JAVASCRIPT

Conteúdo(HTML) - Estilo (CSS) - Interação (Javascript)

ECMA- É uma empresa europeia de padronização (estilo a ISO dos EUA). Logo, ECMAscript, nada mais é do que a versão padronizada do Javascript.

Node.js é uma máquina que roda Javascript fora do Navegador. Isso ocorre por que o Node.js usa o motor de processamento(V8) disponibilizado pelo Google Chrome.

Algumas versões famosas de ECMAScript são a 1.0 (1997), 2.0 (1998),3.0 (1999), ES5 (2009- a mais famosa e mais usada pelos navegadores), ES6 (2015), ES 2016, ES 2017, ES 2018.

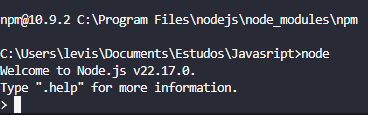
Tecnologias com base no Javascript: jQuery, Angular.js(google), React(facebook). Vue.js, Electron(github, discord...), Ionic, Cordova.

Para usar o Node.js através do prompt de comando do Windows no terminal do vscode:

Selecione o prompt de comando no vscode:

"terminal.integrated.defaultProfile.windows": "Command Prompt"

Agora, digite ‘npm’, após isso digite ‘node’ e o node irá ativar no terminal. Para sair é só digitar ‘.exit’



Tipos de Dados e Variáveis

Variáveis são como vagas para carros em shopping(com identificação), aonde o cliente poderá ‘armazenar’ seu automóvel na vaga designada a ele.

‘=’ significa ‘recebe’ no Javascript. Leia sempre dessa maneira.

‘null’ significa ‘nulo’ no Javascript. Se atribuído a alguma variável, nenhum dado poderá usar essa ‘vaga’.

Para utilizar variáveis devemos usar a palavra ‘var’ (ou ‘let’) e identificar elas.

No Javascript é possível usar a " ", ‘ ‘, ` ` ao identificar os dados atribuídos as variáveis.

As variáveis precisam de identificadores(nomes), no entanto, você precisa saber algumas regras para nomear essas variáveis:

* Podem começar com letra, $ ou \_
* Não podem começar com números
* É possível usar letras ou números
* É possível usar acentos e símbolos
* Não podem conter espaços
* Não podem ser palavras reservadas(usadas pelo próprio Javascript, ex: alert, var, array)

Dicas para ajudar na criação da nomenclatura da variável:

* Maiúsculas e minúsculas fazem diferença
* Tente escolher nomes coerentes para variáveis
* Evite se tornar um ‘programador alfabeto’ ou um ‘programador contador’

Alguns tipos de dados(Data Types) são:

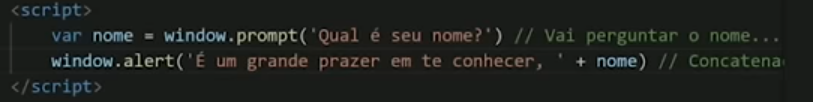
* Numbers = 5, 18, -12, 0.5, -15.9, 3.14
* Boolean = true, false
* String = ‘Javascript’, `Maria`, "Google"
* Null
* Undefined
* Object
* Function

Quando você quiser descobrir o tipo de dado, use o ‘typeof’ antes da variável.

Comentários no Javascript: // para uma única linha, /\* \*/ para mais de uma linha de código.

Tratamento de dados

Ao armazenar uma informação em uma variável, e ela ser retornada ao cliente, você não pode colocar a variável dentro das ‘ ‘ e deve colocar o sinal + antes para ligar a variável a frase, se houver. Isso se chama concatenação (relacionamento de ideias, fatos ou coisas entre si; )



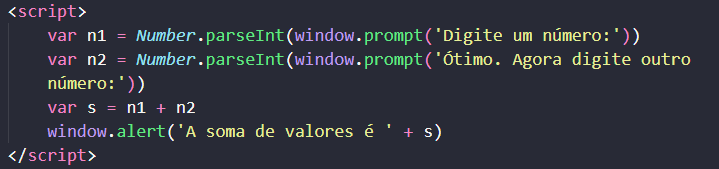
Falando um pouco mais sobre o +, ele tem 2 funções: CONCATENAÇÃO e ADIÇÃO. No Javascript você precisa forçar o sinal a identificar para qual meio ele será usado. Você pode fazer isso usando da seguinte maneira:

(number + number) para adição.

(string + string) para concatenação.

