VARIÁVEIS

O que são variáveis?

Quando precisamos salvar temporariamente o nome que uma pessoa colocou no formulário de contato, ou quando precisamos salvar um valor para executar um cálculo, utilizamos variáveis.

No JavaScript, podemos declarar uma variável de três formas:

- **var**: Declara uma variável podendo atribuir um valor junto na sua criação.

```
var umNumero;
```

```
var umNumero = 10;
```

- **let**: Declara uma variável com escopo de bloco (veremos sobre escopo de bloco mais adiante):

```
let outroNumero = 5;
```

- **const**: Declara uma variável apenas de leitura. Após atribuir um valor a uma const, não será possível altera-lo :

```
const ola = "Olá Mundo";
```

IMPORTANTE!

Apesar de ainda ser muito utilizado, o var não é mais recomendado pelas boas práticas de programação.

Sobre os Nomes das Variáveis:

- **Case-Sensitive:** O JavaScript diferencia entre letras maiúsculas e minúsculas. NOME é diferente de nome.

```
let nome = "Paulo";
let NOME = "Alexandre";
```

Caracteres válidos:

- **Letras**: Você pode usar letras maiúsculas e minúsculas, mas evite cedilha e acentuação.

```
let NomeComprador = "João Silva";
```

Caracteres válidos:

- **Números:** Desde que a variável comece com 1 letra, você pode usar números na definição.

```
let carro2000 = 123;
let w3c = "https://www.w3.org/";
```

Caracteres válidos:

- **Underline “_” e cifrão “$” :** São permitidos em qualquer posição e são mais usados nas variáveis de bibliotecas e frameworks.

```
let _variavel = 99;
let $www = true;
```

TIPOS DE DADOS

Apesar das variáveis das linguagens dinâmicas não precisarem ter seus tipos de dados identificados, isso não quer dizer que eles não existam.

TIPOS DE DADOS

O JavaScript possui os seguintes tipos de dados:

- Number
- String
- Boolean
- Array
- Null e Undefined
- Symbol*
- Object
- Function

Number:

São todos os números reais que aprendemos nas aulas de matemática.

```
let numeroInteiro = 100;

let numeroDecimal = 5.38;

let numeroNegativo = -200.99;
```

Importante:

Estes valores numéricos não são infinitos. Mesmo assim, o JavaScript trabalha com ponto flutuante de dupla precisão (double) que permite valores muito altos.

Máximo e Mínimo:

Para saber quais são estes valores máximos e mínimos, você pode efetuar os seguintes comandos no console.

```
console.log(Number.MAX_VALUE);

console.log(Number.MIN_VALUE);
```

TIPOS DE DADOS

Strings:

São valores de texto, normalmente chamados de “cadeias de caracteres”:

```
let olaMundo = "Olá Mundo";
```

- Você precisa utilizar aspas simples ` ou aspas duplas " para encapsular a String.

```
let olaMundo = "Olá Mundo";

let olaMundo2 = 'Olá Mundo';
```

- **Caracteres Especiais** - Para adicionar caracteres especiais (‘, “, &, quebra de linha e etc.) utilizamos a barra invertida.

```
let caracteresEspeciais = "Lorem \n Ipsum \\ dolor \' \" amet";
```

\n : Nova Linha
\\ : exibe a \
\’ : Exibe ‘
\” : Exibe “

- **Concatenação de Strings:** Você pode unir (concatenar) Strings de diversas formas.

1) Utilizando o sinal de +

```
let ola = "Olá ";
let mundo = "Mundo";

let olaMundo = ola+mundo;
```

2) Utilizando o String.concat

```
let ola = "Olá ";
let mundo = "Mundo";

let olaMundo = ola.concat(mundo);
```

Este método também permite concatenar diversas Strings.

```
let ola = "Olá ";
let mundo = "Mundo";

let olaMundo = ola.concat(mundo," frase ", ola);
```

3) Template de Strings: O template de Strings é uma nova forma de concatenação e utiliza o ${expressão} e a crase ao redor da string.

```
let ola = "Olá";
let mundo = "Mundo";

let olaMundo = `${ola} ${mundo} ${1+1}`;
```

TIPOS DE DADOS

- **Booleanos** – São tipos de dados que possuem apenas dois valores:
  - TRUE: Verdadeiro
  - FALSE: Falso

```
let verdadeiro = true;

let falso = false;
```

Observações:

- Os valores true e false NÃO são escritos entre aspas.
- Cuidado! O JavaScript entende certos valores dos outros tipos de dados como true ou false.

- **Comparações:** Uma das formas mais comuns de se obter valores booleanos é através de comparações.

```
// true
let comparacao = 1 == 1;

//false
let comparacao2 = 1 > 5;

//true
let comparacaoString = "banana" == "banana";
```

- **Comparações**: Se você testar 1 == “1”, obterá true. Para fazer comparações que levam em consideração o valor e o tipo do dado, utilize ===

```
//true
console.log(1 == "1");

//false
console.log(1 === "1");
```

OPERADOR	DESCRIÇÃO	EXEMPLO
==	Valor Igual	3 == 3 (true) 2 == 3 (false)
===	Valor e tipos iguais	3 === "3" (true) 3 === 3 (true) 2 === 3 (false)
!=	Valor diferente de	3 != "3" (false) 1 != 2 (true) 1 != "2" (true)
!==	Valor e tipo diferentes de	1 !== 1 (false) 1 !== 2 (true) 1 !== "1" (true) 1 !== "2" (true)
<, <=	Menor (menor igual) a	10 < 100 (true) 100 < 100 (false) 10 <= 100 (true)
>, >=	Maior (Maior Igual) a	10 > 100 (false) 100 > 100 (false) 10 >= 100 (false) 100 >= 100 (true)

&&	"E" Lógico Retorna true se todos testes forem verdadeiros	(10 < 100) && (5 > 4) (true) (10 < 100) && (5 > 11) (false)
||	"Ou" Lógico Retorna true pelo menos 1 teste for verdadeiro	(10 < 100) || (5 > 4) (true) (10 < 100) || (5 > 11) (false) (10 > 100) || (5 > 11) (false)
!	"Não" Inverte o booleano a sua esquerda	!true (false) !(1 == 1) (false) !(10 != 10) (true)

TIPOS DE DADOS

- **Array (vetor):** É uma lista ou coleção de dados que pode ser acessada por índice.

