

FULL STACK WEB DEVELOPER

FERNANDO LIRA

FLAG

JAVASCRIPT

APRESENTAÇÃO – FERNANDO LIRA

3



it.fernandolira@gmail.com



<https://www.linkedin.com/in/fernandolira74/>



+351 93 317 99 21



@fernandolira74



Built-in objects

- **Date**
- **Math**
- **String**
- **Array**

Objetos Globais

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects

Number

```
var num1 = 1;
var num2 = "2";
var num3 = Number("3");

console.log(num1);
console.log(num2);
console.log(num3);

console.log(typeof num1);
console.log(typeof num2);
console.log(typeof num3);

var num4 = Number(num2);

console.log(num4);
console.log(typeof num4);
```

Number - isNaN

```
var num2 = "1";  
var num4 = Number(num2);  
  
if (!Number.isNaN(num4)) {  
    console.log(num4);  
    console.log(typeof num4);  
}
```

Number - parseInt

```
var num2 = "1";  
var num5 = Number.parseInt(num2);  
console.log(num5);  
console.log(typeof num5);
```


Number – toFixed e toString

```
var num2 = 22;  
var num3 = 3;  
var num6 = num3 / num5;  
console.log(num6); // 0.1363636363636  
console.log(num6.toFixed(2));  
console.log(num6.toString(2)); //em binário
```

Number

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Number

Math

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Math

String

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/String

Date

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Date

EXERCÍCIOS

Desafios

Elabore uma função que recebe dois parâmetros: o primeiro é um array de números e o segundo é um número que especifica uma quantidade de dígitos. Essa função deverá retornar somente aqueles números do array que têm a quantidade de dígitos indicada pelo segundo parâmetro. Exemplos:

```
filtrarPorQuantidadeDeDigitos([38, 2, 365, 10, 125, 11], 2)
```

```
// retornará [38, 10, 11]
```

```
filtrarPorQuantidadeDeDigitos([5, 9, 1, 125, 11], 1) //
```

```
retornará [5, 9, 1]
```

Desafios

Elabore uma função que escreve data e hora atual no seguinte formato:

Hoje é : Quarta.

Hora atual : 10 : 30 PM

Elabore uma função que escreve o dia da semana de uma data especificada em dia, mês e ano:

Desafios

Elabore uma função que remove um caracter de uma string e uma posição recebidas.

Elabore uma função que troca o primeiro caracter com o último de uma string e recebida.

Elabore uma função que retorne o número de palavras de uma frase.

Elabore uma função que receba uma frase e um caracter e remova todas as ocorrências desse caracter.

Elabore uma função que recebe uma frase e exibe-a no sentido inverso

VAR, LET e CONST

Na maioria das linguagens de programação, o escopo das variáveis locais é vinculado ao bloco onde elas são declaradas. Sendo assim, elas “morrem” no final da instrução em que estão a ser executadas. Será que isso se aplica também à linguagem JavaScript?

VAR, LET e CONST

```
var exibeMensagem = function() {  
  var mensagemForaDoIf = 'Caelum';  
  if(true) {  
    var mensagemDentroDoIf = 'Alura';  
    console.log(mensagemDentroDoIf) // Alura ;  
  }  
  console.log(mensagemForaDoIf); // Caelum  
  
  console.log(mensagemDentroDoIf); // Alura  
}  
  
exibeMensagem();  
//imprime tudo!!!
```

VAR, LET e CONST

```
var exibeMensagem = function() {  
  mensagem = 'Alura';  
  console.log(mensagem);  
  var mensagem;  
}  
  
exibeMensagem();  
//imprime mesmo com a definição da variável após
```

VAR, LET e CONST - Hoisting

Em JavaScript, toda variável é “elevada/içada” (hoisting) até o topo do seu contexto de execução. Esse mecanismo move as variáveis para o topo do seu escopo antes da execução do código.

No exemplo acima, como a variável `mensagemDentroDolf` está dentro de uma function, a declaração da mesma é elevada (hoisting) para o topo do seu contexto, ou seja, para o topo da function.