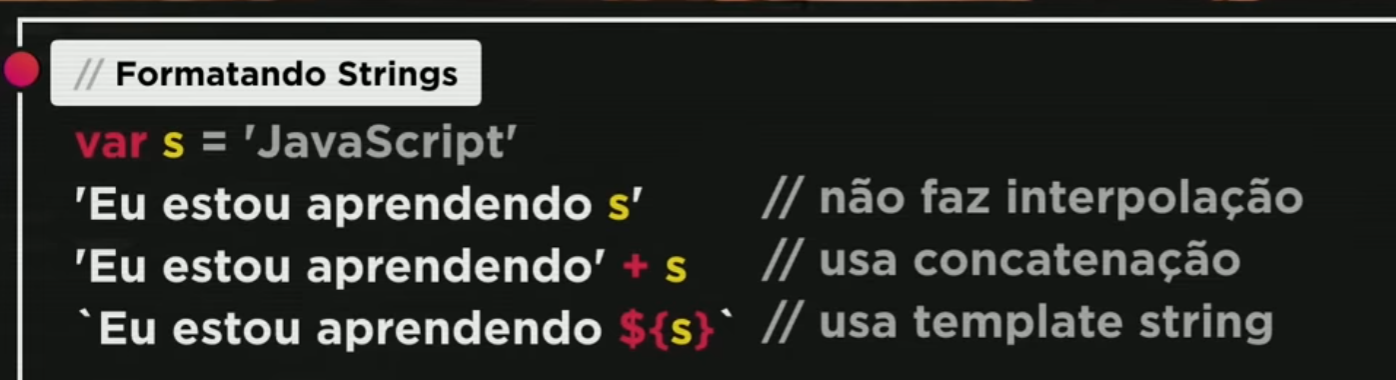
Para o Javascript identificar a string como number é necessário que ocorra a conversão de dados.

Para converter um número para um número inteiro, use – Number.parseInt(n) – caso seja para número real, use – Number.parseFloat(n) –



Também é possível usar apenas - Number(n) –

Agora, se você quiser converter um número em string, você deve usar os elementos – String(n) – ou – n.toString() –



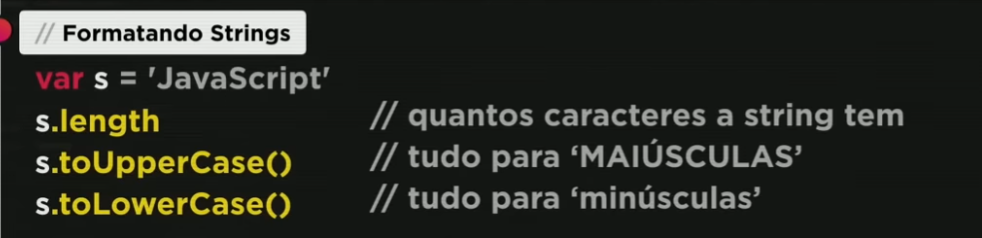
Ao usar o template string, atente-se a usar crase ao invés de aspas.

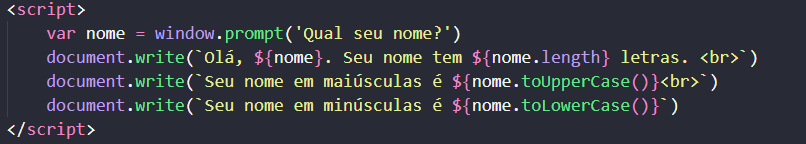
Comandos para usar no Javascript:

.length : Diz quantos caracteres tem a palavra.

Ex: document.write(`Seu nome tem $(nome.length) letras.`)

Alguns comandos para formatar as string:





Alguns comandos para formatar numbers:

.toFixed(2) = Adiciona 2 casas decimais ao número.

Curiosidades:

n1.toLocaleString('pt-BR', {style: 'currency', currency: 'BRL'})

Deixa o número no valor do real brasileiro. Ele faz isso localizando a informação na internet e serve também para outras moedas do mundo.

.replace(‘ ‘ , ‘ ‘) : Troca os valores digitados entre as aspas.

Operadores

* Aritméticos
* Atribuição
* Relacionais
* Lógicos
* Ternário

**Operadores aritmético**

Alguns operadores são : +(Soma) , -(Subtração) , \*(Multiplicação) , /(Divisão(por padrão é a numeração real 5/2= 2.5) , %(Resto da divisão) , \*\*(potência) Todos esses são considerados operadores binário, pois necessitam de 2 números para funcionar.

No Javascript a ordem de precedência para operadores aritméticos é: ()Parentes > \*\*Potência > /Divisão - \* Multiplicação - %Resto da divisão (esses 3 tem o mesmo valor de precedência, porém será priorizado da esquerda para direita) > + Soma -Subtração.

