

H1 – JavaScript

H2 – Apostila de

H3 – JavaScript

H4 – Básico

1. Softwares Necessários

- Navegador (Chrome ou Edge)
 - Node JS
 - Visual Studio Code
-

2. Primeiros Comandos JavaScript

```
<script>
  window.alert('Minha primeira mensagem')
  window.confirm('Está gostando de JS?')
  window.prompt('Qual seu nome?')
</script>
```

>> <https://github.com/viniusschnoor/JavaScript-Learning/blob/main/04/ex001.html>

3. Comentando o Código

```
<script>
  window.alert('Minha primeira mensagem') //Comentário Linha Única
  window.confirm('Está gostando de JS?')
  window.prompt('Qual seu nome?')
  /*
   |   Comentário
   |   Em
   |   Diversas
   |   Linhas
  */
</script>
```

>> <https://github.com/viniusschnoor/JavaScript-Learning/blob/main/04/ex001.html>

// – Para comentário de linha única

/**/ – Para comentários de diversas linhas

4. Variáveis

1. Delimitar
2. Nomear
3. Definir tamanhos

Exemplificando, delimitar as vagas em um estacionamento, nomear as vagas para encontrar onde um carro específico está e definir o tamanho para diversas categorias de veículos, como vagas para carros, caminhões e motos.

Para colocar o carro, definir qual vaga receberá que carro com “=”. O sinal de igual sozinho, em JavaScript, chamará “recebe”. `vaga01 = carro01`

Cada vaga receberá apenas um carro, mas podemos definir que nenhum veículo entrará naquela vaga usando “null”. `vaga01 = null`

Agora com o mesmo raciocínio, dentro da memória do computador (estacionamento), delimite espaços da memória (delimitando as vagas) que receberão as informações (carros).

As vagas passam a se chamar variáveis e, ao invés de definir vagas, definimos `var` (ou `let`, no JavaScript também é permitido usar esta palavra).

Os carros passam a ser os valores, podemos definir quaisquer valores utilizando o sinal de igual (=). `var n1 = 5` >> Neste caso, definimos a variável n1 (o nome da variável é chamado de identificador) e indicamos que ela receberá o valor 5.

4.1. Identificadores

Os nomes dos identificadores devem seguir algumas regras, são elas:

1. Podem começar com `letras`, `$` ou `_`;
2. Não podem começar com `números`;
3. É possível usar `letras` ou `números`;
4. É possível usar `acentos` e `símbolos`;
5. Não podem conter `espaços`, eles normalmente são substituídos pelo sinal de sublinhado (`_`);
6. Não podem ser `palavras reservadas` (palavras que o JavaScript usa como comandos, por exemplo, `var`

5. Guardando Informações

Simplesmente, para guardar uma informação, basta definir uma variável antes da informação que será armazenada. Por exemplo, o comando `window.prompt('Qual seu nome?')` se tornará `var nome = window.prompt('Qual seu nome?')`.

```
<script>
    var nome = window.prompt('Qual seu nome?')
    window.alert('É um enorme prazer em recebê-lo, ' + nome)
</script>
```

>> <https://github.com/viniusschnoor/JavaScript-Learning/blob/main/06/ex002.html>

5.1. Tratamento de Dados

5.1.1. Inteiros e Reais

`Number.parseInt(n)` >> Converte número para número inteiro

`Number.parseFloat(n)` >> Converte número em número real

```
<script>
    var n1 = Number.parseInt(window.prompt('Digite um número')) //
    Recebendo string
    var n2 = window.prompt('Digite outro número') // Recebendo string
    var r = n1 + Number.parseInt(n2)
    window.alert('A soma dos valores é: ' + r)
    /*
        Símbolo de adição serve para concatenação e adição
        number + number >> Adição
        string + string >> Concatenação
    */
</script>
```

>> <https://github.com/viniusschnoor/JavaScript-Learning/blob/main/06/ex003.html>

Pode-se verificar que temos duas maneiras de converter o dado, antes de colocar na variável (neste caso em `n1`) ou antes de realizar qualquer atividade com a variável (neste caso, usar `n2` que é uma string, formata-la para inteiro e somar).

