


Prof. Marcos Nava



Programação de Scripts



Agenda da Aula

- Ambiente para desenvolvimento
- Instalação do Node.js
- Var, let e const
- Number
- Math
- Strings
- Template Strings
- Boolean
- Array
- Object
- Diferenças entre null e undefined

Ambiente de Desenvolvimento

- Para nossa aula vamos utilizar as seguintes ferramentas:
 - Visual Studio Code
 - Node.js
 - Banco de Dados MariaDB ou MySQL
 - Bibliotecas Node:
 - express
 - express-session
 - EJS
 - bcrypt.js
 - mysql2
 - sequelize

Visual Studio Code

É uma ferramenta da Microsoft gratuita muito utilizada por desenvolvedores do mundo todo.

Endereço para download: <https://code.visualstudio.com/>

Ela é utilizada para várias linguagens e tecnologias:

- JavaScript
- Angular
- Flutter
- Assembly
- Node.js
- C/C++
- Dart
- Python

Node.js

O Node (Node.js) é um ambiente de execução JavaScript.

Ele permite a criação de aplicações back-end em Javascript e também a criação de aplicações locais para desktop.

Para a instalação devemos fazer o download do seguinte endereço: <https://nodejs.org/en/>

Dê preferência à versão LTS (Long Term Service) pois ela é mais estável e será o padrão até 30/04/2024.

JavaScript - Criação de Variáveis

Para o JavaScript podemos criar variáveis usando três comandos:

- `var`
- `let`
- `const`

var

Quando criamos uma variável usado var, temos a criação em escopo global ou local.

Global em qualquer parte do programa, mesmo que dentro de blocos de código.

Local em funções.

var

Exemplo:

```
var numero = 1;
{
  var numero = 2;
  console.log("dentro=", numero);
}
console.log("fora=", numero);
```

Saída:

```
dentro=2
fora=2
```


let

Quando criamos uma variável usado let, temos a criação em escopo global ou local.

Global quando fora de qualquer funções

Local em funções e bloco de código.

Por exemplo, se usar em um for ela só valerá dentro da execução dele

let

Exemplo:

```
let numero = 1;
{
  let numero = 2;
  console.log("dentro=", numero);
}
console.log("fora=", numero);
```

Saída:

```
dentro=2
fora=1
```

const

Usando const, você cria uma constante.

Parece estranho, mas devemos sempre dar preferência para o uso de const.

Isto ocorre porque, quando rodando em um servidor, as constantes vão requerer muito menos memória e CPU para "executarem"

Exemplo:

```
const pi = 3.14;  
console.log(pi * pi);
```

number

JavaScript é uma linguagem fracamente tipada, mas isto não quer dizer que as variáveis não possuem um tipo.

O principal "problema" (é mais uma característica) é que o tipo pode mudar conforme o conteúdo.

Por exemplo:

```
let qualquer = 'Legal';  
console.log(typeof qualquer);
```

```
qualquer = 3.1415;  
console.log(typeof qualquer);
```

Saída:

```
string  
number
```

number

As variáveis do tipo number podem ser inteiras ou em ponto flutuante, conforme o conteúdo muda.

Exemplo:

```
const peso1 = 1.0;
const peso2 = Number('2.0');
console.log(peso1, peso2);
console.log(Number.isInteger(peso1));
const avaliacao1 = 9.871;
const avaliacao2 = 6.871;
const total = avaliacao1 * peso1 + avaliacao2 * peso2;
const media = total / (peso1 + peso2);
console.log(media);
console.log(media.toFixed(2));
console.log(media.toString(2));
```

Saída:

```
1 2
true
7.8709999999999996
7.87
111.11011110111110011101101100100010
110100001110010101
```

number

Por causa da característica dos tipos de variáveis serem "mutáveis" temos que tomar alguns cuidados:

Exemplo:

```
console.log(7/0);  
console.log("10" / 2);  
console.log("Show!" * 2);  
console.log(0.1 + 0.7);  
console.log((10.345).toFixed(2));
```

Saída:

```
Infinity  
5  
NaN  
0.7999999999999999  
10.35
```

math

O objeto math é disponível em qualquer container JavaScript e ele possui vários métodos e propriedades que podem nos ajudar no desenvolvimento.