Auto-atribuições:

Var n = 3

n = n + 4 virou 7, agora de forma simplificada: n+= 4 essa simplificação só serve para auto-atribuições

n = n – 5 virou 2, agora de forma simplificada: n-= 5

n = n \* 4 virou 8, agora de forma simplificada: n\*= 4

n = n / 2 virou 4, agora de forma simplificada: n/= 2

n = n \*\* 2 virou 16, agora de forma simplificada: n \*\*= 2

n = n % 5 virou 1, agora de forma simplificada: n%= 5

**Operadores de incremento**

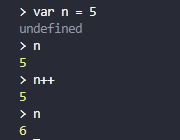
Vamos iniciar com um exemplo:

Var x =

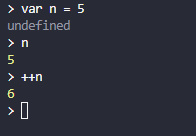
x = x + 1 x + = 1 x++ resultara em 6

x = x – 1 x - = 1 x-- resultara em 4

//Ele dará o resultado na próxima vez que for acionado. (pós-incremento)



//Ele dará o resultado logo após acionar o comando. (pré-incremento)

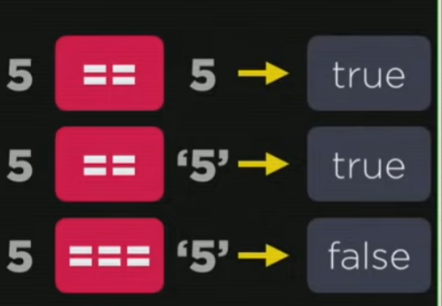


**Operadores relacionais**

Alguns operadores relacionais: < (menor que), > (maior que), >= (menor ou igual), <= (maior ou igual), == (igual), !=( não igual/diferente)

Para toda expressão que tenha um operador relacional ligado a ela, o resultado sempre será um valor booleano (true or false).

Temos também operadores relacionais de identidade:

O == analisa o valor em si, ignorando o tipo de dado (string ou number).

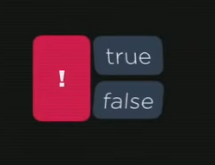
O === analisa apenas o valor exato do dado, portanto, ele apenas reconhecera o mesmo tipo de dado.

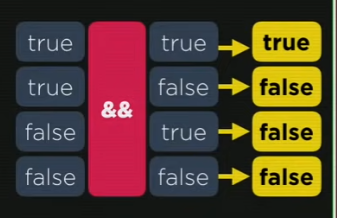
A mesma coisa funciona para o desigual restrito !=, !==.

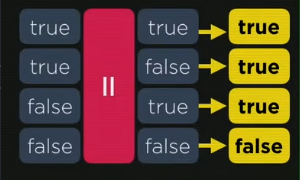
**Operadores lógicos**

Alguns operadores lógicos: ! (negação), && (conjunção), || (disjunção)

Exemplo: ! Não quero nenhuma caneta. && Quero uma caneta azul e uma laranja. || Quero uma caneta azul ou laranja.

O ! é um operador unário, ou seja, só tem um operando, ou ele é true ou ¨false¨

O && é um operador binário, ele tem dois valores lógicos. Lembre-se do exemplo da caneta. Nesse operador só serve se receber as 2 exigidas.



O || também é um operador binário. No entanto, ele se conforma com apenas um dos resultados pedidos por ele. Lembre-se das canetas.

No Javascript, a ordem para operadores no geral é:

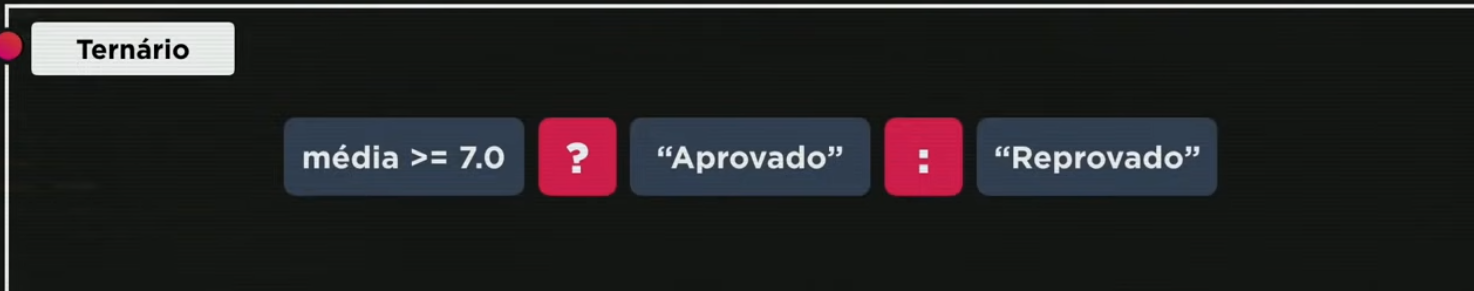
1. Aritméticos
2. Relacionais
3. Lógicos (A ordem de precedência deles é !, &&, ||)
4. Ternário

**Operador ternário**

Os operadores ternário: ¨teste ? True : false¨

Ele se chama ternário por que precisa de 3 elementos.

