Para saber mais: selecionando elementos

1 - O DOM (Document Object Model) e o JavaScript, juntos, possuem grande poder de modificar dinamicamente a estrutura de um documento HTML. Sendo possível, por exemplo:

* Adicionar/modificar/remover tags, textos, imagens e qualquer elemento no HTML.
* Alterar estilos CSS da página.
* Criar novos eventos HTML.

Vamos conferir formas de realizar os itens listados no tópico acima.

**Métodos para selecionar elementos no HTML**

* document.getElementByID(id) - Selecionar um elemento pelo ID.
* document.getElementsByTagName(name) - Selecionar um elemento pelo nome.
* document.getElementsByClassName(name) - Selecionar um elemento pelo nome da classe.

**Propriedades e métodos para alterar elementos no HTML**

* element.innerHTML - Esta propriedade obtém ou altera qualquer elemento no HTML, inclusive tags.
* element.innerText - Esta propriedade permite inserir textos no HTML.
* element.attribute - Esta propriedade altera o valor de um elemento HTML
* element.setAttribute(atributo, valor) - Este método altera o valor de um atributo de um elemento HTML.

**Adicionando e excluindo elementos**

* document.write() - Escreve no fluxo de saída do HTML.
* document.appendChild() - Adiciona um elemento HTML.
* document.removeChild() - Remove um elemento HTML.
* document.replaceChild() - Substitui um elemento HTML.
* document.createElement() - Cria um elemento HTML.

ARTIGO: Hoisting no Javascript

Autores(as): [Luan Alves](https://cursos.alura.com.br/user/luanalvesdev) e [Mônica Hillman](https://cursos.alura.com.br/user/monicahillman)

Quando uma equipe trabalha em algum projeto, muito provavelmente o código será lido e alterado por diversas pessoas, dessa maneira, é necessário entender diferentes formas possíveis de escrever o mesmo código, obtendo uma boa produtividade na sua performance de desenvolvimento.

Contudo, é importante para uma **pessoa programadora** entender o **que é Hoisting** e como se comporta em diferentes casos nesta linguagem.

Vamos analisar este **código de function declaration**, e o que retorna:

console.log(soma(2, 5))

function soma(a, b) {

return a + b

}

Este código retorna o valor: 7

Repare que a função consegue ser chamada antes mesmo de ter sido declarada. **Hoisting** é o termo que explica essa situação, em português ele significa “içamento”, ou “elevação” e foi citado pela primeira vez no **[ECMAScript® 2015 Language Specification](https://262.ecma-international.org/6.0/" \t "_blank)**. O Hoisting permite que você execute funções antes das suas declarações. Na prática, inicialmente as **declarações de funções** são colocadas na memória durante a [fase de compilação](https://www.alura.com.br/artigos/o-que-e-compilacao) e, mesmo assim, permanecem no mesmo lugar que estão digitadas.

A razão pela qual o código anterior funciona é que os mecanismos JavaScript movem a função soma para o início do escopo, como ilustra o código abaixo:

function soma(a,b){

return a + b

}

console.log(soma(2,5))

Entretanto, será que Hoisting funciona em outros tipos de código? Vamos conferir!

console.log(alura)

var alura = ‘cursos’;

Este código retorna:

Undefined

A utilização de **Hoisting em var não é indicada**, pois a variável criada é elevada para o escopo, mas sem o seu valor, e com isso, retorna valor undefined.

O comportamento é parecido utilizando também var function:

function testaHoisting() {

console.log(‘testaHoisting’, alura)

var alura = ‘cursos’

}

Este código retorna: undefined.

**Hoisting também não é indicado utilizando let**, pois acontece um outro tipo de **comportamento não desejado:**

console.log(alura)

let alura = ‘cursos’;

Este código retorna:

script.js:1 Uncaught ReferenceError: Cannot access 'alura' before initialization```

Esta mensagem em português significa “Não é possível acessar ‘alura’ antes de sua inicialização'', ou seja, a linguagem JavaScript reconhece que ‘let = alura’ existe, porém não consegue acessar sua declaração. Esse comportamento de Hoisting com let é o mesmo ao utilizar const.