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

Exemplo:

```
const raio = 5.6;  
const area = Math.PI * Math.pow(raio, 2);  
console.log(area);
```

Saída:

98.5203456165759

strings

As strings são conjunto de caracteres dentro de aspas duplas ou aspas simples.

Existem várias funções para tratamento de strings, e podemos dar os seguintes exemplos:

```
const escola = "Fatec Mogi Mirim";
console.log(escola.charAt(4));
console.log(escola.charAt(6));
console.log(escola.charCodeAt(3));
console.log(escola.indexOf('g'));
console.log(escola.substring(1));
console.log(escola.substring(0, 3));
console.log('Faculdade '.concat(escola).concat('!'));
console.log(escola.replace(/i/, 'I'));
console.log('Ana,Maria,Pedro'.split(','));
```

Saída:

```
c
M
101
8
atec Mogi Mirim
Fat
Faculdade Fatec Mogi Mirim!
Fatec MogI Mirim
[ 'Ana', 'Maria', 'Pedro' ]
```


booleanos

Em JavaScript temos variáveis do tipo boolean, mas temos um poder muito maior das outras linguagens.

Um recurso interessante para transformarmos qualquer coisa em boolean é negarmos duas vezes: !!

Vejamos um exemplo de uso mais comum:

```
let isAtivo = false;  
console.log(isAtivo);  
  
isAtivo = true;  
console.log(isAtivo);  
  
isAtivo = 1;  
console.log(!!isAtivo);
```

Saída:

```
false  
true  
true
```

booleanos

Os exemplos abaixo são todos verdadeiros:

```
console.log('os verdadeiros...');  
console.log (!!3);  
console.log (!! -1);  
console.log (!! ' ');  
console.log (!! []);  
console.log (!! {});  
console.log (!! Infinity);  
console.log (!! (isAtivo = true));
```

Saída:

```
os verdadeiros...  
true  
true  
true  
true  
true  
true  
true
```

booleanos

Os exemplos abaixo são todos falsos:

```
console.log('os falsos...');  
console.log (!!0);  
console.log (!!' ');  
console.log (!!null);  
console.log (!!NaN);  
console.log (!!undefined);  
console.log (!! (isAtivo = false));
```

Saída:

```
os falsos...  
false  
false  
false  
false  
false  
false
```

booleanos

Olhe este exemplo bem interessante.

Qual será o resultado?

```
let nome = ''  
console.log(nome || 'Desconhecido');  
nome = 'Lucas';  
console.log(nome || 'Desconhecido');
```

Saída:

Desconhecido
Lucas

array

Os vetores são muito flexíveis em JavaScript.

Eles podem expandir, ter valores excluídos e conter qualquer coisa.

Vejamos alguns exemplos:

```
const valores = [7.7, 8.9, 6.3, 9.2];  
console.log(valores[0], valores[3]);  
console.log(valores[4]);  
console.log(valores);
```

Saída:

```
7.7 9.2  
undefined  
[ 7.7, 8.9, 6.3, 9.2 ]
```

array

O vetor pode conter qualquer coisa e também possui alguns recursos interessantes (pop, push e delete):

```
valores.push({id: 3}, false, null, 'teste');  
console.log(valores);  
console.log(valores.pop());  
delete valores[5];  
console.log(valores);
```

Saída:

```
[  
  7.7,      8.9,  
  6.3,      9.2,  
  10,       { id: 3 },  
  false,    null,  
  'teste'  
]  
teste  
[ 7.7, 8.9, 6.3, 9.2, 10,  
<1 empty item>, false, null  
]
```

array

Mas qual é o tipo de dados de um vetor?