`Number(n)` >> Identifica automaticamente qual tipo de número e o converte

```
<script>
    var n1 = Number(window.prompt('Digite um número')) // Recebendo
    string e formatando para number
    var n2 = window.prompt('Digite outro número') // Recebendo string
    var r = n1 + Number(n2)
    window.alert('A soma dos valores é: ' + r)
    /*
        Símbolo de adição serve para concatenação e adição
        number + number >> Adição
        string + string >> Concatenação
    */
</script>
```

>> <https://github.com/viniusschnoor/JavaScript-Learning/blob/main/06/ex003.html>

5.1.2. Strings

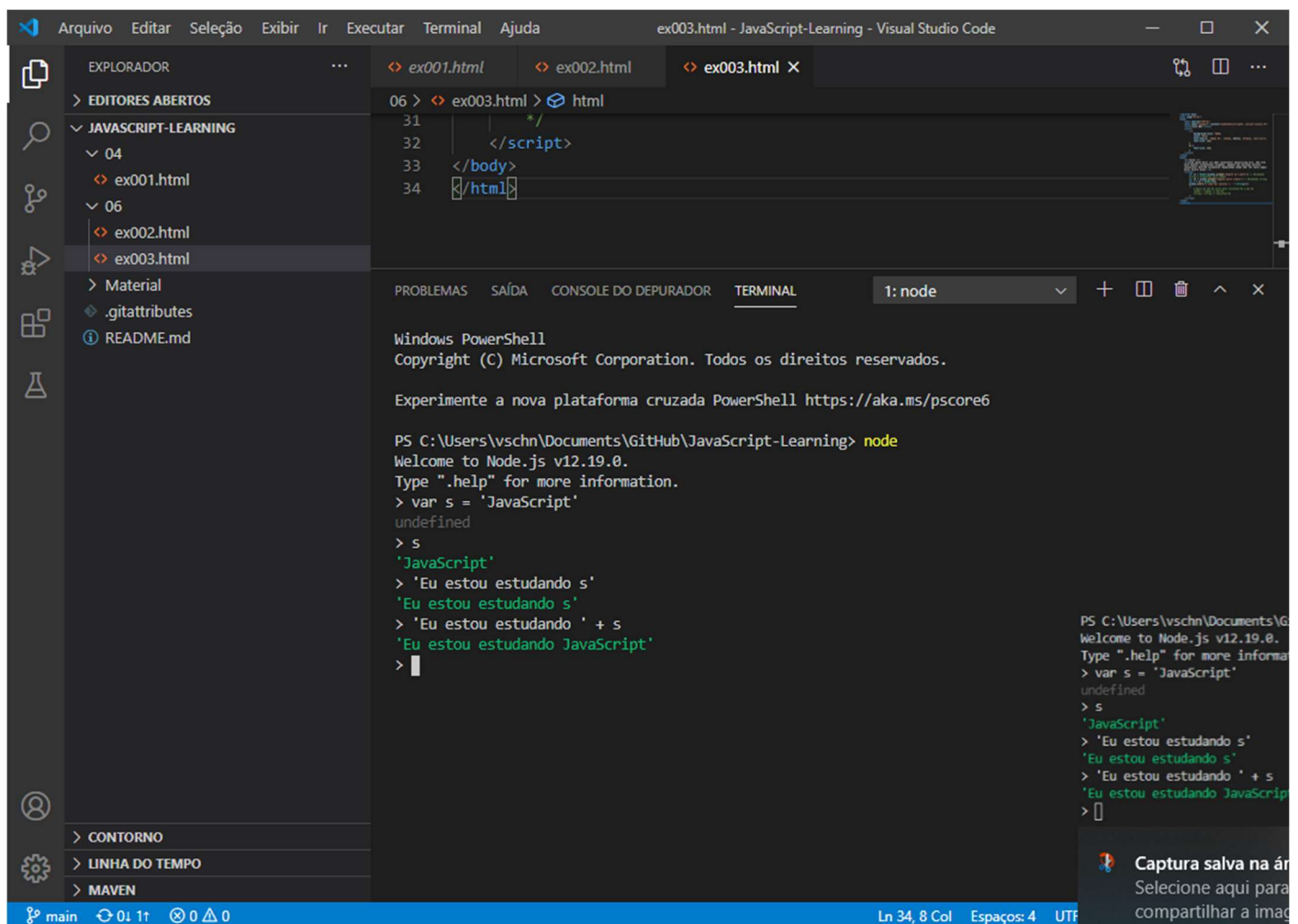
`String(n)` >> Converte número em string