Ex:



**Entendendo o DOM**

[*https://developer.mozilla.org/pt-BR/docs/Web/API/Document\_Object\_Model/Introduction#dom\_e\_javascript*](https://developer.mozilla.org/pt-BR/docs/Web/API/Document_Object_Model/Introduction#dom_e_javascript)

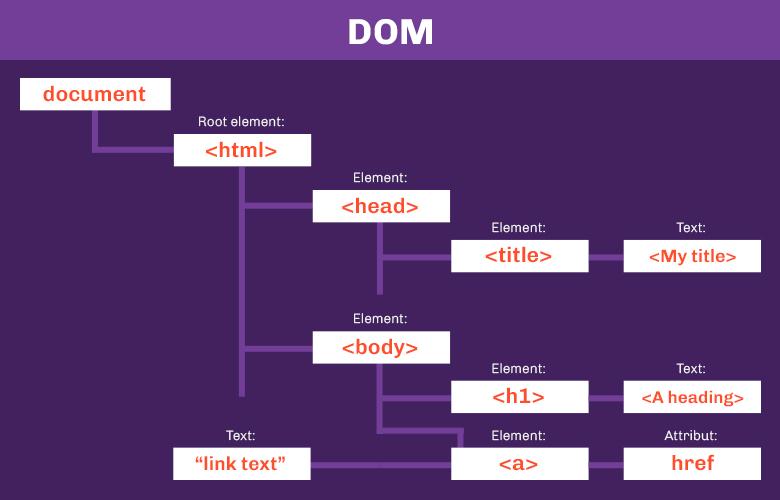
*Acesse para um estudo mais aprofundado!!*

Document Object Model = Modelo de Objeto de Documento]

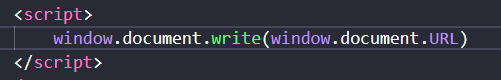
É uma interface que representa documentos HTML e XML como onjetos, permitindo que linguagens de programação como JavaScript acessem e manipulem esses documentos.

DOM é um conjunto de objetos dentro do seu navegador que vai dar acesso aos componentes internos do seu website. Ele funciona quando você está rodando o JavaScript dentro do navegador.

**Árvore DOM**



Os elementos “Pai” são chamados de “Parent” e os elementos “Filho” são chamados de “Child”. Cada site tem sua árvore hierárquica.

Você pode observar que o código escrito é basciamento o caminho da árvore DOM ‘window > document > ...’

1. **Windows**
   1. **Location -**
   2. **Document -**
   3. **History -**
   4. **Navigator -**
   5. **Screen -**
   6. **innerWidth -**
   7. **innerHeight -**

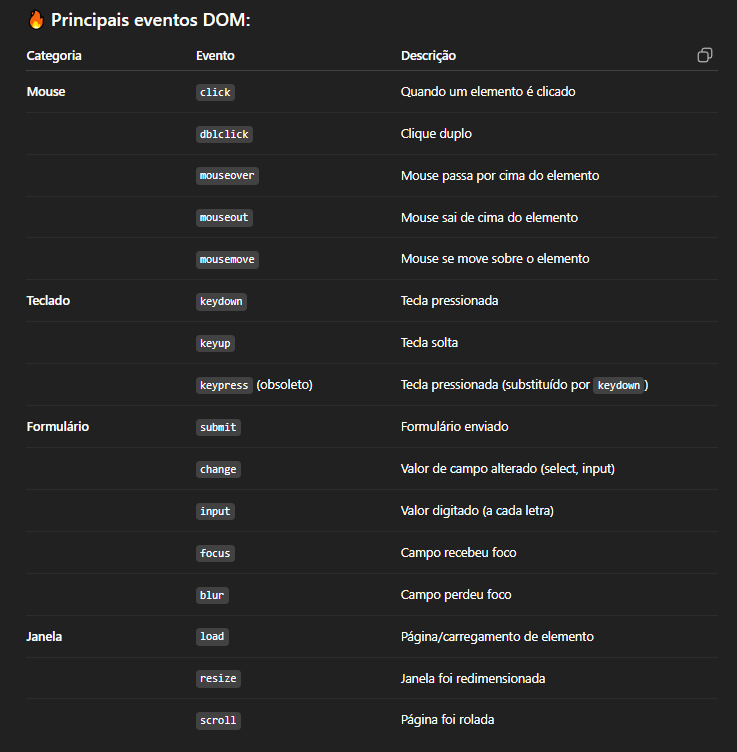
**Selecionando os elementos**

Você pode acessar por:

* por Marca – getElementsByTagName()
  + Para selecionar parágrafos em ordem você deve usar [ ] depois do parênteses e digitar o número referente a posição hierárquica do elemento. [0] Primeiro parágrafo da tag x, [1] segundo parágrafo da tag x...
* por ID – getElementById()[ ]
* por Nome – getElementsByName() [ ]
* por Classe – getElementsByClassName() [ ]
* por Seletor - querySelector() e querySelectorAll()
  + Nesse você define se vai ser ‘id’ ou ‘class’ usando o ‘.’ ou ‘#’ igual no HTML e CSS

**Eventos DOM**

Eventos DOM é tudo que pode acontecer com os elementos HTML. Tudo que você pode fazer, até mesmo com o simples mover do mouse sobre uma div, ex: mouseenter, mousemove, mousedown, mouseup, click, mouseout; todos são disparados com o mouse ao interagir com um elemento.

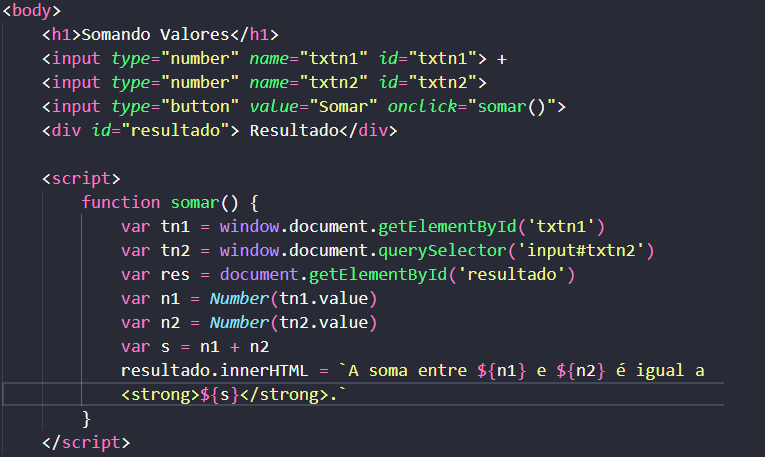


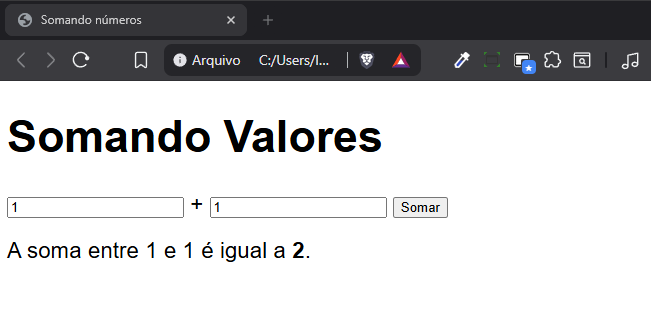
**Funções**

[*https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Functions#definindo\_fun%C3%A7%C3%B5es*](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Functions#definindo_fun%C3%A7%C3%B5es)

*https://www.youtube.com/watch?v=g4zT5GHljbM*

Funções são blocos { } de construção fundamentais em JavaScript. Uma função é um procedimento de JavaScript - um conjunto de instruções que executa uma tarefa ou calcula um valor. Para usar uma função, você deve defini-la em algum lugar no escopo do qual você quiser chamá-la.

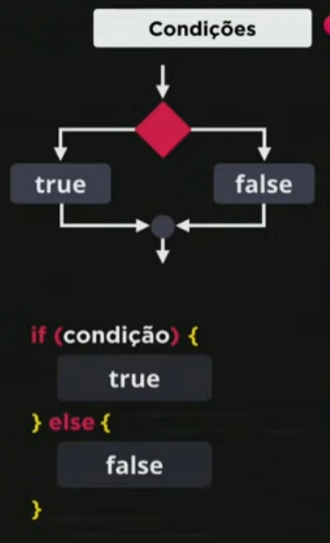




**Condições**

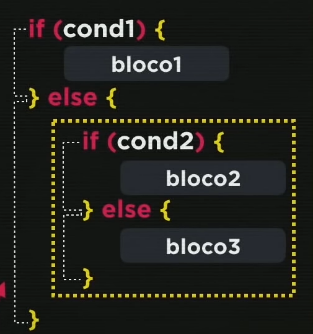
As **estruturas condicionais** são usadas para determinar quais blocos de código serão executados, dependendo de uma condição ser verdadeira ou falsa. Basicamente, elas permitem que seu código "pense", tomando decisões com base nas condições que você definir.