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

Saída:

object

array

Olha que interessante podemos fazer em JavaScript, mas qual seria o resultado?

```
valores[20] = 10;  
console.log(valores);
```

Saída:

```
[  
  7.7,  
  8.9,  
  6.3,  
  9.2,  
  10,  
  <1 empty item>,  
  false,  
  null,  
  <12 empty items>,  
  10  
]
```


object

Objetos em JavaScript são parecidos, mas não são iguais às outras linguagens.

Existem algumas particularidades que vamos ver agora.

No exemplo você vê a criação de um objeto e suas propriedades:

```
const prod1 = {};  
prod1.nome = 'Celular Ultra Mega';  
prod1.preco = 4998.90;  
prod1['Desconto Legal'] = 0.40; // evitar atributos com espaço  
console.log(prod1);
```

Saída:

```
{ nome: 'Celular Ultra Mega', preco: 4998.9, 'Desconto Legal': 0.4 }
```

object

Ele até se parece com um JSON mas não é.

Outra maneira de se criar um objeto é assim:

```
const prod2 = {  
  nome: 'Camisa Polo',  
  preco: 79.90  
};  
console.log(prod2);
```

Saída:

```
{ nome: 'Camisa Polo', preco: 79.9 }
```

object

Você até pode utilizar classes em JavaScript, mas tenha em mente que ele simplesmente irá criar a mesma coisa que já vimos.

Para terminarmos o assunto aqui vejamos isso:

```
console.log(typeof Object);  
console.log(typeof new Object);  
const Cliente = function() {}  
console.log(typeof Cliente);  
console.log(typeof new Cliente);  
class Produto{} // ES 2015 (ES6)  
console.log(typeof Produto);  
console.log(typeof new Produto());
```

Saída:

```
function  
object  
function  
object  
function  
object
```

Diferenças entre null e undefined

Antes de analisarmos estas diferenças precisamos entender como são relacionadas alguns tipos de igualdade.

Vamos considerar um objecto criado e depois um outro que recebe o mesmo objeto. Qualquer alteração em um será refletido no outro. Isto indica que temos duas "variáveis" com a mesma referência.

Exemplo:

```
const a = {name: 'Teste'};  
const b = a;  
b.name = 'Opa';  
console.log(a);
```

Saída:
{ name: 'Opa' }

Diferenças entre null e undefined

Então entendemos que podemos ter mais de uma referência a uma mesma "coisa" na memória.

Toda vez que eu crio uma variável e não coloco nada nela ela fica sem referência ou seja "undefined"

Exemplo:

```
let valor;  
console.log(valor)
```

Saída:
undefined

Diferenças entre null e undefined

Com isso entendemos que undefined quer dizer que uma variável, objeto, vetor, ou qualquer outra coisa ainda não foi inicializada.

Podemos também colocar um valor nulo para uma variável qualquer. Isto indica que ela foi inicializada mas não tem nenhum valor válido, ou seja, ela é nula.

Exemplo:

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

Saída:

null

Diferenças entre null e undefined

Imagine agora que eu crie um objeto vazio e tente acessar uma propriedade que ainda não tenha sido criada. O que aconteceria?

Um objeto vazio indica que ele não é undefined pois foi inicializado e também ele não é nulo pois o objeto existe.

Já a propriedade é undefined pois não foi definida ainda.

Exemplo:

```
const produto = {}  
console.log(produto.preco);  
console.log(produto);
```

Saída:

```
undefined  
{}
```

Diferenças entre null e undefined

À partir do momento que eu coloco um valor na variável ela se torna válida, mesmo se o valor passe a ser undefined. Logicamente que não faz sentido você muda o valor de uma variável para undefined, mas a linguagem permite isso.

Exemplo:

```
produto.preco = 3.50;  
console.log(produto);  
produto.preco = undefined // evite atribuir undefined  
console.log(!!produto.preco);  
console.log(produto);  
produto.preco = null; // sem preço  
console.log(produto);
```

Saída:

```
{ preco: 3.5 }  
false  
{ preco: undefined }  
{ preco: null }
```