**Programando com JavaScript**

Um IDE (ou Ambiente de Desenvolvimento Integrado) é um software que permite programar utilizando recursos extras.

Com IDEs, você pode editar, modificar, debugar, executar e implantar seu código. (Sem um IDE, um desenvolvedor deve selecionar, implementar, integrar e gerenciar todas essas ferramentas separadamente.)

IDE: (mais utilizados)

* Eclipse
* Visual Studio
* **Visual Studio Code (que vamos usar)**
* Netbeans
* Atom
* WebStorm

\*com o console do navegador tem como testar alguns dos códigos.

JavaScript pode ser usado de varias formas (tanto no lado do cliente como servidor).

* Para o lado do servidor iremos usar o NodeJS.
* No lado do cliente iremos usar com HTML.

Os documentos com JavaScript são feitos com (.js) ex: script1.js

Sempre que quiser inserir/informar algo (número, nome...):

Let nome =

Se eu colocar Console.log ele imprime as informações:

Console.log (“seu nome é ” + nome + “ e sua idade é “ + idade);



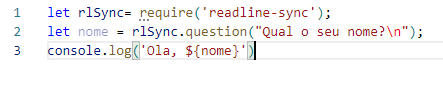
**Instalando um modulo que permite que faça input pelo CMD:**

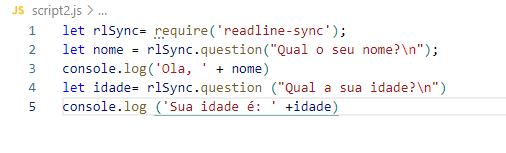
Vou em terminal e escrevo *“npm install readline-sync –save”*

Se eu quiser usar ele coloca require (‘readline-sync’);

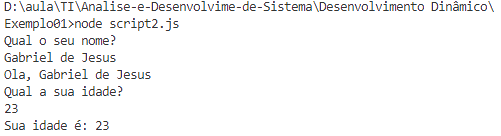
Isso permite que eu faça pergunta para quem vai utilizar! (obs: **\n** para soltar espaço no js.)

* Se eu não quiser usar +nome... posso usar ${nome})





Para executar eu vou no terminal e coloco node nome do arquivo.js

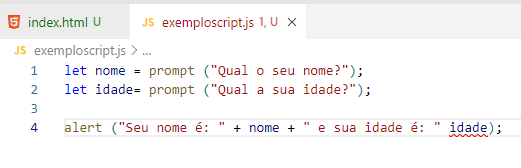


**Posso usar com Html**

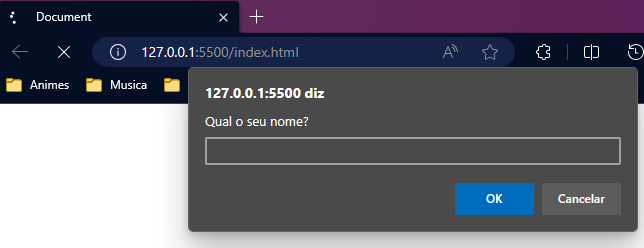
Faz um index.html padrão. Na cabeça eu crio um script src e coloco **“script.js”** e criar

Nele posso fazer meus let nome; let idade e etc

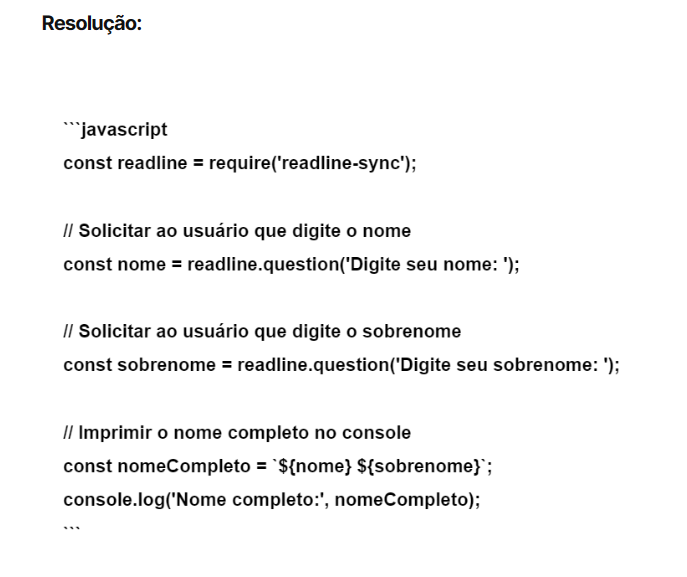
* para eu solicitar para alguém posso colocar o comando prompt
* e se eu quiser mostrar alert



E assim que executar o html ele aparece:



Desafio 01: perguntar nome, sobrenome e falar



**Conceitos gerais do JS**

Linguagem de script orientada a objetos, multiplataforma.

