

H1 – JavaScript

H2 – Apostila de

H3 – JavaScript

H4 – Básico

Disponível em >> <https://github.com/viniusschnoor/JavaScript-Learning/blob/main/Material/Apostila%20Digitada.pdf>

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

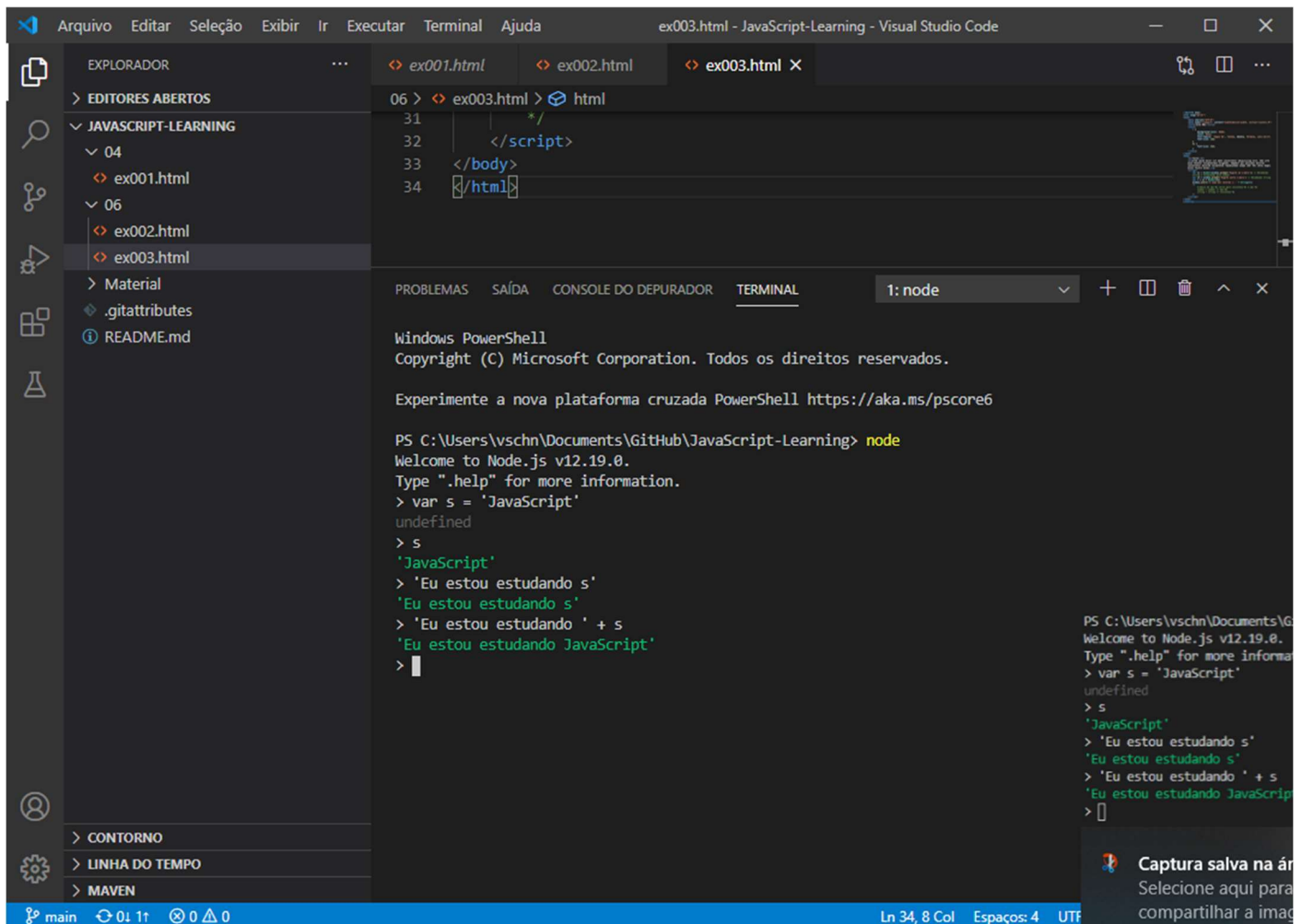
`String(n)` >> Converte número em string

`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. On the left, the Explorer sidebar shows a project named 'JAVASCRIPT-LEARNING' with files 'ex001.html', 'ex002.html', and 'ex003.html'. The main editor area displays the content of 'ex003.html', which contains a JavaScript script. Below the editor, the TERMINAL panel is active, showing the output of running the script in a Node.js environment. The output demonstrates string concatenation and addition using the '+' operator. The status bar at the bottom indicates the current line and column (Ln 34, 8 Col) and the encoding (UTF-8).

