TRYBE
Modulo II – Front-end

Bloco 7 – Javascript ES6

Nomes: ES6, ECMAScript 6 ou ES2015.

Conceito de refatoração: alterar estrutura sem alterar o comportamento do código.
Saber riscos e lembrar do importante: code legível por outro ser humano, regressão e teste unitário.

1) let, const, template literals e arrow functions

var, let e const

Três elementos de diferenciação: escopo, declaração e hoisting.
Hoisting é um mecanismo do Js em que variáveis e funções são movidas para o topo do escopo atual.

var
- com hoisting (assim, podem ser utilizadas mesmo antes de declaradas)
- valor default é “undefined”.
- risco de vazamento de escopo.

let
- com escopo de bloco (só podem ser utilizadas após declaradas).
- Erro “undefined” por ex com “let namevariable;”

const
- impossivel dar “undefined”, inicializada obrigatoriamente no momento de sua declaração.
- pode dar erro “Assignement to constant variable”
- mutável em apenas um caso: sendo objeto pode alterar o valor enquanto chave fica constante.
- pode ser redeclarada apenas em escopo diferente.

Dica: sempre usar const, se der erro, trocar por let! Esquecer var.

freeCodeCamp(🔥) Donate

- `var` declarations are globally scoped or function scoped while `let` and `const` are block scoped.
- `var` variables can be updated and re-declared within its scope; `let` variables can be updated but not re-declared; `const` variables can neither be updated nor re-declared.
- They are all hoisted to the top of their scope. But while `var` variables are initialized with `undefined`, `let` and `const` variables are not initialized.
- While `var` and `let` can be declared without being initialized, `const` must be initialized during declaration.

keyword	const	let	var
global scope	NO	NO	YES
function scope	YES	YES	YES
block scope	YES	YES	NO
can be reassigned	NO	YES	YES

Template literals

Definições
Literal: uma notação. Usamos literals o tempo todo, para String, RegExp, Array…
Template literals: literals que representam uma String com suporte a interpolation e multiplas linhas.

Interpolation

```
let myVar = 'es6';

console.log(`Welcome ${myVar}!`);

// > Welcome es6!
```

The ${variable} syntax used above is a placeholder. Basically, you won't have to use concatenation with the + operator anymore.

Multiplas linhas

```
let myText = `This is the first line
This is the second line
This is the third line`;

console.log(myText);

// > This is the first line
//   This is the second line
//   This is the third line
```

Entende automaticamente as quebras de linha, substitui ***\n***

Interpolated Expressions
= expressões que executa para inserir seu resultado dentro da string.
Sintax ${}

```
let value = 5.123124,
  name = 'Vincent';

console.log(`${name.toUpperCase()}, you owe me U$${value.toFixed(2)}!`);

// > VINCENT, you owe me U$5.12!
```

Expression scope: com o conhecimento de var let, entender que resultado vai sair usando interpolation.

Tagged Template Literals, forma mais avançada de Template Literals, com possibilidade de modificar a saída dos template strings usando uma função.

```
1  var a = 5;
2  var b = 10;
3
4  function tag(strings, ...values) {
5    console.log(strings[0]); // "Hello "
6    console.log(strings[1]); // " world"
7    console.log(values[0]);  // 15
8    console.log(values[1]);  // 50
9
10   return "Bazinga!";
11 }
12
13 tag`Hello ${ a + b } world ${ a * b}`;
14 // "Bazinga!"
```

Arrow functions

Para escrever functions de um jeito mais curto, geralmente em apenas uma linha.
Sem {} ou palavra function ou return que fica implicito.

Sintax
let/const namefunction = parameters function => what function does and return;

Exemplos com 2, 1 e 0 param

```
var soma = function(num1, num2) {
  return num1 + num2;
}

var soma = (num1, num2) => {
  return num1 + num2;
}

var soma = (num1, num2) => num1 + num2;
```

```
// var contaPalavras = function(frase) {
//   return frase.split(' ').length;
// }
//
// var contaPalavras = (frase) => {
//   return frase.split(' ').length;
// }

var contaPalavras = frase =>
  frase.split(' ').length;

console.log(contaPalavras('Canal Sou Dev'))
```

```
// var mostraSegundos = function() {
//   return new Date().getSeconds();
// }
//
// var mostraSegundos = () => {
//   return new Date().getSeconds();
// }

var mostraSegundos = () =>
  new Date().getSeconds();

console.log(mostraSegundos())
```