Dentro de um host o JS pode ser ligado aos objetos deste ambiente para prover um controle programático sobre eles.

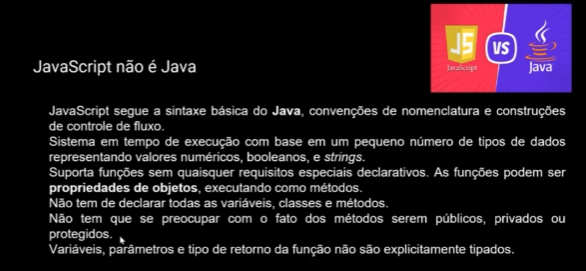
**Biblioteca padrão de objetos**: Array, Date e Math.

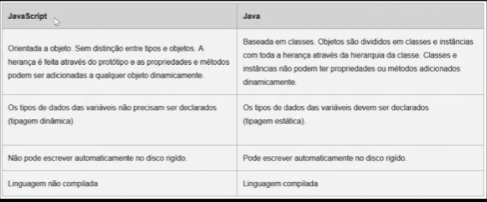
**Conjunto de elementos**: Operadores, estruturas de controle e declarações.

Pode usar do lado do:

* **Cliente:** fornece objetos para controlar um navegador web e seu Document Object Model (DOM), eventos do usuário. (Interação!)
* **Servidor:** fornece objetos relevantes à execução do JS em um servidor. Operacoes com BD e arquivos.

**Javascript não é java!**





JS tem versões! Padronizado pela Ecma Internacional.

**Estrutura da linguagem:**

* Ele é case-sentive: Ele sabe que maiúscula é diferente de minúscula.
* Espaços, tabulação e uma nova linha são considerados espaços em brancos.
* É lido da esquerda para a direita.
* Recomenda-se sempre adicionar ponto e virgula no final das declarações.
* Escolha nomes coerentes para as variáveis> var **Nome**= “Maria”;
* Adicionar comentários no código: \\ ou /\* e no final \*/

**Regras para identificadores de variáveis:**

* Podem conter somente caracteres alfanuméricos (ou “$” ou “\_”).
* Não podem iniciar com um digito. (ou seja, não começa com numero)
* É “case sensitive”, Estado\_origem é diferente de estado\_origem

Ex valido: Nome\_funcionario, \_Salario, $cidade.

**Declarações:**

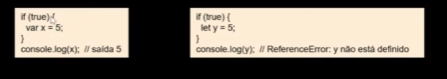
* **var**: declara uma variável e a inicializa com um valor. Ex: var Nome= “Maria”; *(global)*
* **let**: declara uma variável local, escopo do bloco, a inicialização do valor é opcional. Ex: let idade = 22; *(local)*
* **const**: declara uma constante de escopo do bloco, somente leitura.

**Variáveis global: (var)**

* Declaradas fora de uma função.
* Disponíveis para todos os blocos.

**Variáveis locais: (let)**

* Declarada dentro de uma função.
* Disponível apenas para a função onde foi declarada.



**Constantes: (const)**

* Apenas leitura.
* Sintaxe igual da variável.
* Não pode alterar seu valor por meio de uma atribuição ou ser declarada novamente enquanto o script está em execução.
* Deve ser inicializado com um valor.

Ex: const PI= 3.14;

**Diversas formas de trabalhar com JS:**

Node, HTML...

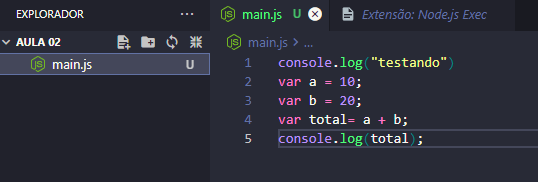
Abrir o navegador e abrir o console do navegador! (para testar).

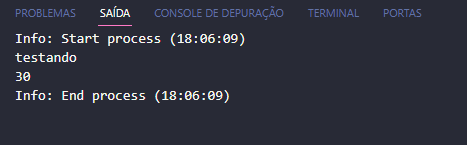
O comando console.log é para da uma informação no console!

Tipo console.log (“Olá”)

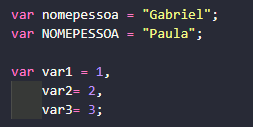
O arquivo javascript tem o “.js”

Para executar com a extensão só escrever o código e apertar “f8”





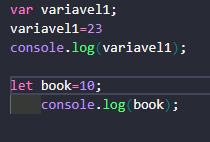
**Posso declarar variáveis separadamente ou junto separando por “,”:**



Ultimamente não se usa muito o var e sim o let!