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

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL 1: node

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.

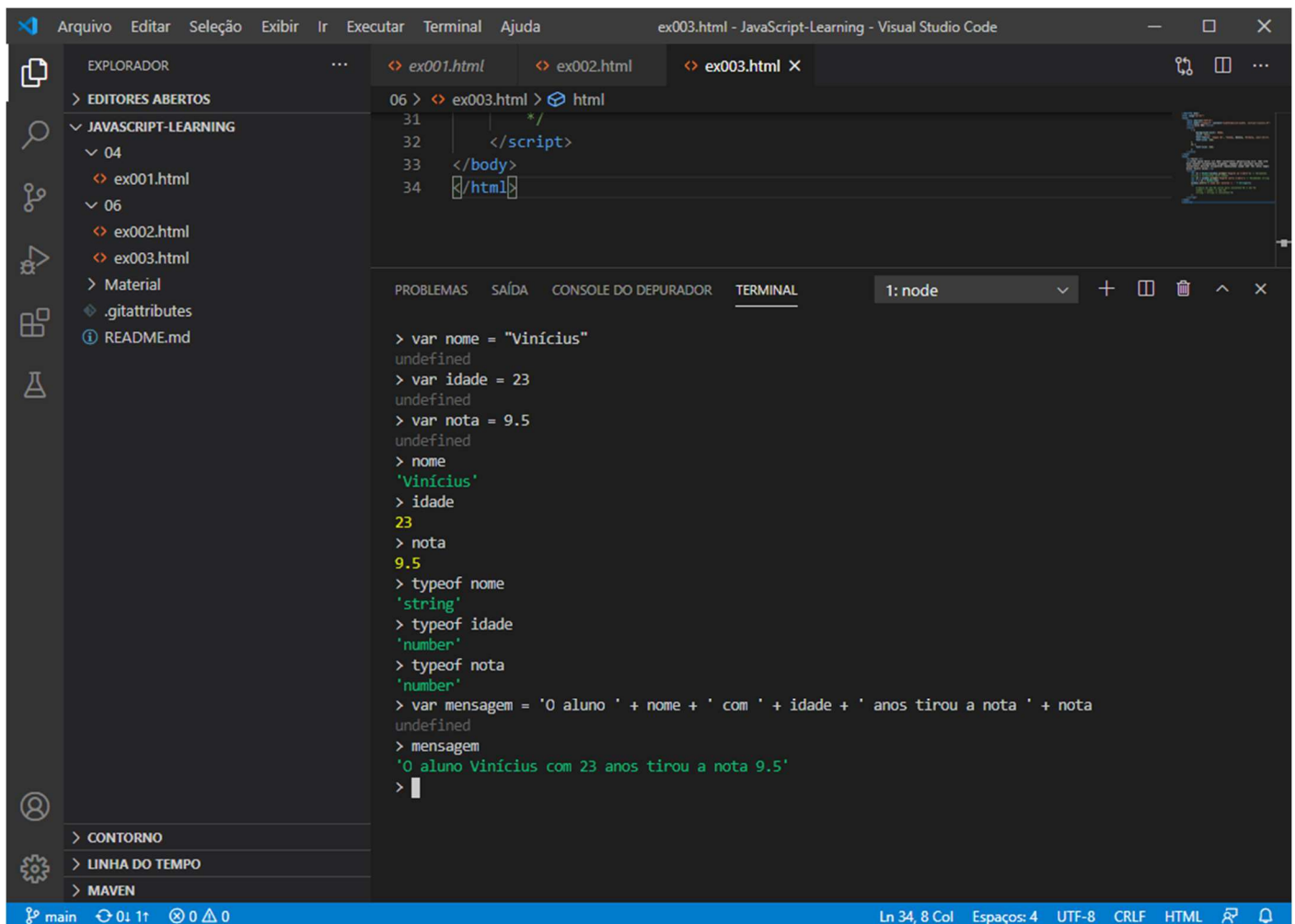
```
> var s = 'JavaScript'
undefined
> s
'JavaScript'
> 'Eu estou estudando s'
'Eu estou estudando s'
> 'Eu estou estudando ' + s
'Eu estou estudando JavaScript'
> 
```

PS C:\Users\vschn\Documents\G...
Welcome to Node.js v12.19.0.
Type ".help" for more informat...
> var s = 'JavaScript'
undefined
> s
'JavaScript'
> 'Eu estou estudando s'
'Eu estou estudando s'
> 'Eu estou estudando ' + s
'Eu estou estudando JavaScript'
>