  Para criar um vetor vazio basta criar uma variável e atribuir [] a

```
let vetor = [];
```

- **Atribuindo valores:** Você pode criar um vetor com seus valores separados por vírgula.

```
let vetor = [1, 22, 0, 100];
```

Você pode adicionar valores de qualquer tipo no vetor.

```
let vetor = [1, "Olá Mundo", true, [1,2,3]];
```

- **Acessando valores:** Os valores podem ser acessados através de seu

```
let vetor = [1, 22, 0, 100];
```

```
//exibirá 22
console.log(vetor[1]);
```

Observe que o índice começa no 0. Então o primeiro item está na posição 0, o segundo na posição 1 e assim por diante.

- **Alterando e atribuindo valores pelo índice:** Com o índice, você pode:

  a) Alterar um valor existente

  b) Inserir um novo valor em uma posição específica.

```
let vetor = [11, 21, 23, 433, 50];

// Altera o valor da primeira posição
vetor[0] = 9000;
// Insere um valor na após o ultimo elemento
vetor[5] = 7;
```

TIPOS DE DADOS

- **Null**: O null é um tipo de dado especial, ele representa a falta de valor de qualquer outro tipo de dado.

```
let x = null;


let y = 1;
y = null;
```

- **Undefined:** Este tipo de dado aparece quando criamos uma variável e tentamos acessar seu valor antes de ter atribuído algo a ela.

```
let x;

console.log(x);
```

- **Undefined !== Null**: undefined e null são diferentes.

```
console.log(null === undefined);
```

Resumidamente, isto ocorre pois null ainda é um valor e undefined é quando o JavaScript não sabe qual o tipo de dado.

TIPOS DE DADOS

- **Objeto:** É um tipo de dado composto pelos outros tipos. Com ele, podemos organizar informações relacionadas em uma variável.

```
let carro = {
    rodas: 4,
    portas: 2,
    nome: "um carro",
    aVenda: true
};
```

- **Criação**: Um objeto vazio é bem simples de criar.

```
let carro = {};
```

- No caso de um objeto com propriedades (variáveis), fazemos assim:

```
let carro = {
    rodas: 4,
    nome: "Carro"
};
```

- Caso você já tenha criado o objeto e queira adicionar um novo, você pode fazer de duas formas:

```
carro.portas = 2;
carro["portas"] = 2;
```

- Note que se você usar a segunda opção, precisa ter uma String dentro dos [].

- A alteração de dados funciona da mesma forma.

```
carro.rodas = 5;
carro.portas = 4;
carro.nome = "Carrão";
```

```
carro["rodas"] = 3;
carro["portas"] = 2;
carro["nome"] = "Carrinho";
```

TIPOS DE DADOS

- **Funções:** As funções são utilizadas para criarmos uma sequencia de operações para serem executadas.

```
let olaMundo = function(){
    console.log("Olá Mundo");
    console.log("Olá Mundo novamente");
    console.log("Olá Mundo mais uma vez");
}

olaMundo();
```

- Note que para fazer a função executar, você precisa chama-la com os parênteses.

- Outra forma de criar uma função é chamando o function com o nome dela.

```
function olaMundo(){
    console.log("Olá Mundo");
    console.log("Olá Mundo novamente");
    console.log("Olá Mundo mais uma vez");
}

olaMundo();
```

- Também é possível passar valores para a função acessar.

```
let somar = function(valor1, valor2){
    let resultado = valor1 + valor2;
    console.log(resultado);
}

somar(1,2);
somar(4,4);
somar(99,1);
```

Replique este código no seu computador para que você possa ver os resultados.

- O último recurso que estaremos vendo da função (por enquanto) é o comando return.

```
let somar = function(valor1, valor2){
    let resultado = valor1 + valor2;
    return resultado;
}
```

A primeira coisa que você precisa saber. O return para a função e devolve um valor.

Então, estes estão corretos:

```
let somar = function(valor1, valor2){
    let resultado = valor1 + valor2;
    return resultado;
}
```

```
let subtrair = function(valor1, valor2){
    console.log(valor1 - valor2);
    return valor1 - valor2;
}
```

E estes não executarão corretamente

```
let somar = function(valor1, valor2){
    return valor1 + valor2;
    console.log(1);
}
```

```
let subtrair = function(valor1, valor2){
    valor2 = valor2 + 5;
    return valor1 - valor2;
    valor1 = 1;
}
```

- Agora que você sabe como usar o return, vamos falar sobre a utilidade dele:

```
let somar = function(valor1, valor2){
    return valor1 + valor2;
}

let resultado = somar(10,10);
```

Uma função com return devolve um valor que podemos guardar em uma variável.

- O return faz com que a função seja uma ferramenta excelente para tornarmos nosso código mais simples de entender.
- Conforme formos avançando no curso, você verá que utilizaremos bastante as funções em nosso código
- **Curiosidade:** Caso você não coloque o return, por default as funções devolvem undefined.