Exemplo envolvendo objeto
Diferença: uso de ({})

```
// var objetoUsuario = function(id, nome) {
//   return {
//     id: id,
//     nome: nome,
//   };
// };
//
// var objetoUsuario = (id, nome) => {
//   return {
//     id: id,
//     nome: nome,
//   };
// };

var objetoUsuario = (id, nome) =>
  ({id: id, nome: nome});

console.log(objetoUsuario(1, 'Henrique'))
```

Exemplo com Array

```
var itemsCarrinho = [
  { id: 1, nome: 'Item 1', preco: 1200 },
  { id: 2, nome: 'Item 2', preco: 2500 },
];

// var total = itemsCarrinho.reduce(function(total, item) {
//   return total + item.preco;
// }, 0);

// var total = itemsCarrinho.reduce((total, item) => {
//   return total + item.preco;
// }, 0);

var total = itemsCarrinho.reduce((total, item) => total + item.preco, 0);

console.log(total);
```

this
Js tem a palavra-chave *this*, que tem *o valor determinado pela forma como chamamos a função.*
A arrow function, tem seu *this* definido lexicalmente, seu *this* recebe o contexto de execução de onde a arrow function é declarada.
(iremos aprofundar o this no bloco sobre React)

Quando não usar arrow function
Arrow functions são funções anônimas, por isso é melhor evitar usar quando a function for *executada mais de uma vez* no code inteiro, para melhor rastreabilidade de potenciais erros.
Também é melhor evitar em caso de precisar usar o *this*.

Dicas diversas:

use strict dispara exceções de **erros silenciosos** do Js, tornando elas em erros explicitos.
https://www.w3schools.com/js/js_strict.asp

_

String.raw faz com que qualquer **string de escape** não seja processada, lendo tudo o que contem como literal.

Ex:
Criando um exemplo com caminhos de pasta no windows usando separador de caminhos o \, meta caracter que significa escape, trocando o comportamento do próximo caracter que vem depois dele (\n é nova linha \t é tab, \s é espaço).

```
const caminho = 'C:\novaPasta\teoria_conspiracao\sempre_ativos'
console.log(caminho)
Retorna quebrado:
C:
ovaPasta eoria_conspiracao empre_ativos

console.log(String.raw(caminho))
Retorna certinho:
C:\novaPasta\teoria_conspiracao\sempre_ativos
```

Como escrever || para ler | literalmente, Ou \$ para entender o $ literalmente

_

//Lambda Function: ainda devo achar exact def.//

_
JavaScript String split() Method
https://www.w3schools.com/jsref/jsref_split.asp

---

2) Metódos sobre objetos

Lembrando dos objetos

```
let object = {
propriedade/chave: valor,
propriedade2/chave2:valor2,
}
```

O conjunto chave:valor é chamado de **entrada**.

Adicionar novas propriedades a um objeto

Duas técnicas:

- *reescrever* diretamente o objeto

- adicionar a nova propriedade com a sintax
objeto.nomeDaPropriedade

ou
objeto[variavelQueContemONomeDaPropriedade]
“Note that we do not use quotes around the variable name when using it to access the property because we are using the value of the variable, not the name.”
→ O resultado dessas sintax é o mesmo. Porém, a diferença é que pode usar direto o objeto.propriedade quando vc conhece as propriedades do objeto, mas para resgatar o valor da propriedade, vc usa o objeto[propriedade], por ex num for.

Object.keys

Uso
Para acessar todas as chaves de um objeto, listando elas e retornando-as em um array.
Vantagem: não precisar mais utilizar um for para percorrer um objeto para pegar sua lista de chaves.

Sintax
Object.keys(objectname);

Object.values

Uso
Para acessar todos os valores de um objeto, listando e retornando eles em um array.
Vantagem: não precisar mais utilizar um for para percorrer um objeto para pegar sua lista de valores.

Sintax
Object.values(objectname);

Object.entries

Uso
Para acessar os pares chave-valor de um objeto.
O método Object.entries() retorna uma array de arrays, dos próprios pares [key, value] enumeráveis de um dado objeto, na mesma ordem dos objetos providos através do loop for...in .

Sintax
Object.entries(objectname);

Object.assign

Uso
Copiar os valores de todas as propriedades de um ou mais objetos para um objeto destino.
Pode sobreescrever também umas propriedades com o mais recente.

Sintax
Object.assign(destination, object1);
Object.assign(destination, object1, object2, etc);

Para modificar os dados do novo objeto sem alterar o objeto inicial: colocar como primeiro paramêtro um objeto vazio {} ------→

```
const person = {
  name:'Roberto',
};

const lastName = {
  lastName: 'Silva',
};

const newPerson = Object.assign({},person,lastName);
newPerson.name = 'Gilberto';
console.log(newPerson);
console.log(person);
```

Agora, apenas o objeto newPerson será modificado.