Captura salva na ár...
Selecione aqui para...
compartilhar a imag...

main 0 11 0 0 0 0 Ln 34, 8 Col Espaços: 4 UTF

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



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

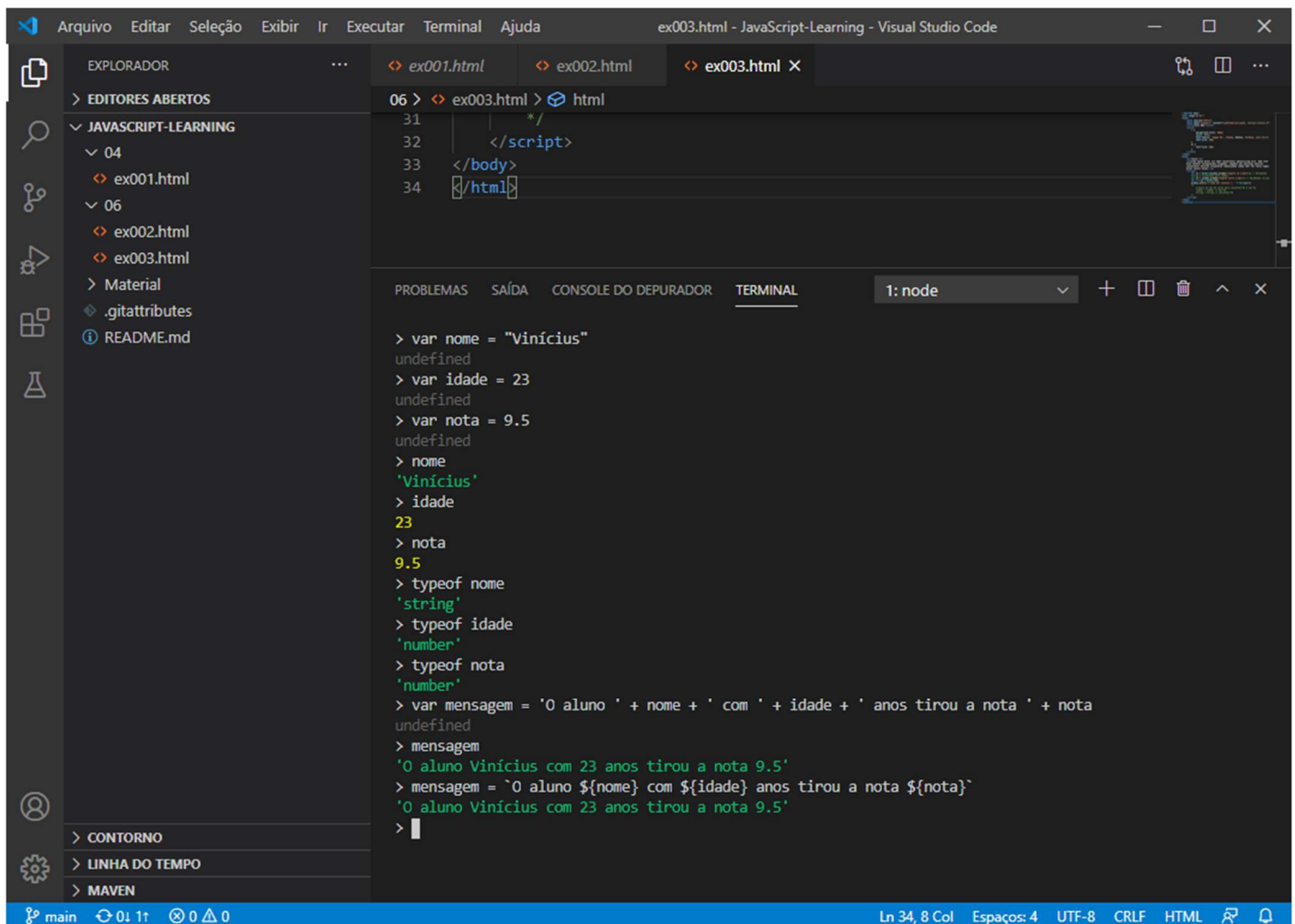
```
> var nome = "Vinicius"
undefined
> var idade = 23
undefined
> var nota = 9.5
undefined
> nome
'Vinicius'
> idade
23
> nota
9.5
> typeof nome
'string'
> typeof idade
'number'
> typeof nota
'number'
> var mensagem = 'O aluno ' + nome + ' com ' + idade + ' anos tirou a nota ' + nota
undefined
> mensagem
'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 shows a Windows PowerShell session with the following output:

```
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
>

```

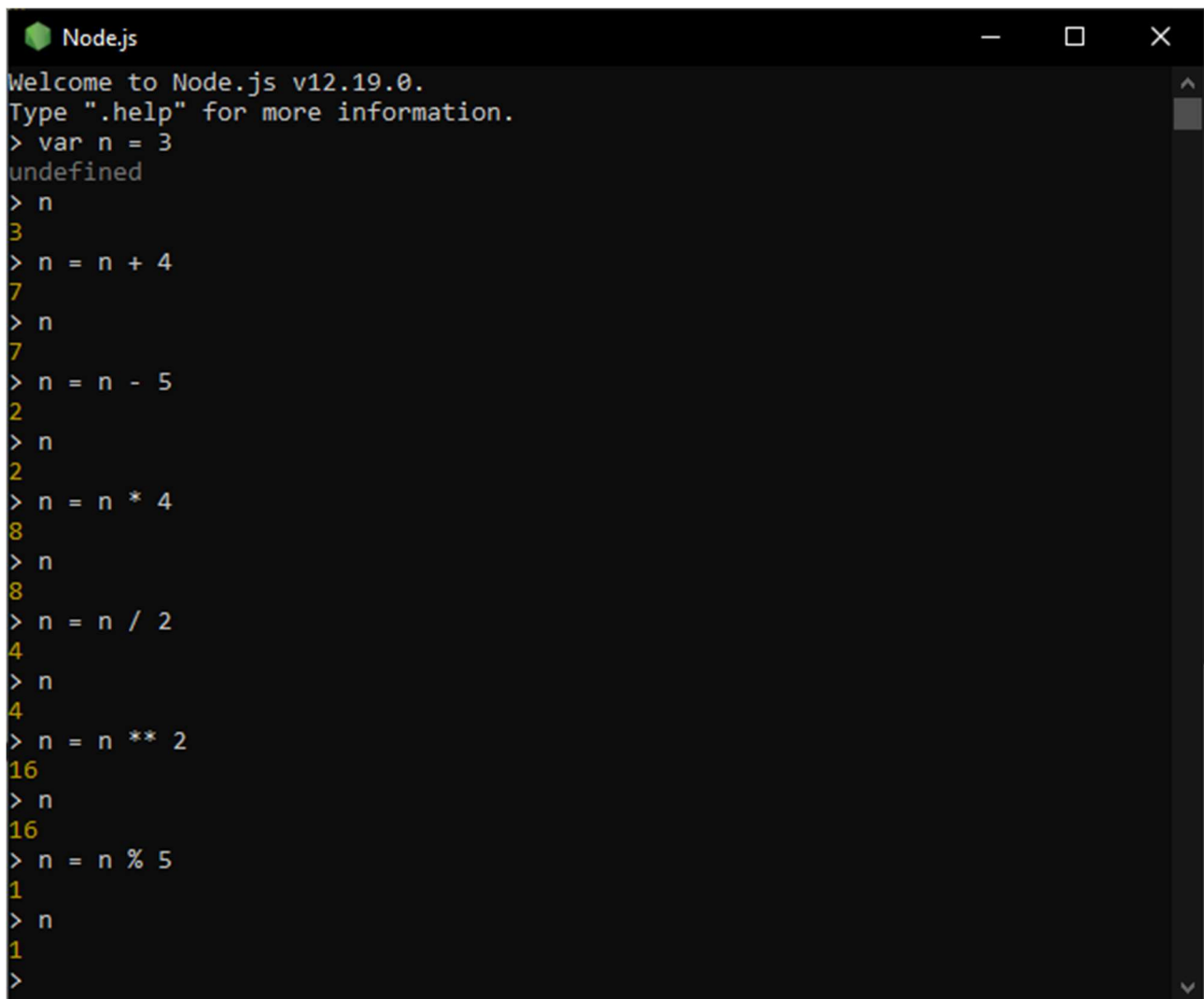
>> Terminal do Node.js

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

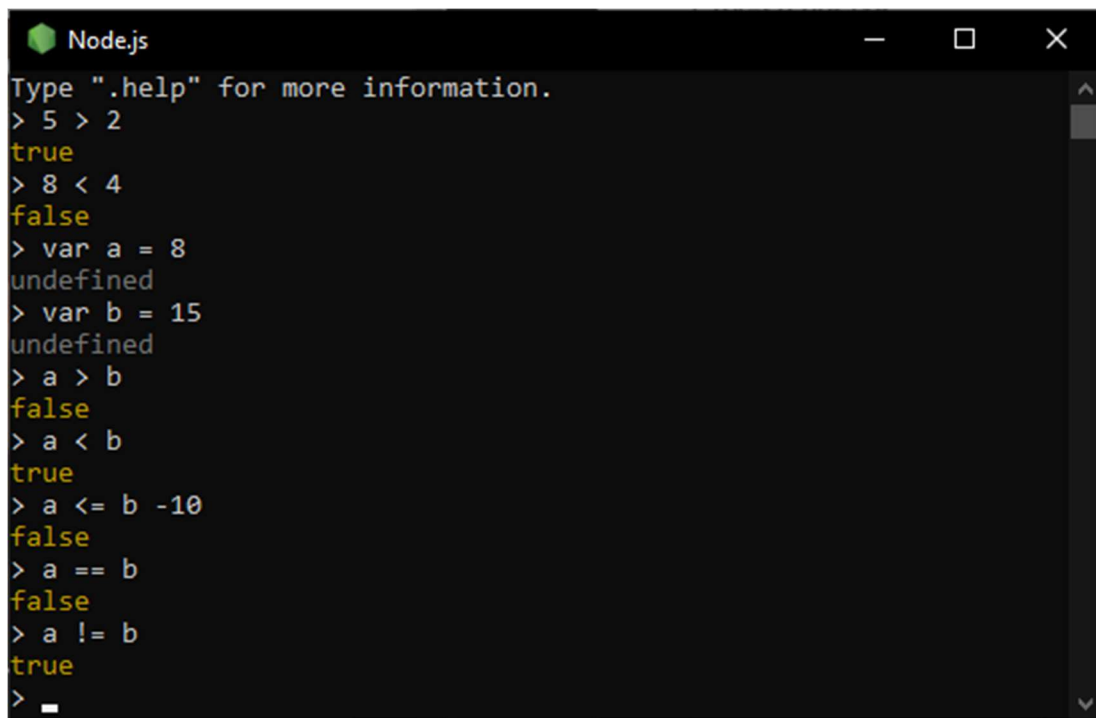
Acompanhe pelas linhas do terminal acima que, utilizando o pós-incremento, o valor é atribuído apenas depois do cálculo, enquanto, no pré-incremento, o valor é atribuído antes do fim do cálculo.

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'
>
```

