Programação Estruturada

Aula 03 - Variáveis e Constantes

Videoaula 02 - *Hoisting*, *Strict Mode* e `Let`

Videoaula 02 - *Hoisting*, *Strict Mode* e `Let`

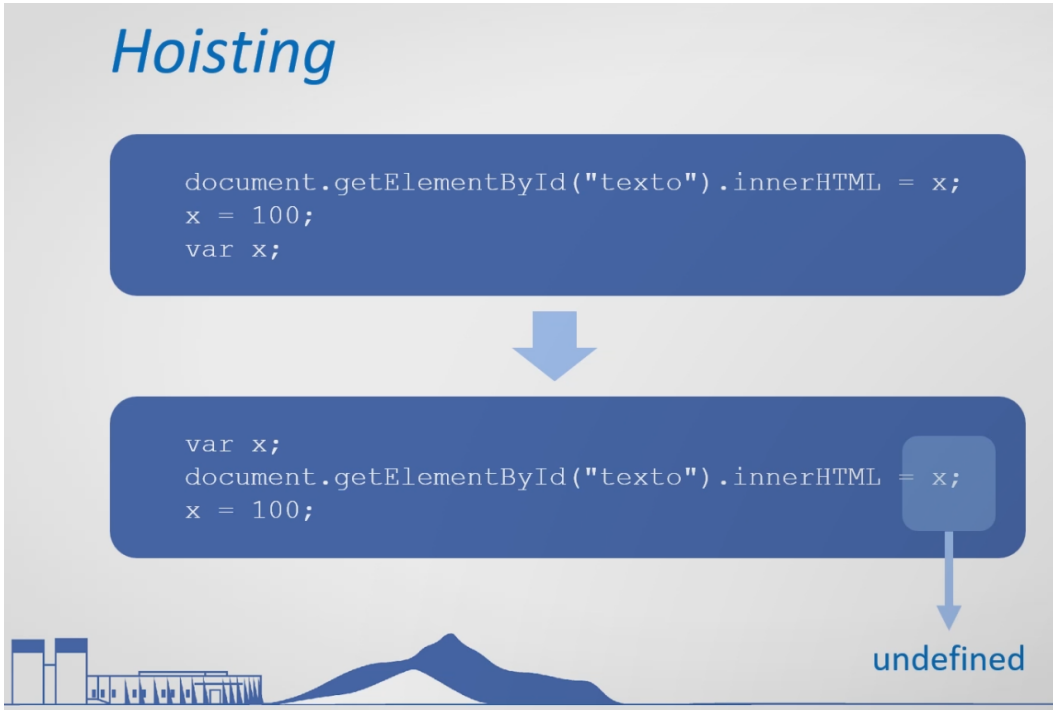
Hoisting e Strict Mode

Agora, você aprenderá um conceito importante de JavaScript que tem consequência direta no comportamento de seus programas: o conceito de *Hoisting*, ou, em português, **içamento ou elevação**. Isso porque *Hoisting* é o comportamento padrão de JavaScript que move as declarações das variáveis para o topo do programa.

Por exemplo, suponha que temos um programa que atribui 100 a uma variável `x`, depois altere o conteúdo de um texto da página HTML usando a variável `x`, e só depois declare a variável `x`. Esse programa tem o mesmo comportamento de um programa que declara `x`, atribui o valor `100` a `x` e, só depois, altera o texto da página HTML usando a variável `x`.

No entanto, é importante ressaltar que JavaScript leva apenas as declarações para o topo. As inicializações não são movidas. Vamos supor um exemplo que altere o conteúdo de um texto da página HTML usando a variável `x`, depois atribui 100 a essa variável e, finalmente, declare a variável `x`. Esse programa tem o mesmo comportamento de um programa que declara `x`, altera o conteúdo de um texto da página HTML usando a variável `x`, e só depois atribui 100 a essa variável. Ou seja, no texto da página HTML receberá o valor `undefined` (Figura 1).