`n.toString()` >> Converte número em string

```
<script>
  var n1 = Number(window.prompt('Digite um número')) // Recebendo
  string e formatando para number
  var n2 = window.prompt('Digite outro número') // Recebendo string
  var r = n1 + Number(n2)
  window.alert('A soma dos valores é: ' + String(r))
  /*
  |
  |   Símbolo de adição serve para concatenação e adição
  |   number + number >> Adição
  |   string + string >> Concatenação
  |
  */
</script>
```

>> <https://github.com/viniusschnoor/JavaScript-Learning/blob/main/06/ex003.html>

Neste caso a formatação para string não se fazia necessária, foi feita apenas para efeito de exemplificação. Veja o mesmo exemplo usando o terminal:



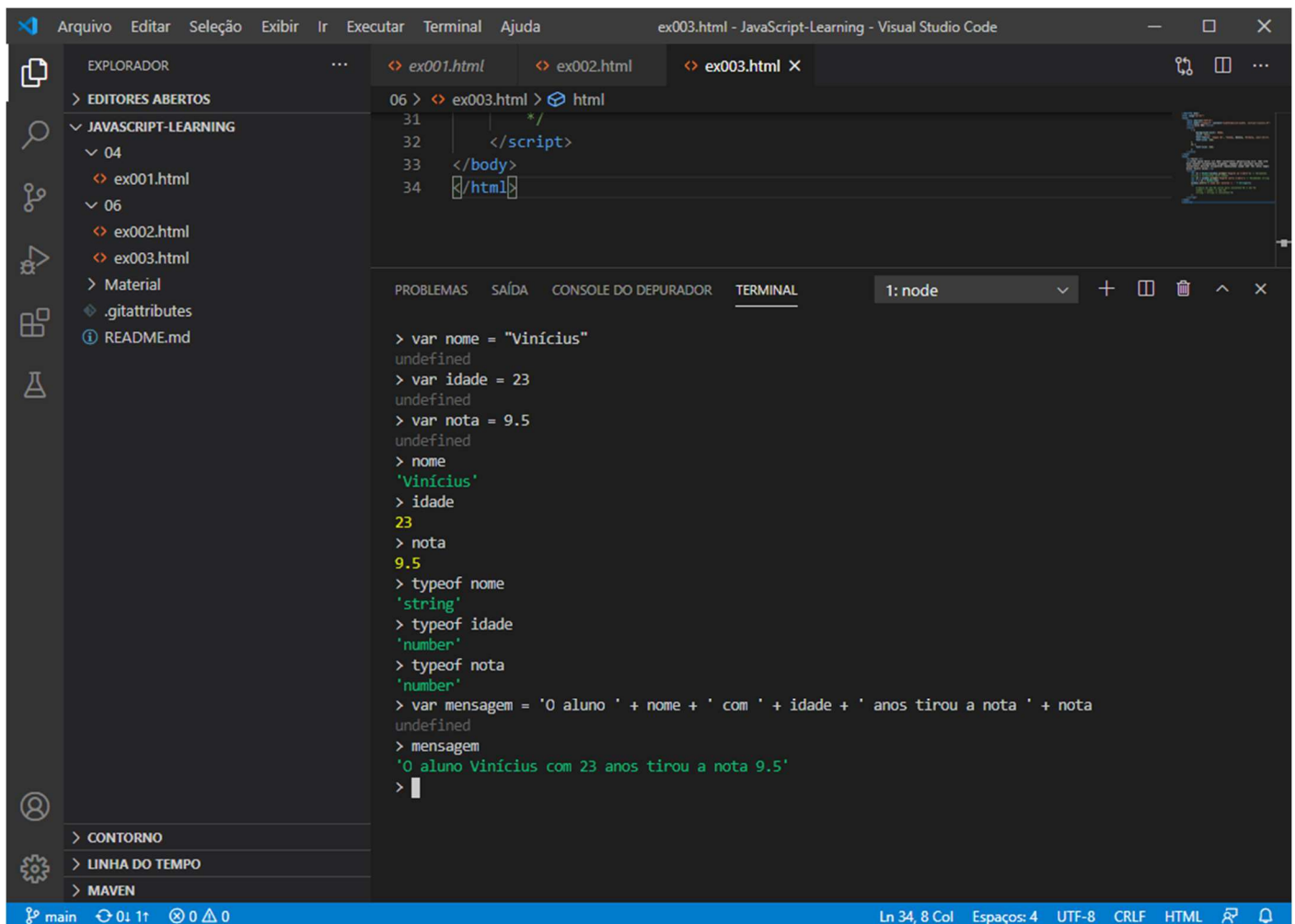
The screenshot shows the Visual Studio Code interface. The Explorer panel on the left shows the file structure of the 'JAVASCRIPT-LEARNING' project, with 'ex003.html' selected. The main editor area displays the content of 'ex003.html', which is an HTML file containing a JavaScript script. The Terminal panel at the bottom shows the output of running 'node' in a PowerShell window. The terminal output shows the Node.js version (v12.19.0) and the execution of several JavaScript commands: 'var s = \'JavaScript\'', 's', 'Eu estou estudando s', and 'Eu estou estudando ' + s'. The output of these commands is displayed in the terminal.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\Users\vschn\Documents\GitHub\JavaScript-Learning> node
Welcome to Node.js v12.19.0.
Type ".help" for more information.
> var s = 'JavaScript'
undefined
> s
'JavaScript'
> 'Eu estou estudando s'
'Eu estou estudando s'
> 'Eu estou estudando ' + s
'Eu estou estudando JavaScript'
>
```

>> `Ctrl + Shift + `` para abrir o Terminal do Visual Studio Code