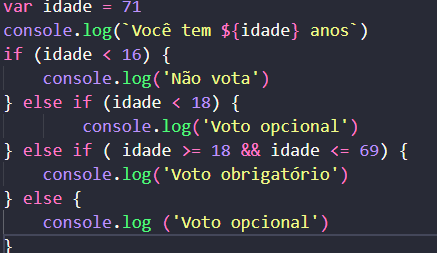
Em **JavaScript**, estruturas condicionais são usadas para executar diferentes blocos de código com base em condições específicas. As principais estruturas condicionais são if, else if e else, além do operador ternário ? : . Elas permitem que o código tome decisões e execute diferentes ações dependendo do resultado de uma avaliação booleana.



**Tipos de Condição:**

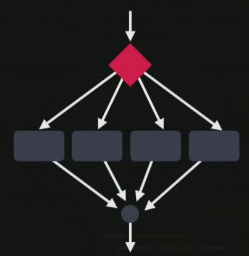
Condição simples contém somente o: if (condição) {true}

Condição compostas contém: if (condição) {true} else {false}



Condição aninhada:

uma condição dentro da outra;

Condição múltipla: Serve para valores fixos e situações especificas. Ela tem possibilidades de valores além do if e else.

Condições Múltiplas:

Switch (expressão) {

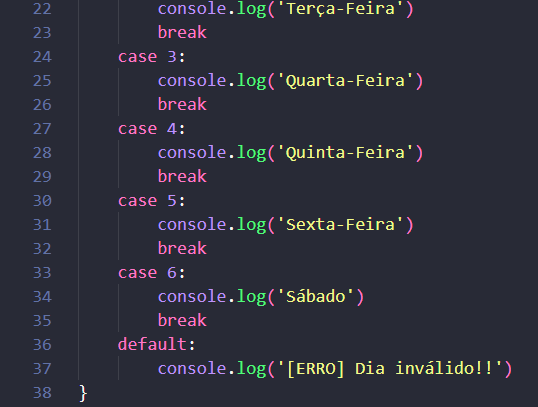
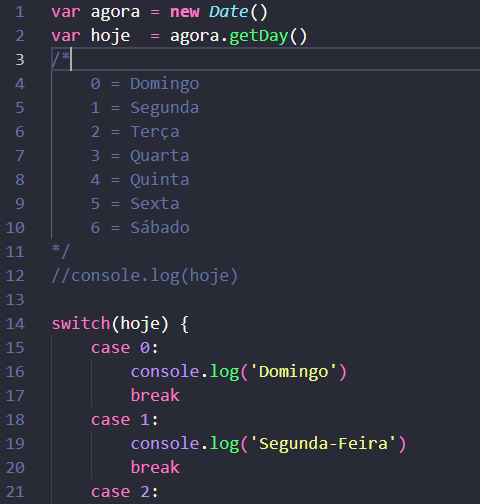
Case valor1: É obrigatório colocar o em cada bloco ‘break’

Case valor 2:

Case valor 3:

Default:

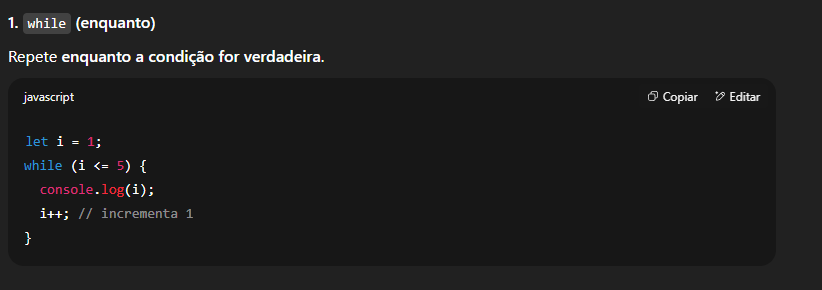
}

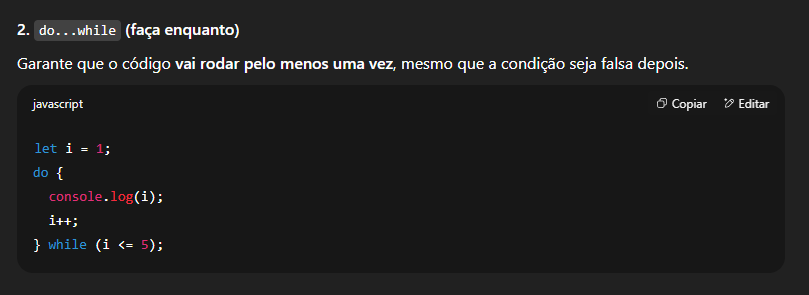
 Ele só funciona com números inteiros e ‘strings’. Ele é muito útil em situações pontuais.