Figura 1 - Conceito de Hoisting



Vejamos rapidamente esses exemplos na prática.

Nesse exemplo, na linha 16, temos a atribuição de 100 à variável `x`, foi atribuído ao campo texto o valor da variável `x` (linha 17), e só depois é que foi declarada a variável `x` (linha 18), veja a seguir (código 1).

Código 1 - 03_2 Hoisting.html

```
1  <html>
2
3    <head>
4      <meta charset="UTF-8" />
5      <title>Programação Estruturada - Aula 03</title>
6    </head>
7
8    <body>
9      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>
10
11     <h1>Hoisting</h1>
12
13     <p id="texto"></p>
14
15     <script>
16       x = 100;
```

```
17          document.getElementById("texto").innerHTML = x;
18          var x;
19        </script>
20
21    </body>
22
23 </html>
24
```

Note que esta página exibe o valor 100 (Figura 2). Então, deu tudo certo o que aconteceu, o JavaScript fez o hoisting dessa declaração de variável e, de fato, essa declaração foi trazida para a linha 16 (Figura 3), e o comportamento foi exatamente o que você pode ver na página ao recarregar: a exibição do número 100, do valor 100, que é o valor da variável `x` e que continua aparecendo (Figura 2).

Figura 2 - Comportamento do Hoisting em JavaScript

Figura 3 - Conteúdo do arquivo 03_2 Hoisting.html

```
1 var x;
2 x = 100;
3 document.getElementById("texto").innerHTML = x;
4
```

De fato, muitos programadores desconhecem essa característica de JavaScript. Além disso, o uso de variáveis antes de suas declarações dificulta o entendimento do programa, facilitando a inserção de erros. Por isso, recomendamos declarar todas as variáveis no início de cada escopo do programa.

Uma das opções para forçar esse comportamento é usar o modo rigoroso de JavaScript, chamado em inglês de *strict mode*. Quando executado nesse modo, o JavaScript não permitirá que alguns erros silenciosos de JavaScript sejam apontados pelo interpretador. Um desses, digamos, “erros”, é o uso de variáveis não declaradas.

Você pode usar o modo rigoroso para todo o programa ou para blocos, os quais detalharemos ainda nesta aula. Por enquanto, vamos apenas ver um exemplo de uso do modo rigoroso de JavaScript.

Neste exemplo, vemos na primeira linha do trecho JavaScript, na linha 16, a atribuição do comando para usar o modo rigoroso de JavaScript. Então, com o texto `"use strict"`; eu dou o comando para iniciar o uso desse modo rigoroso (código 2).

Código 2 - 03_3 Strict Mode.html

```
1  <html>
2
3    <head>
4      <meta charset="UTF-8" />
5      <title>Programação Estruturada - Aula 03</title>
6    </head>
7
8    <body>
9      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>
10
11     <h1>Usando o Modo Rigoroso</h1>
12
13     <p id="texto"></p>
14
15     <script>
16       "use strict";
17       var a;
18       a = 10;
19       document.getElementById("texto").innerHTML = a;
20     </script>
21
22   </body>
23
24 </html>
25
```

Em seguida, faço a declaração da variável `a`, atribuo 10 a ela e uso essa variável para alterar o valor do campo texto. Como esperado, a página exibe o valor 10 (Figura 4).

Figura 4 - Usando o modo rigoroso

Agora, vamos fazer um teste: retirar a declaração da variável `a` (linha 17), comentar/remover essa declaração (Figura 5), e note que ao recarregar a página a exibição do 10 some (Figura 3).

Figura 5 - Conteúdo do arquivo 03_3 Strict Mode.html

```
1 //var a;
2 a = 10;
3 document.getElementById("texto").innerHTML = a;
4
```

Figura 6 - Declaração da variável `a`

Usando o Modo Rigoroso

Você ainda vai aprender a trabalhar com essa ferramenta de depuração dos navegadores, mas se apertar o F12, usando o Google Chrome, poderá ver na exibição do console.