---

3) Testes unitários em JavaScript

Porções de código responsáveis por **validar o comportamento de unidades funcionais de código** (functions, propriedades, construtores…).
Critérios: **confiança, funcionalidade e performance**.
Uso: verificar correto, garantir inclusive depois de alteração, fácil manutenção, visão de qualidade, **documentação** do código.

NodeJS Assert

O módulo Assert provê uma forma de testar expressões. Se essa expressão é avaliada com o **valor 0 ou false**, o teste falhará e o programa será terminado.

Métodos do Assert

Assert Methods

Method	Description
assert()	Checks if a value is true. Same as assert.ok()
deepEqual()	Checks if two values are equal
deepStrictEqual()	Checks if two values are equal, using the strict equal operator (===)
doesNotThrow()	
equal()	Checks if two values are equal, using the equal operator (==)
fail()	Throws an Assertion Error
ifError()	Throws a specified error if the specified error evaluates to true
notDeepEqual()	Checks if two values are not equal
notDeepStrictEqual()	Checks if two values are not equal, using the strict not equal operator (!==)
notEqual()	Checks if two values are not equal, using the not equal operator (!=)
notStrictEqual()	Checks if two values are not equal, using the strict not equal operator (!==)
ok()	Checks if a value is true
strictEqual()	Checks if two values are equal, using the strict equal operator (===)
throws()	

Exemplo de sintax, com method strictEqual:
assert.method(expected, actual, “message in case of failure”);
Outro ex:

```
const assert = require('assert');

function add(a, b) {
  return a + b;
}

const expected = add(1, 2);

assert(expected === 3, 'one plus two is three');
assert.ok(expected === 3, 'one plus two is three');
assert.equal(expected, 3, 'one plus two is three');
assert.notEqual(expected, 4, 'one plus two is three (NOT Four!)');
```

Todas sintax de methods
https://nelsonic.gitbooks.io/node-js-by-example/content/core/assert/README.html
e material da Node https://nodejs.org/api/assert.html .

Fluxo de trabalho
- primeiro verificar se o que quer testar (function…) existe (*typeof*)
- fazer o teste, escrever o esperado, falhar, voltar.
- com o Run Code (atalho ctrl alt n), teste deu certo quando não aparece nada. -->

```
[Running] node "/Users/leandro/example.js"

[Done] exited with code=0 in 0.087 seconds
```

Organizar
Arquivo separado de .js (para cada arquivo .js seu a ser testado)

---

4) Testando em pequenos passos

"Um bug coberto por um teste é um bug que nunca voltará
Uma funcionalidade coberta por um teste é uma funcionalidade que nunca quebrará"
Objetivo: criar uma mente programadora orientada a testes. **TDD: Test Driven Development**

Dicas diversas

–
Escrever ifs e elses em uma linha:
if (condition) return x;
else if (condition) return y;
else return z;
–
.repeat no Js
https://www.w3schools.com/jsref/jsref_repeat.asp
–
.splice num array

```
let arr = ['foo', 'bar', 10, 'qux'];

// arr.splice(<index>, <steps>, [elements ...]);

arr.splice(1, 1);           // Removes 1 item at index 1
// => ['foo', 10, 'qux']

arr.splice(2, 1, 'tmp');    // Replaces 1 item at index 2 with 'tmp'
// => ['foo', 10, 'tmp']
```

```
arr.splice(0, 1, 'x', 'y'); // Inserts 'x' and 'y' replacing 1 item at index 0
// => ['x', 'y', 10, 'tmp']
```

5) Projeto final do bloco 7

Uso do avaliador npm, Travis, sem cypress

→ *npm*
Ao fazer o npm install, sem o fix até necessário, desconfiar do json alterado.
npm test testfilepath
Em caso de retornar “no test found”, verificar o caminho passado.

→ Avalidor *Travis* fica sem responder até completar o projeto.

→ Lembrar de retirar ou comentar o *assert.fail()* que foi colocado apenas para impedir o aluno de esquecer de editar o arquivo.

Dicas diversas

Math.round() https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Math/round
ou *Math.floor()* para arredondar por baixo
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Math/floor
–
toPrecision() https://www.w3schools.com/jsref/jsref_toprecision.asp
–
Sintax para escrever arrow function dentro de objeto

```
const createStudent = (nameStudent) => {
const object = {
name: nameStudent,
feedback: () => 'Eita pessoa boa!',
};
return object;
};
```

–
endsWith() para avaliar finais de palavras, que retorna true ou false
https://www.w3schools.com/jsref/jsref_endswith.asp#:~:text=The%20endsWith()%20method%20determines,()%20method%20is%20case%20sensitive.