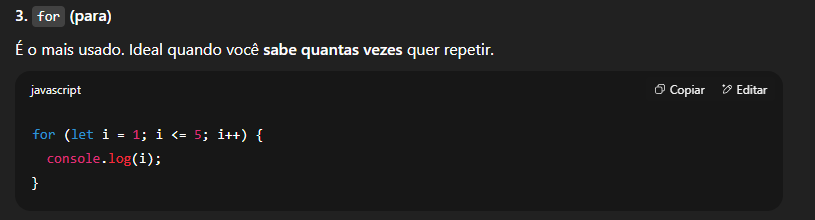
**Estruturas de repetição**

As **estruturas de repetição** (ou **loops**) servem para **executar um bloco de código várias vezes** — até que uma condição seja atingida.

São muito úteis quando você precisa **repetir algo sem escrever o mesmo código várias vezes**. Por exemplo: mostrar os números de 1 a 10, percorrer itens de uma lista, contar de trás pra frente, etc.







**Use o console de depuração para testar o código durante a escrita dele.**

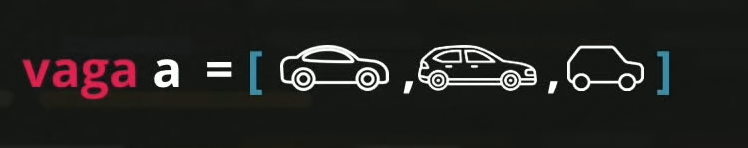
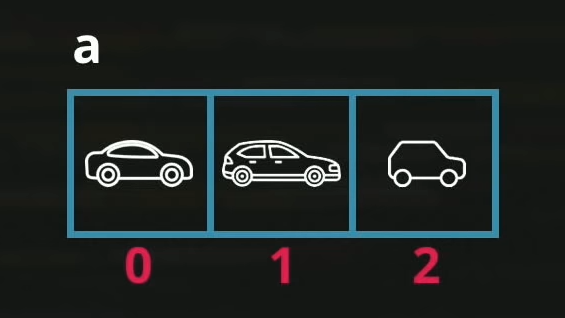
**Variáveis Compostas (Arrays)**

Variáveis simples só conseguem armazenar **um valor** por vez.

Variáveis compostas são capazes de armazenar **vários valores** em uma mesma estrutura.

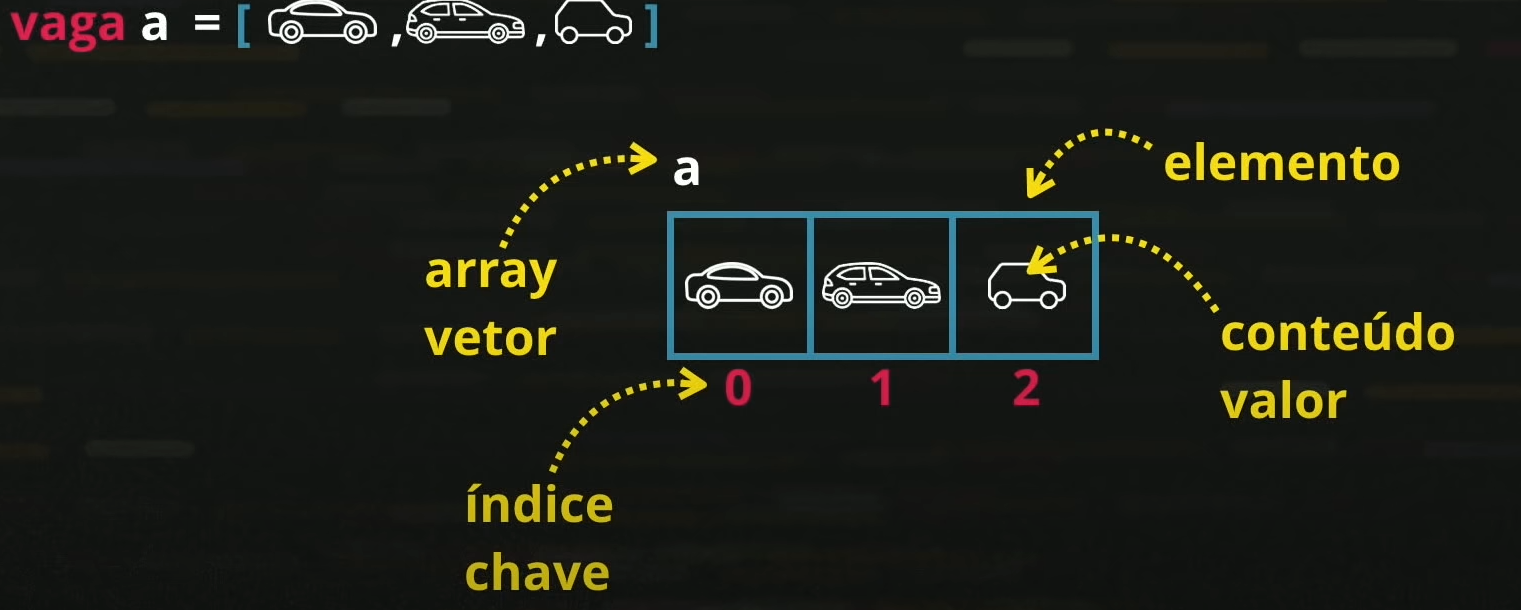
Exemplo : var A = Identificador de índice [0, 1 , 2, 3, 4]