Ainda entrarei em detalhes sobre isso em outras aulas, mas veja que ele diz exatamente que *"a is not defined"*, ou seja, ele está dizendo que a não foi definido (Figura 7). Isso porque, como vimos, tiramos a declaração dessa variável e estamos usando o modo rigoroso; se novamente voltarmos com a declaração da variável `a`, e recarregarmos a página, o erro some e o número 10 voltará a ser exibido (Figura 8).

Figura 7 - Declaração da variável a não definida

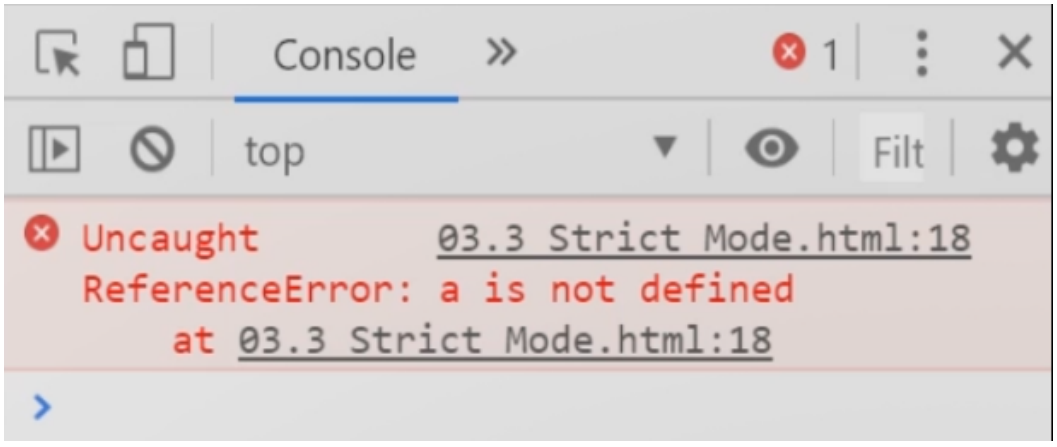
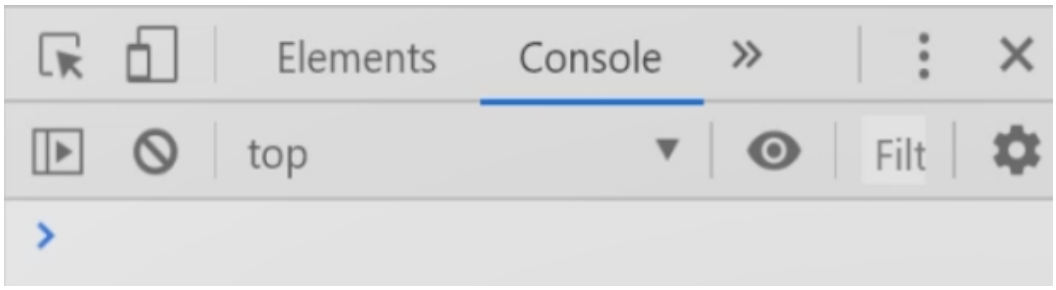


Figura 8 - Declaração da variável a definida



Então, lembre-se: sempre que usar o modo rigoroso de JavaScript, você será obrigado a declarar as variáveis antes de usá-las.

Atenção

No anexo C da Especificação do JavaScript 2018, você encontra a lista de todas as restrições impostas quando usar o modo rigoroso.

https://www.ecma-international.org/ecma-262/9.0/index.html#sec-strict-mode-of-ecmascript

Existem, porém, duas declarações que não são levadas para o topo pelo *hoisting* de JavaScript. Declarações de variáveis usando a palavra-chave `let` e declarações de constantes usando a palavra-chave `const`. Vamos conhecer cada uma delas?