**O var não precisa iniciar com valor, posso declarar depois!**

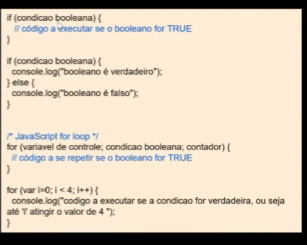
O let precisa iniciar com valor.



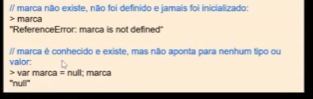
**Tipos de dados em JS:**

Tipos primitivos:

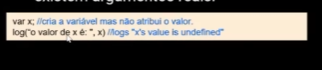
* **Boolean**: é um tipo de dado logico (**verdadeiro ou falso**).



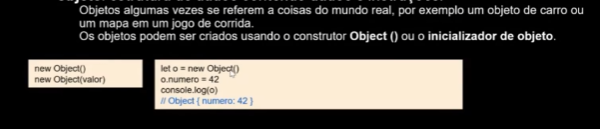
* **Null (nulo):** Vazio.



* **Undefined**: valor atribuído automaticamente a variável que acabou de ser declarada.



* **Number:** é um dado numérico.
* **String:** é uma sequencia de caracteres usados para representar texto. (“entre aspas”)
* **Objeto:** estrutura de dados contendo dados e instruções.

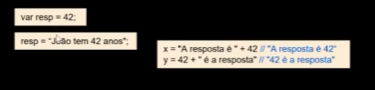


**Conversão de tipos**:

JS é uma linguagem dinamicamente tipada.

Não precisa especificar o tipo de dado de uma variável quando declará-la.

Os tipos de dados são convertidos automaticamente conforme a necessidade durante a execução do script.

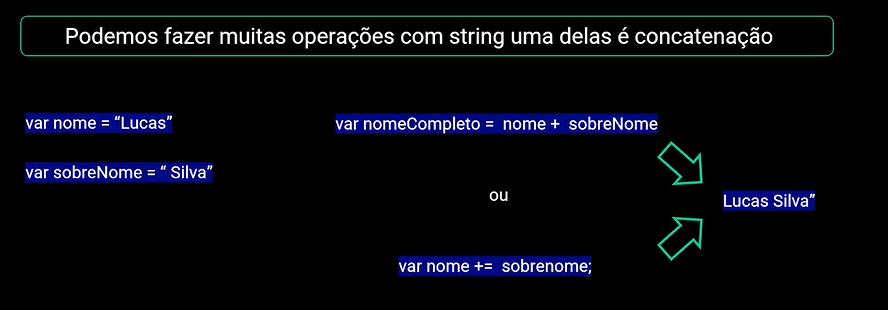


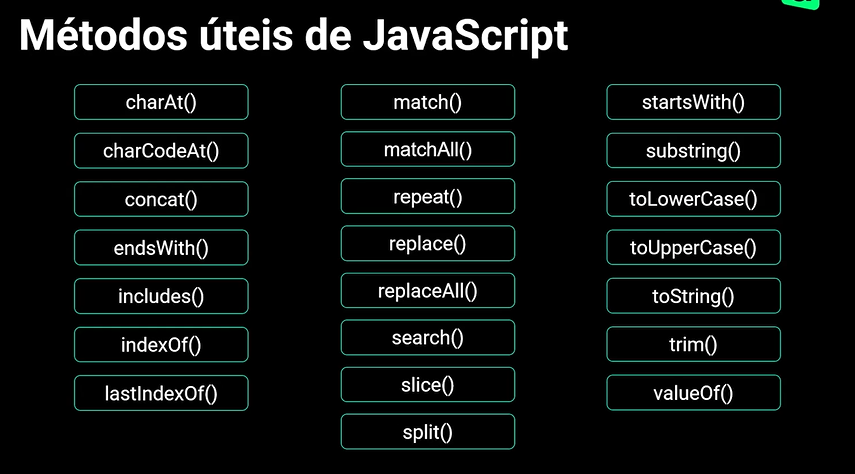
* Para saber o tipo utilizo o typeof
* Colocar o **tudo em caixa alta**: .toLocaleUpperCase()
* *Usar o typeof com o if e else é muito bom!*