Array(vetor):

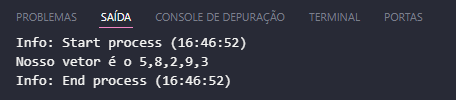
As variáveis compostas iniciam o índice com 0. EX= 0, 1, 2 ,3 ,4 5, 6, 7, 8, 9. (Várias linguagens de programação funcionam assim.)

Array(vetor) é uma variável com vários espaços. O Vetor é composto de elementos. O elemento ocupa um espaço na memória, o valor colocado dentro dele e o índice(chave, key).



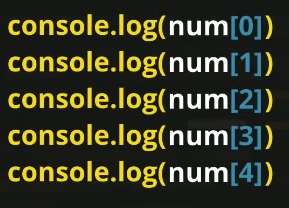
O vetor (ou uma variável composta) é uma variável que tem vários elementos. Cada elemento é composto por seu valor e uma chave de identificação.





Vamos ver um exemplo:

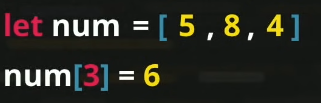
Se eu quiser que o vetor me mostre apenas um elemento eu devo digitar a chave do elemento entre colchetes:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

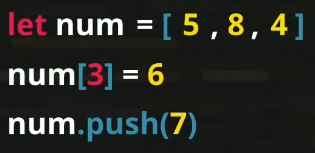


Se você quiser acrescentar um elemento nesse vetor, de forma automatizada, basta seguir a resolução a seguir:

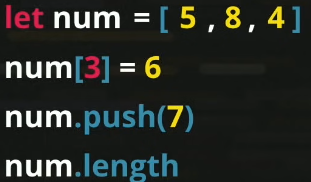
 O próprio JavaScript irá criar o elemento.

Vale lembrar que, o “3” do “num[3]” é referente a chave 3 do elemento no vetor(Lembre que no vetor a contagem do índice se inicia do zero).

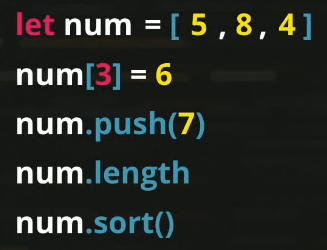
Se você quiser acrescentar um elemento e deixar explícito que ele pertence a última posição, use:

 O “.push” determina que ele devera ficar em última posição.

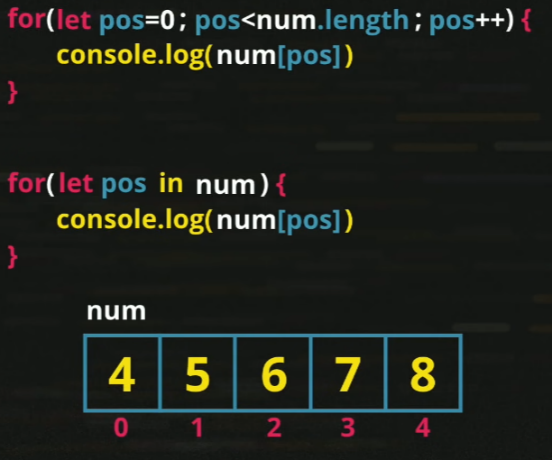
Para descobrir quantos elementos tem no vetor, você deve usar o atributo “.length”:

 Especificamente no Javascript, o “.lenght” não tem (), diferente de outras linguagens que usam.

Para coloca todos os elemento sem ordem crescente, use o método “.sort()”.



Tem uma estrutura de repetição que é usada apenas em arrays e objetos(arrays são objetos). Ela serve para simplificar seu código:

O de cima é a forma normal.

O de baixo é da forma simplificada.

**Dicas do Javascript:**



Também serve para várias outras funcionalidades.

**.value**

No JavaScript, o **.value** é usado para **acessar ou definir o valor de um elemento de formulário** (como **<input>**, **<textarea>**, **<select>**).

.value **só funciona com elementos que aceitam valor**, como:

* <input>
* <textarea>
* <select>

Para pegar texto de outros elementos como <p>, <div>, etc., você usa:

* .innerText ou .textContent