7. DOM

Document Object Model >> Conjunto de Objetos do Navegador que dará acesso aos componentes internos do site.

7.1. Árvore DOM

```

1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width,
6          initial-scale=1.0">
7      <title>Aprendendo DOM</title>
8      <style>
9          body {
10              background-color: #333;
11              color: #ddd;
12              font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
13              sans-serif;
14              font-size: 1em;
15          }
16      </style>
17  </head>
18  <body>
19      <h1>Início de Estudos com DOM</h1>
20      <p>Aqui será o resultado</p>
21      <p>Aprendendo a usar <strong>DOM</strong> em JavaScript</p>
22      <div>Clique em mim</div>
23  </body>
24  </html>

```

>> <https://github.com/viniusschnoor/JavaScript-Learning/blob/main/09/ex005.html>

ELEMENTOS:

- 7.1.1. window (a raiz da árvore, a janela do navegador)
 - 7.1.1.1. location (URL, página atual)
 - 7.1.1.2. document (documento atual)
 - 7.1.1.2.1. html (parent de head e body)
 - 7.1.1.2.1.1. head (child de html)
 - 7.1.1.2.1.1.1. meta
 - 7.1.1.2.1.1.2. title
 - 7.1.1.2.1.2. body (child de html)
 - 7.1.1.2.1.2.1. h1
 - 7.1.1.2.1.2.2. p
 - 7.1.1.2.1.2.3. p
 - 7.1.1.2.1.2.3.1. strong
 - 7.1.1.2.1.2.4. div
 - 7.1.1.3. history (de onde você veio, para onde vai...)

7.2. Selecionando

Seguem 5 métodos de acesso aos elementos:

1. Por marca `<p>`
 - `getElementsByTagName()`
2. Por ID `<p id="name">`
 - `getElementById()`
3. Por Nome `<p id="teste" name="campo01">`
 - `getElementsByName()` >> repare no uso do Elements no plural, uso recomendado dos `[]`
4. Por Classe `<p id="teste" name="campo01" class="linha09">`
 - `getElementByClassName()`
5. Por Seletor (CSS) `'div#teste'`
 - `querySelector()`
 - `querySelectorAll()`

```
1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width,
6     initial-scale=1.0">
7   <title>Aprendendo DOM</title>
8   <style>
9     body {
10       background-color: #333;
11       color: #ddd;
12       font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
13         sans-serif;
14       font-size: 1em;
15     }
16   </style>
17 </head>
18 <body>
19   <h1>Início de Estudos com DOM</h1>
20   <p>Aqui será o resultado</p>
21   <p>Aprendendo a usar <strong>DOM</strong> em JavaScript</p>
22   <div>Clique em mim</div>
23   <script>
24     var p1 = window.document.getElementsByTagName('p')[0]
25     window.document.write('A linha 0 (primeira linha) está
26       escrito: ' + p1.innerText)
27   </script>
28 </body>
29 </html>
```