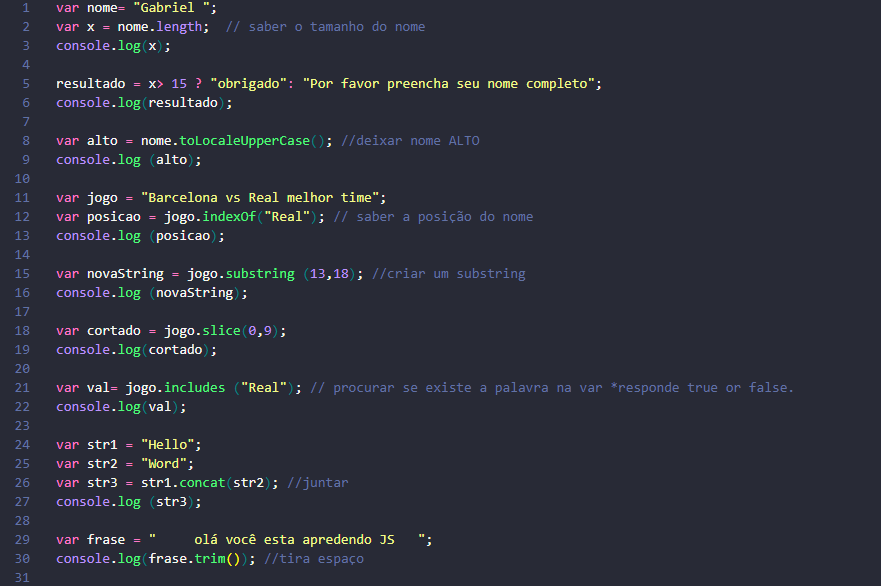
**Strings**

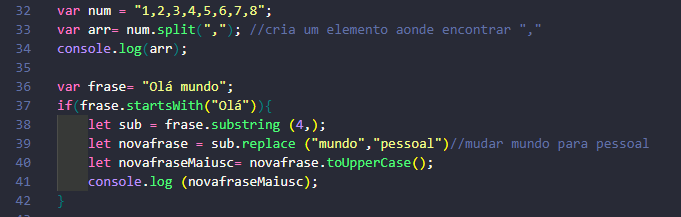
**Usamos string para atribuir valores relacionados a um texto**. Seus valores são delimitados usando “” ou ‘’.