No resto desta videoaula, iremos conhecer o uso da palavra-chave `let` e vamos deixar para a próxima videoaula a apresentação da declaração de constantes.

Declaração de Variáveis Usando `Let`

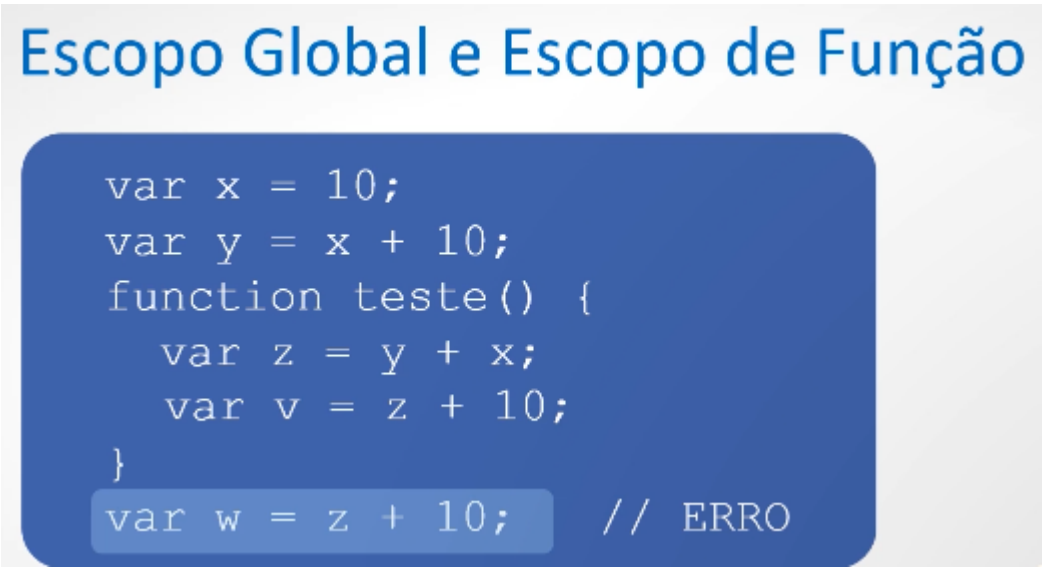
Antes das palavras-chave `let` e `const`, que foram introduzidas na versão ECMAScript 2015, JavaScript só tinha dois tipos de escopo: **escopo global** e **escopo de função**.

As variáveis declaradas fora de qualquer função são consideradas como declaradas globalmente e têm o que chamamos de escopo global, ou seja, podem ser referenciadas em todo o programa. Por outro lado, variáveis declaradas dentro de uma função são consideradas como declaradas localmente e só podem ser referenciadas dentro das funções onde elas são declaradas.

Neste slide (Figura 9), vemos que a declaração das variáveis `x` e `y` são feitas fora da função `teste`. Dessa forma, elas podem ser referenciadas em todo o programa. Como podemos ver, `x` é referenciado tanto na atribuição de `y`, ou seja, fora da função `teste`, como também na atribuição de `z`, dentro do corpo dessa função. A variável `y` é referenciada apenas dentro do corpo da função.

Por fim, temos a variável `z`, que é declarada dentro do corpo da função `teste`. Por isso, ela pode ser referenciada apenas dentro da função, como, por exemplo, na atribuição de `v`. Porém, ela não pode ser referenciada fora da função, como na atribuição de `w` no slide.

Figura 9 - Tipos de escopo JavaScript



A partir da ECMAScript 2015, JavaScript apresentou o **escopo de bloco**, o qual é delimitado usando { (abre chave) e } (fecha chave). Porém, variáveis declaradas usando a palavra-chave `var` não podem ter escopo de bloco. Para forçarmos esse tipo de escopo a uma variável, temos que usar a palavra-chave `let`. Vejamos um exemplo.