>> Repare que a fórmula segue a árvore DOM acima descrita

>> window, document e, no html, é usado para capturar o elemento pela tag de parágrafo `[p]`

>> O `[0]` é utilizado na maioria das vezes para indicar qual tag `p` será capturada, se não, ele captura todas as tags `p`

```

<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
  <title>Aprendendo DOM</title>
  <style>
    body {
      background-color: #333;
      color: #ddd;
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
        sans-serif;
      font-size: 1em;
    }
  </style>
</head>
<body>
  <h1>Início de Estudos com DOM</h1>
  <p>Aqui será o resultado</p>
  <p>Aprendendo a usar <strong>DOM</strong> em JavaScript</p>
  <div>Clique em mim</div>
  <script>
    var p1 = window.document.getElementsByTagName('p')[0]
    p1.style.color = 'blue'
  </script>
</body>
</html>

```

Início de Estudos com DOM

Aqui será o resultado

Aprendendo a usar **DOM** em JavaScript

Clique em mim

>> <https://github.com/viniusschnoor/JavaScript-Learning/blob/main/09/ex005.html>

```

<div name="msg" class="msg" id="msg">Clique em mim</div>
<script>
  var corpo = window.document.body
  var p1 = window.document.getElementsByTagName('p')[1]
  /*
  var d = window.document.getElementById('msg')
  d.style.background = 'green'
  d.innerText = 'Estou aguardando...'
  window.document.getElementById('msg').innerText = 'Olá'
  */
  var d = window.document.getElementsByName('msg')[0]
</script>

```

>> Você pode capturar o componente por nome, classe ou ID

```

<div name="msg" class="msg" id="msg">Clique em mim</div>
<script>
  var corpo = window.document.body
  var p1 = window.document.getElementsByTagName('p')[1]
  /*
  var d = window.document.getElementById('msg')
  d.style.background = 'green'
  d.innerText = 'Estou aguardando...'
  window.document.getElementById('msg').innerText = 'Olá'
  */
  var d = window.document.querySelector('div#msg')
  d.style.color = 'blue'
</script>

```

>> Com o `querySelector` você seleciona diretamente a classe ou ID do objeto

8. Eventos DOM

Imagine uma `<div>`, qualquer coisa que possa acontecer com a `<div>`, como um “evento de mouse”, se chama evento.

Quando o mouse “entra” na `<div>`, o evento que ocorre chama-se “`mouseenter`”. Quando o mouse se movimenta por dentro da `<div>`, por exemplo, chama-se “`mousemove`”. Um click sem soltar, chama-se “`mousedown`” e, o momento que solta o botão do mouse chama-se “`mouseup`”.

Ainda possuí o evento de clique rápido, o “`click`”, e, ao tirar o mouse de dentro da `<div>`, temos o “`mouseout`”.

Veja uma lista completa dos “JavaScript DOM Events”:

<https://developer.mozilla.org/pt-BR/docs/Web/Events>

8.1. Funções (quase um 9.)

Para começarmos a tratar de eventos, precisa-se entender o que são as funções, gostaria de colocar funções em um tópico só dela, porém, por ser necessário saber antes de tratar os eventos DOM, vamos começar a entender as funções.



Funções são um conjunto de códigos que serão executados apenas quando eventos ocorrerem. Esse conjunto de linhas/códigos são chamados de bloco. Estes blocos não executam automaticamente como estava ocorrendo até agora com nossos códigos.

O primeiro passo então para controlar o momento que os códigos JavaScripts são incluindo estes códigos em blocos. Os blocos são delimitados por chaves (`{}`) e nomeados por `functions`.

Existem dois tipos de funções:

```
function {
```

```
BLOCO
```

```
} // Função Anônima
```

```
function () {
```

```
BLOCO
```

```
} // Função Nomeada, precisa nomear para o método funcionar
```

Geralmente, os nomes das funções, são ações que são possíveis realizar. Você coloca o nome da ação e abre e fecha parênteses. Opcionalmente também é possível parametrizar esta ação colocando os parâmetros dentro destes parênteses.