```
06 > ex003.html > html
31 | /*
32 |  </script>
33 | </body>
34 | </html>
```

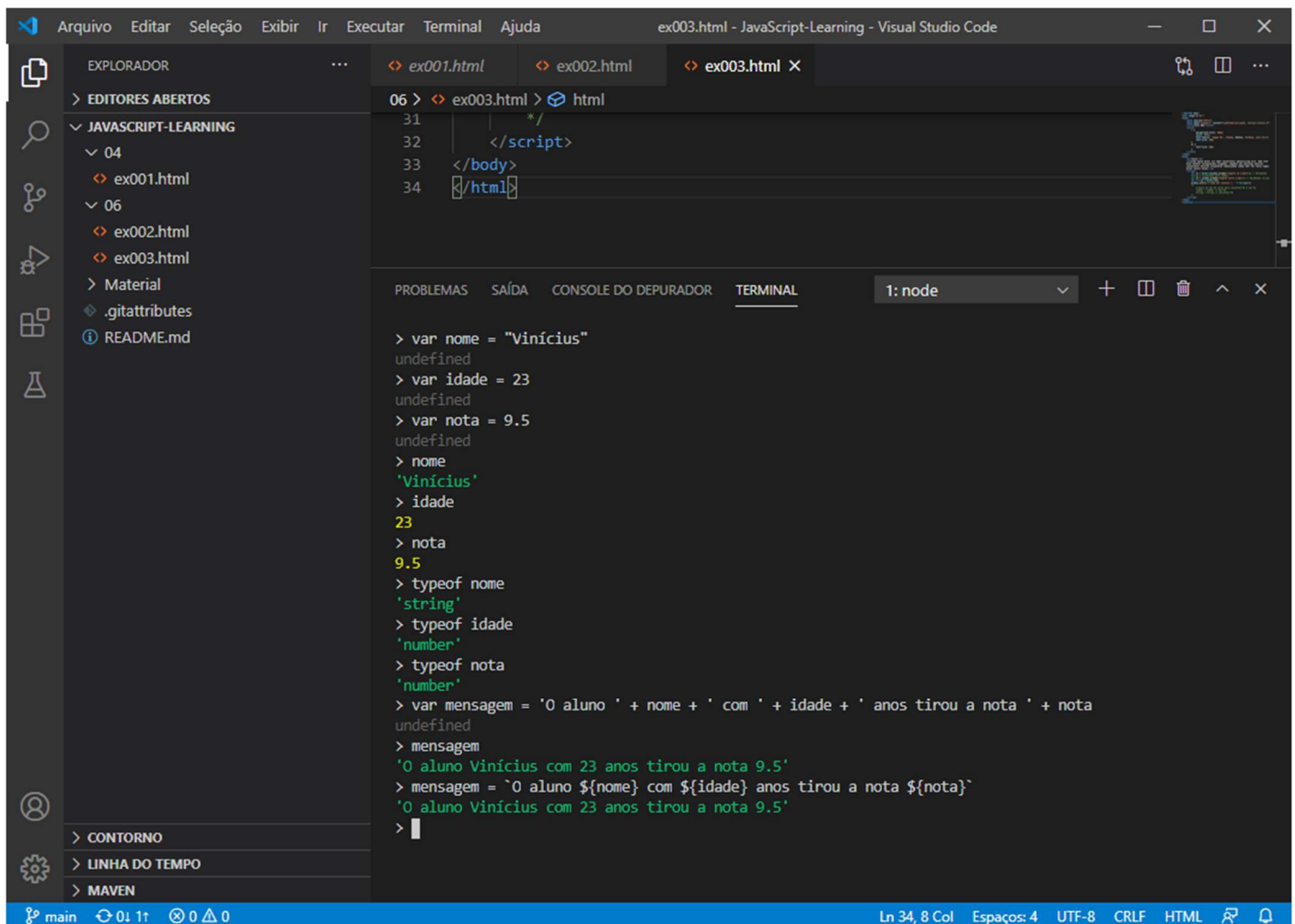
```
> var nome = "Vinicius"
undefined
> var idade = 23
undefined
> var nota = 9.5
undefined
> nome
'Vinicius'
> idade
23
> nota
9.5
> typeof nome
'string'
> typeof idade
'number'
> typeof nota
'number'
> var mensagem = 'O aluno ' + nome + ' com ' + idade + ' anos tirou a nota ' + nota
undefined
> mensagem
'O aluno Vinicius com 23 anos tirou a nota 9.5'
>
```

>> Repare a concatenação, precisa separar por aspas simples e o sinal de adição a cada alteração entre string e variável

Para resolver o problema acima, podemos utilizar da Template String utilizando crases (``) e o Place Holder (\${ }).

Por exemplo:

```
var s = 'JavaScript'
'Eu estou usando s'      // Não faz interpolação
'Eu estou usando ' + s   // Usando concatenação
'Eu estou usando ${s}'    // Usando Template String
```



>> Repare na diferença entre os comandos

```
<script>
    var n1 = Number(window.prompt('Digite um número')) // Recebendo
    string e formatando para number
    var n2 = window.prompt('Digite outro número') // Recebendo string
    var r = n1 + Number(n2)
    window.alert(`A soma entre os números ${n1} e ${n2} é ${r}`)
    /*
        Símbolo de adição serve para concatenação e adição
        number + number >> Adição
        string + string >> Concatenação
    */
</script>
```

>> <https://github.com/viniusschnoor/JavaScript-Learning/blob/main/06/ex003.html>

`s.length` >> Mostra quantidade de caracteres de uma string (sem parêntese, um atributo)

`s.toUpperCase()` >> Coloca caracteres em MAIÚSCULAS (com parêntese, um método)

`s.toLowerCase()` >> Coloca caracteres em minúsculas


```
<script>
  var nome = window.prompt('Qual seu nome?')
  document.write(`Seu nome, <strong>${nome}</strong>, tem ${nome.
length} letras<br/>`)
  document.write(`Seu nome em maiúsculas fica ${nome.toUpperCase()}`)
</script>
```

>> <https://github.com/viniusschnoor/JavaScript-Learning/blob/main/06/ex004.html>

>> Repare na inclusão de tags HTML dentro da Script

>> Repare também o uso do atributo .length e do método .toUpperCase

5.1.3. Números

`n1.toFixed(2)` >> Define duas casas decimais

`n1.toFixed(2).replace('.', ',')` >> Substitui ponto por vírgula

`n1.toLocaleString('pt-BR', {style: 'currency', currency: 'BRL'})` >> Define como moeda Real

`n1.toLocaleString('pt-BR', {style: 'currency', currency: 'USD'})` >> Define como moeda Dólar

`n1.toLocaleString('pt-BR', {style: 'currency', currency: 'EUR'})` >> Define como moeda Euro

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a project named 'JAVASCRIPT-LEARNING' with files 'ex001.html', 'ex002.html', 'ex003.html', and 'ex004.html'. The main editor displays 'ex004.html' with the following HTML code:

```

06 > ex004.html > html
1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>

```

Below the editor, the TERMINAL panel is active, showing a Node.js prompt. The output of the terminal is as follows:

```

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\Users\vschn\Documents\GitHub\JavaScript-Learning> node
Welcome to Node.js v12.19.0.
Type ".help" for more information.
> var n1 = 1345.5
undefined
> n1
1345.5
> n1.toFixed(2)
'1345.50'
> n1.toFixed(2).replace('.', ',')
'1345,50'
> n1.toLocaleString('pt-BR',{style: 'currency', currency: 'BRL'})
'R$ 1,345.50'
> n1.toLocaleString('pt-BR',{style: 'currency', currency: 'USD'})
'US$ 1,345.50'
> n1.toLocaleString('pt-BR',{style: 'currency', currency: 'EUR'})
'€ 1,345.50'
> n1.toLocaleString('pt-BR',{style: 'currency', currency: 'EUR'}).replace('.', ',')
'€ 1,345,50'
>

```

>> Terminal do Visual Studio Code

>> Não é necessário o uso do .replace() no último caso, no navegador será convertido automaticamente

6. Operadores

Tipos de Operadores:

1. Aritméticos
2. Atribuição
3. Relacionais
4. Lógicos
5. Ternário

6.1. Aritméticos

OPERADOR		RESULTADO	
5	+	2	7
	-		3
	*		10
	/		2.5

	%		1
	**		25

>> Operador de Exponenciação não existia em versões anteriores

Cuidado!

`5 + 3 / 2 = 6.5` >> Importância da Divisão no cálculo

`(5 + 3) / 2 = 4.0` >> Parênteses altera Importância do cálculo

()		
**		
*	/	%
+	-	

>> Ordem de Importância de cima para baixo

Exemplo aplicado no Terminal do Node.js:

```

Node.js
Welcome to Node.js v12.19.0.
Type ".help" for more information.
> var a = 5 + 3
undefined
> a
8
> var b = a % 5
undefined
> b
3
> var c = 5 * b ** 2
undefined
> c
45
> var d = 10 - a / 2
undefined
> d
6
> var e = 6 * 2 / d
undefined
> e
2
> var f = b % e + 4 / e
undefined
> f
3
>

```

>> Terminal do Node.js

6.2. Atribuições

6.2.1. Auto Atribuições

Relembrando a comparação com estacionamento e vagas para veículos. Para um carro estacionar em uma vaga já ocupada, é necessário retirar o carro que está ocupando a vaga para podermos colocar o outro carro.

No caso das variáveis é a mesma coisa! Se fizermos uma auto atribuição (atribuir a uma variável o valor já presente nela realizado uma operação qualquer com outro número, a variável deixa o valor anterior e passa a assumir o novo resultado.

