



ES6+

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

- **Revisão**
 - JSON & localStorage
- **ES6 (ECMA Script 2015)**
 - Escopo

- **JSON**

- `var json = JSON.stringify(objeto);`
- `var objeto = JSON.parse(json);`

- **localStorage**

- `localStorage.setItem('chave', string);`
- `var item = localStorage.getItem('chave');`
- `localStorage.removeItem('chave');`
- `localStorage.clear();`

Porque ECMAScript?

- **E**uropean **C**omputer **M**anufacturers **A**ssociation (**ECMA**)
- Assumiu o nome de apenas **ECMA** para ser considerada “global”
- Organização sem fins lucrativos que padroniza informação e sistemas de comunicação (um deles, o **ECMAScript**)

JavaScript X ECMAScript

- Na década de 90, o **JavaScript** foi absorvido pela maioria dos navegadores, mas não tinha um padrão (cada navegador fazia sua implementação);
- Com o objetivo de padronizar (para melhorar a vida de nós, desenvolvedores), a **ECMA** recebe a responsabilidade de fazer uma padronização da linguagem.
- Chamamos essa padronização (aceita pela maioria dos navegadores), de **ECMAScript**.

- **ES6:** 6ª edição do ECMAScript (também conhecida como **ES2015**);
- Mudanças significativas:
 - Declaração de **classes** com sintaxe mais amigável para desenvolvedores de linguagens com orientação a objeto baseada em classes
 - ES6 **Modules: import, export**
 - **for ... of** loops
 - **Generators**
 - **Arrow functions**
 - **let, const**
 - etc

- Variáveis: `let`, `const`
- Arrow Functions: `() => {}`
- Promises (promessas)
- Parâmetros padrão:
`function (a = 10) {}`
- Rest: `function (...args) {}`
- Operadores `**` e `**=`
- String e `Array.includes()`
- `Object.entries()` e `.values()`
- Funções `async` `await`
- Destruct:
`let { a, b, ...rest } = obj`
- Spread:
`let obj = { a: 'b', ...o }`
- Módulos `import` `export`
- ...

- A partir do **ES6**, as mudanças foram mais suaves, seguindo direções estabelecidas nesta revisão da linguagem.
- Convencionou-se usar o sinal “+” para se referir às mudanças implementadas no ES da sexta edição em diante.

- Escopo é o que responde a pergunta **“Onde essas variáveis estão disponíveis?”**
- Pensando no JavaScript, temos 3 grandes regiões onde a variável pode estar disponível:
 - Escopo global;
 - Escopo da função;
 - Escopo do bloco;

- Escopo global: Disponível para toda a aplicação;
- Escopo da função: Disponível apenas dentro da função na qual foi criada;
- Escopo do bloco: Disponível dentro dos blocos;

Obs.: Quando vemos chaves abrindo e fechando no código, ali há um bloco

Onde as variáveis estão disponíveis?

```
function definirLargura(){  
  var largura = 100;  
  console.log(largura);  
}  
definirLargura();  
  
console.log(largura);
```

Onde as variáveis estão disponíveis?

```
var altura = 100

if(altura > 90) {
  var largura = 100;
  console.log(largura);
}

console.log(largura);
```

- Quando dentro de uma função, o escopo é de função.
- Quando fora de uma função, o escopo é global;

O que acontecerá neste caso?



```
//Como pode confundir?  
var idade = 31;  
  
if(idade > 12){  
    var idadeEmAnosDeCachorro = idade * 7;  
    console.log("Você tem " + idadeEmAnosDeCachorro + " anos em idade de cachorro!");  
}  
  
console.log(idadeEmAnosDeCachorro);  
//Continua acessível fora do bloco onde foi criada
```

- Escopo de bloco (acessíveis apenas dentro do bloco).

O que acontecerá neste caso?



```
let pontos = 50;  
let vencedor = false;  
  
if(points > 40){  
  console.log("passei pelo if");  
  let vencedor = true;  
}  
  
console.log("vencedor", vencedor);
```


- Diferente de `let` e `var`, não pode ter sua referência alterada depois do momento da sua criação (reatribuição);
- Como a referência não pode ser atribuída fora do momento de sua criação, também não pode ser declarada sem que se insira imediatamente o seu valor;
- Não é que seus valores são imutáveis, mas ela não pode ter sua referência alterada. Se o valor dentro da referência for alterado, sem problemas

O que acontecerá neste caso?

```
const pessoa = {  
  nome: 'Vinícius',  
  idade: 31  
}  
  
//O que vai acontecer nas ocasiões abaixo?  
pessoa = { nome: 'Chinforínfolá'};  
  
pessoa.idade = 40;  
  
//Como impedir que propriedades sejam alteradas?  
const vinicius = Object.freeze(pessoa);
```

Temporal dead zone

- Declarações de variável `var` são “hoisted” (içadas);
- Declarações de `let` e `const` não;

```

//Não lança exceção:
console.log(pizza);
var pizza = 'Hmmm... 🍕🍕🍕';

//Lança exceção:
console.log(picanha);
const picanha = 'So tasty... 🐮';
```

Var

O JavaScript apenas eleva (hoists) as declarações, não as inicializações. Se uma variável for declarada e inicializada após usá-la, o valor será undefined. Por exemplo;

```
console.log(num); // Retorna undefined
var num;
num = 6;
```

```
num = 6;
console.log(num); // retorna 6
var num;
```

Var

Se você declarar a variável depois que ela for usada, mas inicializá-la antecipadamente, ela retornará o valor;

Mas afinal, qual usar?

- **var** - quando quiser que algo esteja disponível de forma global e você estiver consciente, lidando com todos os riscos;
- **let** - Quando for necessário reatribuição;
- **const** - quando não for necessário reatribuição.

Escopos

Escopo	Const	Let	Var
Global	no	no	yes
Função	yes	yes	yes
Bloco	yes	yes	no
Reatribuído	no	yes	yes

MATERIAL COMPLEMENTAR



Var, Let, Const - Tudo o que você precisa saber | <https://youtu.be/ZOx7iTnBqFQ>

Como funciona o var, let e const? #01 | <https://youtu.be/EFoEqHlwxqY>

Differences Between Var, Let, and Const | <https://youtu.be/9WlIQDvt4Us>

MATERIAL COMPLEMENTAR



Javascript ES6 | https://www.w3schools.com/js/js_es6.asp

let - JavaScript | <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let>

const - JavaScript | <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const>

var - JavaScript | <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var>



DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!



<LAB365>