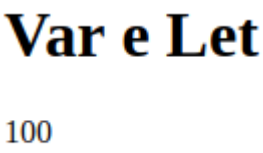
Nesse exemplo (Código 3), foi definido um bloco com abre chave e fecha chave, entre as linhas 16 e 19 e, dentro do bloco, foi declarada a variável `x` usando `var`.

Código 3 - 03_4 Var e Let.html

```
1  <html>
2
3    <head>
4      <meta charset="UTF-8" />
5      <title>Programação Estruturada - Aula 03</title>
6    </head>
7
8    <body>
9      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>
10
11     <h1>Var e Let</h1>
12
13     <p id="texto"></p>
14
15     <script>
16       {
17         var x = 100;
18         let y = 100;
19       }
20       document.getElementById("texto").innerHTML = x;
21     </script>
22
23   </body>
24
25 </html>
26
```

A variável `x` está recebendo o valor 100, declarando `let` para a variável `y`, que também está recebendo o valor 100, e fora do bloco, na linha 20, foi alterado o campo texto para receber o valor da variável `x`. Abrindo a página HTML, de fato, é exibido o valor da variável `x`, que é 100 (Figura 10).

Figura 10 - Tipos de escopo `var` e `let`



Agora, voltando ao exemplo, vou trocar `x` pelo `y` (linha 20), lembrando: o `y` está sendo declarado dentro de um bloco e a alteração do campo texto está fora desse bloco, e declaramos a variável `y` (linha 18) usando o `let` (Figura 11).

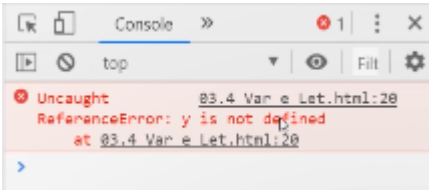
Figura 11 - Conteúdo do arquivo 03_4 Var e Let.html

```
1     var x = 100;
2     let y = 100;
3  }
4  document.getElementById("texto").innerHTML = y;
5
```

Então, se você acessar a página HTML e recarregá-la, notará que o valor para de ser exibido (Figura 12), isso porque temos um erro, estamos usando uma variável que foi declarada com `let` dentro do bloco e estamos usando-a fora do bloco.

Se abrirmos a ferramenta de depuração haverá uma mensagem de erro dizendo que "*y is not defined*", ou seja, `y` não está definido, a variável não está enxergando a declaração de `y` (Figura 12).

Figura 12 - Tipos de escopo `var` e `let`



Atenção

Lembre-se: se você declarar a variável dentro de um bloco, usando `let`, ela só pode ser vista dentro daquele bloco.

A redeclaração de variáveis usando apenas a palavra-chave `var` pode causar confusões. Essas confusões podem ser resolvidas usando a palavra-chave `let`. No entanto, o uso da palavra-chave `let` introduz restrições nas redeclarações de variáveis. De maneira bastante resumida, variáveis declaradas utilizando `let` não podem ser redeclaradas (com `var` ou `let`) no mesmo escopo. Vejamos alguns exemplos.

Neste exemplo (Código 4), vamos brincar um pouco com as declarações usando `var` e `let`. Na linha 16, pode ser vista a declaração da variável `a`, recebendo o valor `true`, e, nas linhas 17 a 19, foi criado um bloco e declarada novamente a variável `a` usando o `var`, dessa vez recebendo o valor `false` no final (linha 20); vamos sempre alterar o valor do campo texto, chamado `"texto"`, para receber o valor de `a`.