```
Node.js
Welcome to Node.js v12.19.0.
Type ".help" for more information.
> var n = 3
undefined
> n
3
> n = n + 4
7
> n
7
> n = n - 5
2
> n
2
> n = n * 4
8
> n
8
> n = n / 2
4
> n
4
> n = n ** 2
16
> n
16
> n = n % 5
1
> n
1
>
```

>> Repare que demonstro o valor de n após cada cálculo e ele se altera a cada nova auto atribuição

6.2.2. Auto Atribuições (forma encurtada)

var n = 3	
n = n + 4	n += 4
n = n - 5	n -= 5
n = n * 4	n *= 4
n = n / 2	n /= 2
n = n ** 2	n **= 2
n = n % 5	n %= 2

>> Apenas funciona caso seja um cálculo de um valor já atribuído a ele mesmo

Não é apenas o JavaScript que utiliza esta forma de auto atribuição, Java, C, Python, php, por exemplo, também aceitam essa forma de auto atribuição.

```
Node.js
Welcome to Node.js v12.19.0.
Type ".help" for more information.
> var n = 3
undefined
> n
3
> n += 4
7
> n
7
>
```

>> Exemplo aplicado no Terminal do Node.js

6.2.3. Incremento

<code>var x = 5</code>	
<code>x = x + 1</code>	<code>x ++</code>
<code>x = x - 1</code>	<code>x --</code>

>> Sempre utilizar o Incremento quando forem situações de adição ou subtração de apenas uma unidade

```
Node.js
Welcome to Node.js v12.19.0.
Type ".help" for more information.
> var x = 0
undefined
> x
0
> x ++
0
> x
1
> x --
1
> x
0
> ++ x
1
> x
1
> -- x
0
> x
0
> _
```

>> Pré e Pós-Incremento

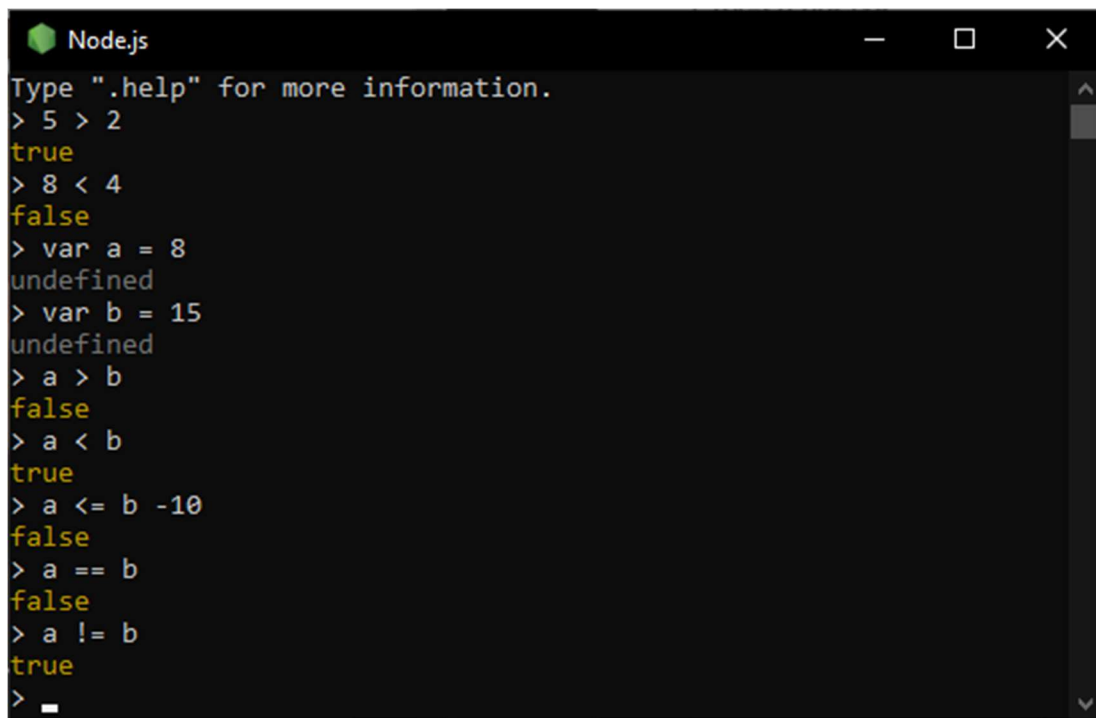
Acompanhe pelas linhas do terminal acima que, utilizando o pós-incremento, o valor é atribuído apenas depois do cálculo, enquanto, no pré-incremento, o valor é atribuído antes do fim do cálculo.

6.3. Relacionais

OPERADOR		RESULTADO	
5	>	2	true
7	<	4	false
8	>=	8	true
9	<=	7	false
5	==*	5	true
4	!=	4	false

>> Resultados de Operações Relacionais sempre serão boolean (true ou false)

* Dois sinais de igual