Temos dois métodos possíveis, a primeira é definindo as ações no HTML e rodando a ação no JavaScript:

```
1 <!DOCTYPE html>
2 <html lang="ptbr">
3 > <head> ...
8 </head>
9 <body>
10 | <div id="area" onclick="clicar()" onmouseenter="entrar()" onmouseout="sair()">
11 |   Interaja
12 | </div>
13 | <script type="text/javascript" src="js/js.js"></script>
14 </body>
15 </html>
```

>> Primeiro definimos as ações na Tag HTML

```
1 var a = window.document.getElementById('area')
2 function clicar() {
3   /*
4    | 1º OnClick na HTML
5    | 2º Criado a Função no JS
6    */
7   a.innerHTML = 'Clicou!'
8 }
9 function entrar() {
10  a.innerHTML = 'Entrou!'
11 }
12
13 function sair() {
14  a.innerHTML = 'Saiu!'
15 }
```

>> Chamamos as ações através das functions e mostramos quais atividades deverão ser realizadas

```
< ex006.html # style.css JS js.js x
10 > js > JS js.js > ...
1  var a = window.document.getElementById('area')
2  function clicar() {
3    /*
4     | 1º OnClick na HTML
5     | 2º Criado a Função no JS
6     */
7    a.innerHTML = 'Clicou!'
8    a.style.backgroundColor = 'red'
9  }
10 function entrar() {
11  a.innerHTML = 'Entrou!'
12  a.style.backgroundColor = 'green'
13 }
14
15 function sair() {
16  a.innerHTML = 'Saiu!'
17  a.style.backgroundColor = 'black'
18 }
19 }
```


A segunda é utilizando de *listeners* (ouvintes) pelo próprio JavaScript:

```
1 <!DOCTYPE html>
2 <html lang="ptbr">
3 > <head> ...
8 </head>
9 <body>
10 |   <div id="area">
11 |     Interaja
12 |   </div>
13 |   <script type="text/javascript" src="js/js.js"></script>
14 </body>
15 </html>
```

>> Não é necessário indicar nada referente aos eventos no HTML

```
1 // 1ª Define a Variável
2 var a = window.document.getElementById('area')
3
4 //Adiciona os listeners de acordo comos eventos possíveis no
  código
5 a.addEventListener('click', clicar)
6 a.addEventListener('mouseenter', entrar)
7 a.addEventListener('mouseout', sair)
8
9 //Chama os eventos e define as ações
10 function clicar() {
11 |   a.innerHTML = 'Clicou!'
12 |   a.style.backgroundColor = 'red'
13 | }
14 function entrar() {
15 |   a.innerHTML = 'Entrou!'
16 |   a.style.backgroundColor = 'green'
17 | }
18
19 function sair() {
20 |   a.innerHTML = 'Saiu!'
21 |   a.style.backgroundColor = 'black'
22 | }
```

>> Criando os *listeners*, basta programa-los para prestarem atenção a todos os eventos que precisa que sejam o gatilho das ações

9. Condições

9.1. Condições Simples

```
if (condição) {true}
```

```
11 > js > JS js008.js > ...
1  var vel = 60.1
2  console.log(`A velocidade do seu carro é ${vel} km/h`)
3  if (vel > 60) {
4      console.log(`Você ultrapassou a velocidade permitida e foi
      MULTADO!`)
5  }
6  console.log(`Dirija sempre utilizando cinto de segurança`)
```

PROBLEMAS SAÍDA ... Node.js

Info: Start process (11:44:46)
A velocidade do seu carro é 60.1 km/h
Você ultrapassou a velocidade permitida e foi MULTADO!
Dirija sempre utilizando cinto de segurança
Info: End process (11:44:47)

>> Utilizando a extensão node exec instalada no Virtual Studio Code

9.2. Condições Compostas

```
if (condição) {true} else {false}
```

```
11 > JS js009.js > ...
1  var pais = 'EUA'
2  console.log(`Vivendo no ${pais}...`)
3  if (pais == 'Brasil') {
4      console.log(`Sou Brasileiro`)
5  } else {
6      console.log(`Sou Estrangeiro`)
7  }
```

PROBLEMAS SAÍDA ... Node.js

Info: Start process (11:49:10)
Vivendo no EUA...
Sou Estrangeiro
Info: End process (11:49:14)

>> Lembre-se que pode utilizar qualquer operador

Segue um exemplo disponibilizado no GitHub:

Sistema de Multas

Velocidade do carro: km/h

Sua velocidade é de **61 km/h**

Dirija sempre com cinto de segurança

Você está acima da velocidade e foi **multado**

>> HTML: <https://github.com/viniusschnoor/JavaScript-Learning/blob/main/11/ex010.html>

```
ex010.html JS js010.js X
11 > js > JS js010.js > verifySpeed
1  var velocidadeDigitada = document.getElementById('txtvel')
2  var verificarVelocidade = document.querySelector('input#btnver')
3  var res = document.getElementById('result')
4
5  verificarVelocidade.addEventListener('click', verifySpeed)
6
7  function verifySpeed() {
8      var vel = Number(velocidadeDigitada.value)
9      res.innerHTML = `Sua velocidade é de <strong>${vel} km/h</strong>`
10     res.innerHTML += `<p>Dirija sempre com cinto de segurança</p>`
11     if (vel > 60) {
12         res.innerHTML += `Você está acima da velocidade e foi <strong>multado</strong>`
13     }
14 }
```

>> JavaScript: <https://github.com/viniusschnoor/JavaScript-Learning/blob/main/11/js/js010.js>

9.3. Condições Aninhadas

As condições aninhadas são quando você utiliza de condições dentro de uma das saídas. Por exemplo, se a primeira condição retornar verdadeiro, o código realizará um bloco de comandos, caso seja falso, ao invés de rodar um bloco de comandos, o código passa a analisar outra condição, aninhando condições.

```
if (condição01) { true } else { if (condição02) { true } else { false } }
```

Lembrando que também podemos aninhar ainda mais, criando, por exemplo, uma terceira condição no resultado falso da segunda condição.

```
1  var idade = 17
2  if (idade < 16) {
3      console.log('Não Vota')
4  } else {
5      if (idade < 18) {
6          console.log('Voto Opcional')
7      }
8  }

1  var idade = 17
2  if (idade < 16) {
3      console.log('Não Vota')
4  } else if (idade < 18) {
5      console.log('Voto Opcional')
6  }
```

>> Repare que podemos usar a segunda condição de duas formas, abrindo um novo bloco ou utilizando `else if`

>>

```
12 > JS js011.js > ...
1 | var idade = 68
2 | console.log(`Você tem ${idade} anos.`)
3 | if (idade < 16) {
4 |     console.log('Não Vota')
5 | } else if (idade < 18 || idade > 65) {
6 |     console.log('Voto Opcional')
7 | } else {
8 |     console.log('Voto Obrigatório')
9 | }
```

PROBLEMAS SAÍDA ... Node.js

Info: Start process (13:50:17)
Você tem 68 anos.
Voto Opcional
Info: End process (13:50:17)

>> Repare a utilização do "ou" (||)

```
1 | var horaAtual = 5
2 | console.log(`Agora são ${horaAtual} horas`)
3 | if (horaAtual >= 6 && horaAtual < 12) {
4 |     console.log('Bom dia!')
5 | } else if (horaAtual >= 12 && horaAtual < 18) {
6 |     console.log('Boa tarde!')
7 | } else {
8 |     console.log('Boa noite!')
9 | }
```

PROBLEMAS SAÍDA ... Node.js

Info: Start process (14:00:03)
Agora são 5 horas
Boa noite!
Info: End process (14:00:03)

>> Exemplo simples de como utilizar as condições Aninhadas.

```
12 > JS js012.js > ...
1  var agora = new Date()
2  var horaAtual = agora.getHours()
3  var minutoAtual = agora.getMinutes()
4  console.log(`Agora são ${horaAtual} horas`)
5  if (horaAtual >= 6 && horaAtual < 12) {
6      console.log('Bom dia!')
7  } else if (horaAtual >= 12 && horaAtual < 18) {
8      console.log('Boa tarde!')
9  } else {
10     console.log('Boa noite!')
11 }
12 console.log(`Horário: ${horaAtual}:${minutoAtual}`)
```

PROBLEMAS SAÍDA ... Node.js

Info: Start process (14:04:11)
Agora são 14 horas
Boa tarde!
Horário: 14:4
Info: End process (14:04:12)

>> Por título de curiosidade, utilize `new Date()` e `.getHours()` para retornar a hora atual

9.4. Condições Múltiplas

Diferente de todas as demais condições, esta possui não só a possibilidade de `true` e `false`, ela trás a possibilidade de quaisquer outros valores.

```
switch (expressão) { // Início do Bloco Principal
case valor01:
    bloco01
    break
case valor02:
    bloco02
    break
case valor03:
    bloco03
    break
default
    bloco default
    break
}
```

Apenas para não deixar de citar que, no final de cada `case`, faz-se obrigatório o uso de `break`. No último case, o `default`, o uso do `break` é opcional, porém, para facilitar, vamos usá-lo também aqui. O `break` simplesmente indica para o código não continuar lendo a condição, para ele seguir para o final, finalizar o `switch` e continuar o programa.