É por esse mesmo motivo que “é possível usar uma variável antes dela ter sido declarada”: em tempo de execução a variável será elevada (hoisting) e tudo funcionará corretamente.

VAR, LET e CONST - Hoisting

```
void function(){  
    console.log(mensagem);  
}();  
  
var mensagem;  
//undefined
```

VAR, LET e CONST - let

```
var exibeMensagem = function() {  
  if(true) {  
    var escopoFuncao = 'Caelum';  
    let escopoBloco = 'Alura';  
  
    console.log(escopoBloco); // Alura  
  }  
  console.log(escopoFuncao); // Caelum  
  console.log(escopoBloco);  
}  
  
exibeMensagem();  
//Alura  
//Caelum  
//Errooooo!!!!
```

VAR, LET e CONST - let

Foi a pensar em trazer o escopo de bloco (tão conhecido em outras linguagens) que o ECMAScript 6 destinou-se a disponibilizar essa mesma flexibilidade (e uniformidade) para a linguagem.

Através da palavra-chave `let` podemos declarar variáveis com escopo de bloco.

VAR, LET e CONST - const

```
void function(){  
  let mensagem;  
  console.log(mensagem); // Imprime undefined  
}();
```

VAR, LET e CONST - const

```
void function(){  
  const mensagem = 'Alura';  
  console.log(mensagem); // Alura  
  mensagem = 'Caelum'; //Erro!!!  
}();
```

VAR, LET e CONST - const

```
// constante válida  
const idade = 18;  
  
// constante inválida: onde está a inicialização?  
const pi;
```

VAR, LET e CONST

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

Operadores Destructuring

```
const pessoa = {  
  nome: 'José',  
  idade: 30,  
  endereco: {  
    rua: 'Av. Principal',  
    numero: 45  
  }  
}  
  
const {nome, idade} = pessoa;  
console.log(nome, idade); //José 30
```

Operadores Destructuring

```
const {nome: n, idade: i} = pessoa; //mudar o nome das variáveis  
console.log(n,i);
```

```
const {sobrenome, reformado = false} = pessoa; //valor por  
defeito se não existir  
console.log(sobrenome, reformado); // undefined false
```

```
const {endereco: {rua}} = pessoa;  
console.log(rua);
```

```
const {xpto: {abc}} = pessoa; //ERRO!!!! O caminho não existe!
```

Operadores Destructuring

```
//Arrays  
const [a]=[10];  
console.log(a); // 10  
  
const [n1, ,n2, ,n3,n4=0] = [4,7,3,7];  
console.log(n1,n2,n3,n4); //4 3 undefined 0
```

Operadores Destructuring

```
//Funções
function escreveNome({nome = 'Vazio'}) {
  console.log(nome);
}

escreveNome(pessoa); // José
escreveNome(' '); //Vazio
escreveNome(); //Erro
```


Operador Spread – Espalhar os elementos

```
//usar spread com objetos
const funcionario = {nome: 'Maria', salario: 345};
const clone = { ativo: true, ...funcionario};

console.log(clone);

//usar spread com arrays
const grupoA = ['João', 'Rita', 'Camila'];
const grupoFinal = ['Maria', ...grupoA, 'Pedro'];

console.log(grupoFinal);
```

Operador Rest – Juntar os elementos

```
//Operador rest
function total(...numeros) {
  let total=0;
  numeros.forEach(n => total+=n);
  return total;
}

console.log(total(4,8,6,2,5));
```

Operador Spread – útil!! (verificar no can i use!)

```
console.log(Math.max(1,3,5)); //5  
console.log(Math.max([1,3,5])); //NaN  
console.log(Math.max(...[1,3,5])); //5
```

Operador Rest vs Spread

```
// REST
function myBio(firstName, lastName, ...otherInfo) {
  return otherInfo;
}

console.log(myBio("Oluwatobi", "Sofela", "FLAG", "Web Developer", "Male"));

//SPREAD
function myBioSpread(firstName, lastName, company) {
  return `${firstName} ${lastName} runs ${company}`;
}

console.log(myBioSpread(...["Oluwatobi", "Sofela", "FLAG"]));
```