```
Node.js
Type ".help" for more information.
> 5 > 2
true
> 8 < 4
false
> var a = 8
undefined
> var b = 15
undefined
> a > b
false
> a < b
true
> a <= b - 10
false
> a == b
false
> a != b
true
> _
```

>> Repare que também pode ser utilizada das variáveis

Exemplos com outros tipos de dados:

```
preco >= 1,200.00 // O preço é maior ou igual a 1.250,00?
idade < 18 // A idade é menor que 18?
apostila == 'Apostila de JavaScript' // A apostila é JavaScript?
n1 != n2 // n1 é diferente de n2?
```

6.3.1. Identidade

OPERADOR		RESULTADO	
5	==	5	true
5	==	'5'	true
5	===	'5'	false
5	===	5	true

>> Ao usar três sinais consecutivos de igual, o JavaScript também confere o tipo do dado, verificando se os dados são IDÊNTICOS

```
Node.js
> 5 == 5
true
> var x = 5
undefined
> var y = '5'
undefined
> typeof x
'number'
> typeof y
'string'
> x == y
true
> x === y
false
> x != y
false
> x !== y
true
>
```

>> Node.js

6.4. Lógicos

Tipos de Operadores em exemplos em frases comuns:

NEGAÇÃO: Quero um lápis que não seja verde.

CONJUNÇÃO: Quero um lápis roxo e um lápis laranja.

DISJUNÇÃO: Quero um lápis preto ou um lápis cinza.

6.4.1. Negação (!)

!	true	false	Algo não verdadeiro, é falso
	false	true	Algo não falso, é verdade

6.4.2. Conjunção (&&)

true	&&	true	true	Os dois itens atendem o que preciso, então verdadeiro
true		false	false	Um dos itens não atende o que preciso, então falso
false		true	false	Um dos itens não atende o que preciso, então falso
false		false	false	Nenhum dos itens atende o que preciso, então falso

6.4.3. Disjunção (||)

true		true	true	Os dois itens me atendem, então verdadeiro
true		false	true	Um dos itens me atende, então verdadeiro
false		true	true	Um dos itens me atende, então verdadeiro
false		false	false	Nenhum dos itens me atende, então falso

```

Node.js
undefined
> var b = 8
undefined
> true && false
false
> true && true
true
> false || false
false
> true || false
true
> a > b && b % 2 == 0
false
> a <= b || b / 2 == 2
true
> _

```

>> Em prática no Node.js

Exemplos:

idade >= 18 && idade < 70 // Idade está entre 18 e 69?

estado == 'SP' || estado == 'RJ' // O estado é SP ou RJ?

salário > 1500 && sexo != 'M' // Salário está acima de 1.500 e não é homem?

()	**	/ [...]	Aritméticos
>	<	>= [...]	Relacionais*
!	Lógicos **		
&&			

* Relacionais não possui ordem para ser realizado, qual aparecer primeiro, será feito primeiro

** Os lógicos possuem a ordem de realização como descrita na tabela

6.5. Ternário [teste lógico] ? [resultado se verdadeiro] : [resultado se falso]

Exemplo:

média >= 7.0 ? "Aprovado" : "Reprovado" // Se média for maior que 7.0, foi aprovado, se não for, foi reprovado.


```
Node.js
Welcome to Node.js v12.19.0.
Type ".help" for more information.
> var média = 5.5
undefined
> média > 7 ? "Aprovado" : "Reprovado"
'Reprovado'
> média += 3
8.5
> média > 7 ? "Aprovado" : "Reprovado"
'Aprovado'
> _
```

```
Node.js
> var idade = 19
undefined
> var r = idade >= 18 ? "MAIOR" : "MENOR"
undefined
> r
'MAIOR'
> _
```