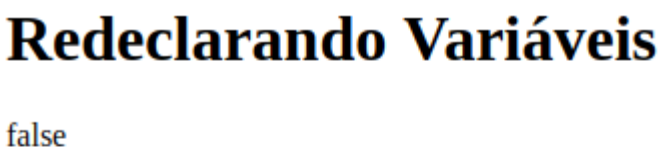
Código 4 - 03_5 Redeclarando Variáveis.html

```
1  <html>
2
3    <head>
4      <meta charset="UTF-8" />
5      <title>Programação Estruturada - Aula 03</title>
6    </head>
7
8    <body>
9      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>
10
11     <h1>Redeclarando Variáveis</h1>
12
13     <p id="texto"></p>
14
15     <script>
16       var a = true;
17       {
18         var a = false;
```

```
19          }
20          document.getElementById("texto").innerHTML = a;
21      </script>
22
23    </body>
24
25 </html>
26
```

Ao recarregar, você verá que o valor exibido foi `false`, ou seja, o que valeu foi, de fato, essa última declaração da variável `a` (Figura 13).

Figura 13 - Redeclarando Variáveis



Agora, vamos fazer o seguinte: suponha que, dentro do bloco (linha 18), a declaração fosse feita usando o `let` (Figura 14) e, ao recarregar a página HTML (Figura 15), o valor exibido foi `true`, isso aconteceu porque essa declaração do `let` dentro do bloco é considerada “local ao bloco” porque foi usado o `let`, e você pode ver fora do bloco o valor da variável `a` com o valor `true`, que é a variável global que foi declarada na linha 16.

Figura 14 - Conteúdo do arquivo 03_5 Redeclarando Variáveis.html

```
1 <script>
2   var a = true;
```

```
3    {
4       let a = false;
5    }
6    document.getElementById("texto").innerHTML = a;
7 </script>
8
```

Figura 15 - Redeclaração de variáveis

Agora, vamos tentar acrescentar uma nova declaração da variável `a`, recebendo o valor `true`, por exemplo, dentro do bloco (linha 19). Então, temos agora dentro do bloco duas declarações da variável `a` usando o `let` (Figura 16).

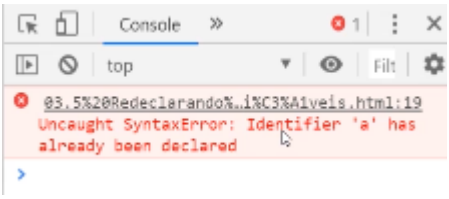
Figura 16 - Conteúdo do arquivo 03_5 Redeclarando Variáveis.html

```
1  <script>
2     var a = true;
3     {
4        let a = false;
5        let a = true;
6     }
7     document.getElementById("texto").innerHTML = a;
8  </script>
9
```

Voltando para a página, se você recarregar verá que o número parou de ser exibido (Figura 17) e, para investigar o que está acontecendo, como vimos anteriormente, usaremos a ferramenta de depuração.

Só lembrando que você verá melhor essa ferramenta em outra aula. Nesse exemplo, a ferramenta está afirmando que "*identifier 'a' has already been declared*", ou seja, o que ela está dizendo é que o identificador `a` já foi declarado.

Figura 17 - Redeclaração de variáveis



Então, você pode ver que temos, de fato, um erro por haver duas declarações da variável `a` usando a palavra-chave `let` (linhas 18 e 19).

A mesma coisa acontece se, por acaso, eu trocar a primeira declaração (linha 18) para `var`, então ficaremos com duas declarações agora da variável `a` dentro do bloco, entre as linhas 17 e 20 (Figura 18).

A variável `a` usando a palavra-chave var recebendo o valor `false`, a variável `a` novamente declarada, só que agora usando a palavra-chave `let`, e, se voltarmos pra página, o erro que vai ser exibido é o mesmo (Figura 17). Isso porque, uma vez que nós declaramos usando `let`, não podemos ter redeclaração daquela variável dentro do bloco.

