables declared within a JavaScript function, become **LOCAL** to the function.

Example

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Try it Yourself »

Local variables have **Function Scope**:

They can only be accessed from within the function.

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

Function Scope

JavaScript has function scope: Each function creates a new scope.

Variables defined inside a function are not accessible (visible) from outside the function.

Variables declared with `var`, `let` and `const` are quite similar when declared inside a function.

They both have **Function Scope**:

```
function myFunction() {
  var carName = "Volvo";   // Function Scope
}
```

```
function myFunction() {
  let carName = "Volvo";   // Function Scope
}
```

```
function myFunction() {
  const carName = "Volvo";   // Function Scope
}
```

Global JavaScript Variables

A variable declared outside a function, becomes **GLOBAL**.

Example

```
let carName = "Volvo";
// code here can use carName

function myFunction() {
// code here can also use carName
}
```

Try it Yourself »

A global variable has **Global Scope**:

All scripts and functions on a web page can access it.

Global Scope

Variables declared **Globally** (outside any function) have **Global Scope**.

Global variables can be accessed from anywhere in a JavaScript program.

Variables declared with `var`, `let` and `const` are quite similar when declared outside a block.

They both have **Global Scope**:

```
var x = 2;       // Global scope
```

```
let x = 2;       // Global scope
```

```
const x = 2;       // Global scope
```

JavaScript Variables

In JavaScript, objects and functions are also variables.

Scope determines the accessibility of variables, objects, and functions from different parts of the code.

Automatically Global

If you assign a value to a variable that has not been declared, it will automatically become a **GLOBAL** variable.

This code example will declare a global variable `carName`, even if the value is assigned inside a function.

Example

```
myFunction();

// code here can use carName

function myFunction() {
  carName = "Volvo";
}
```

Try it Yourself »

Strict Mode

All modern browsers support running JavaScript in "Strict Mode".

You will learn more about how to use strict mode in a later chapter of this tutorial.

In "Strict Mode", undeclared variables are not automatically global.

Global Variables in HTML

With JavaScript, the global scope is the JavaScript environment.

In HTML, the global scope is the window object.

Global variables defined with the `var` keyword belong to the window object:

Example

```
var carName = "Volvo";
// code here can use window.carName
```

Try it Yourself »

Global variables defined with the `let` keyword do not belong to the window object:

Example

```
let carName = "Volvo";
// code here can not use window.carName
```

Try it Yourself »

Warning

Do NOT create global variables unless you intend to.

Your global variables (or functions) can overwrite window variables (or functions).
Any function, including the window object, can overwrite your global variables and functions.

The Lifetime of JavaScript Variables

The lifetime of a JavaScript variable starts when it is declared.

Variáveis de função (locais) são excluídas quando a função é concluída.

Em um navegador da web, as variáveis globais são excluídas quando você fecha a janela do navegador (ou guia).

Argumentos de função

Argumentos de função (parâmetros) funcionam como variáveis locais dentro de funções.

❮ Anterior Próximo ❯