Figura 18 - Conteúdo do arquivo 03_5 Redeclarando Variáveis.html

```
1 <script>
2    var a = true;
3    {
4       var a = false;
5       let a = true;
6    }
7    document.getElementById("texto").innerHTML = a;
8 </script>
9
```

Vamos agora remover essa declaração mais interna (linha 19) e colocar uma declaração mais externa usando o `let` depois do bloco (linha 20). E o que esperamos ao colocar o valor `false` fora do bloco, na linha 20, é uma nova declaração da variável `a` usando a palavra-chave `let` recebendo `false` e, se voltarmos lá para a página, veremos que o erro continua sendo exibido (Figura 17), dessa vez porque no bloco mais externo temos novamente uma redeclaração da variável `a` onde uma das declarações usa a palavra-chave `let` (linha 20).

Atenção

Então, lembre-se: sempre que tivermos a declaração de uma variável usando `let`, não poderemos fazer uma redeclaração dessa variável, seja usando `var` ou seja usando `let`.

É importante ressaltar que o `let` não é totalmente suportado pelo Internet Explorer 11 ou anterior. No slide (Figura 19), você pode ver as primeiras versões de cada navegador que deu suporte ao `let`. Nela, vemos que o primeiro navegador a dar esse suporte foi o Firefox 44, em janeiro de 2015 e que o último a dar esse suporte foi o Safari 11, em setembro de 2017.

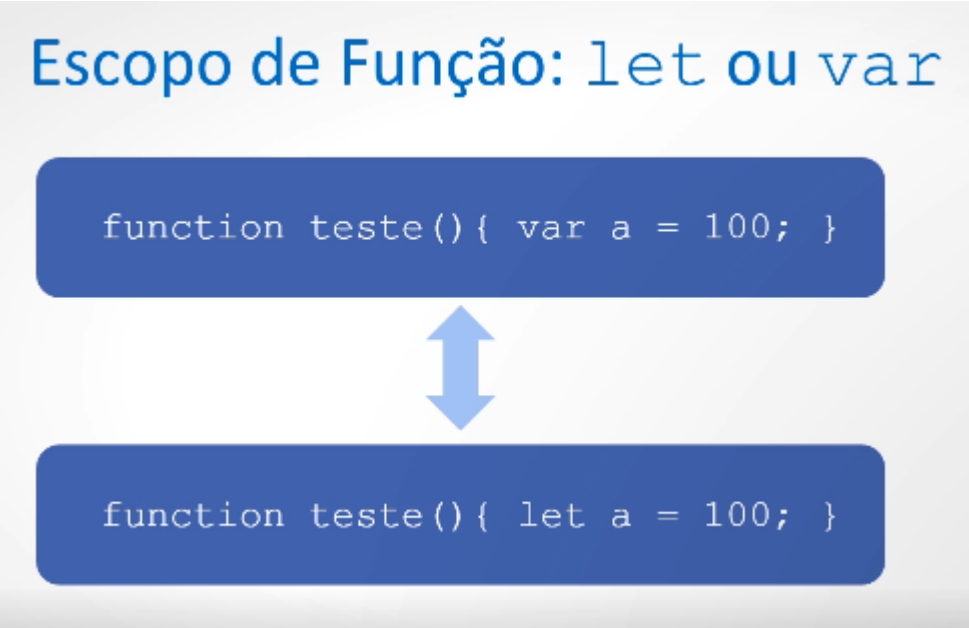
Figura 19 - Suporte dos navegadores

Variáveis declaradas com `let` ou `var` têm o mesmo escopo global se declaradas fora de um bloco. Por exemplo, em ambas as declarações do slide (Figura 20), a variável `a` tem um escopo global e pode ser referenciada em todo o programa JavaScript.

Figura 20 - Escopo Global

Da mesma forma, variáveis declaradas com `let` ou `var` têm o mesmo escopo de função se declaradas dentro de uma função. Por exemplo, em ambas as declarações do slide (Figura 21), a variável `a` tem um escopo de função e só pode ser referenciada dentro de uma função.

Figura 21 - Escopo de Função



Concluímos esta videoaula sobre o uso de *Hoisting*, *Strict Mode* e da declaração de variáveis usando `let`. Na próxima videoaula, você vai aprender como declarar e usar constantes. Até lá!