* Var nome = **“**Gabriel**”** ou Var nome= **‘**Gabriel**’**











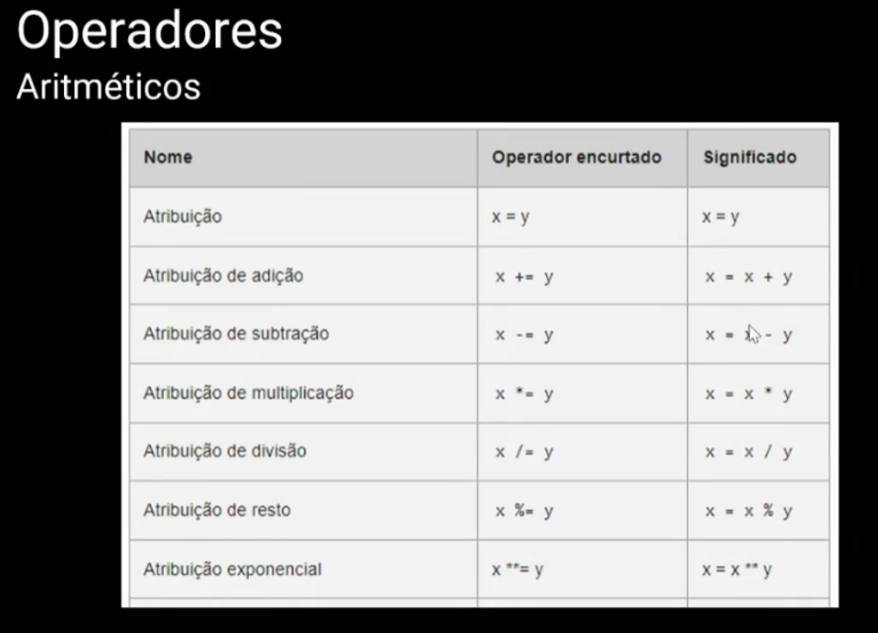
**Operadores**

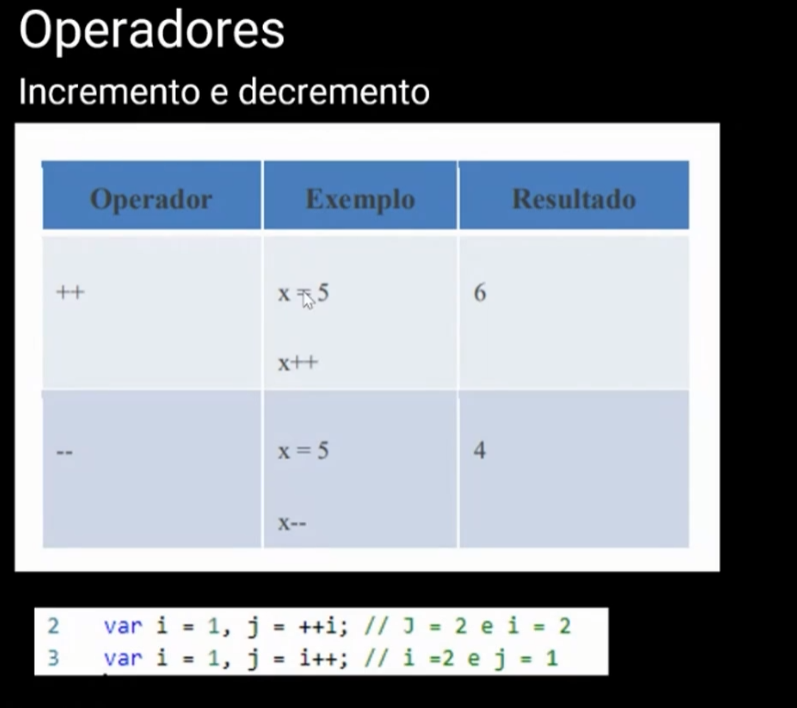
Tipos: **binários e unários**.

* **Binário** (sintaxe): operando 1 operador operando 2
  + Ex: 2+3; 4-5; x\*y
* **Unário** (sintaxe): operador operando // operando operador
  + Ex: ++z, xx--, x++ (atribuindo +1 ou -1)

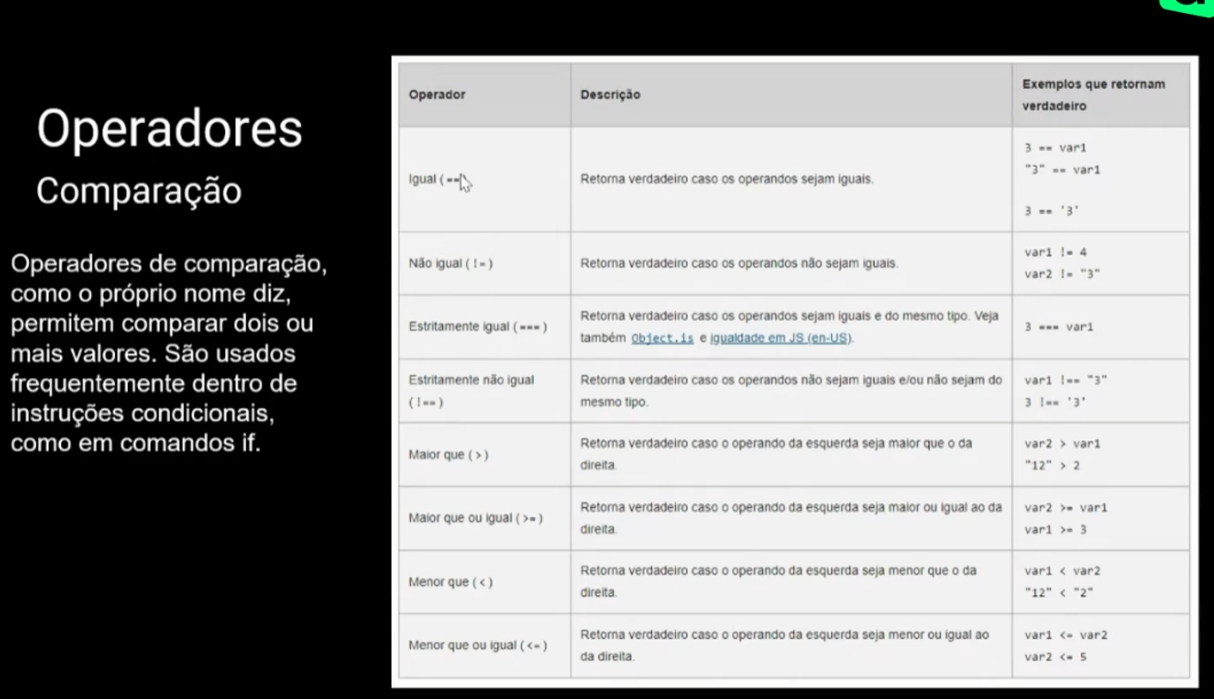
**Atribuição simples.**

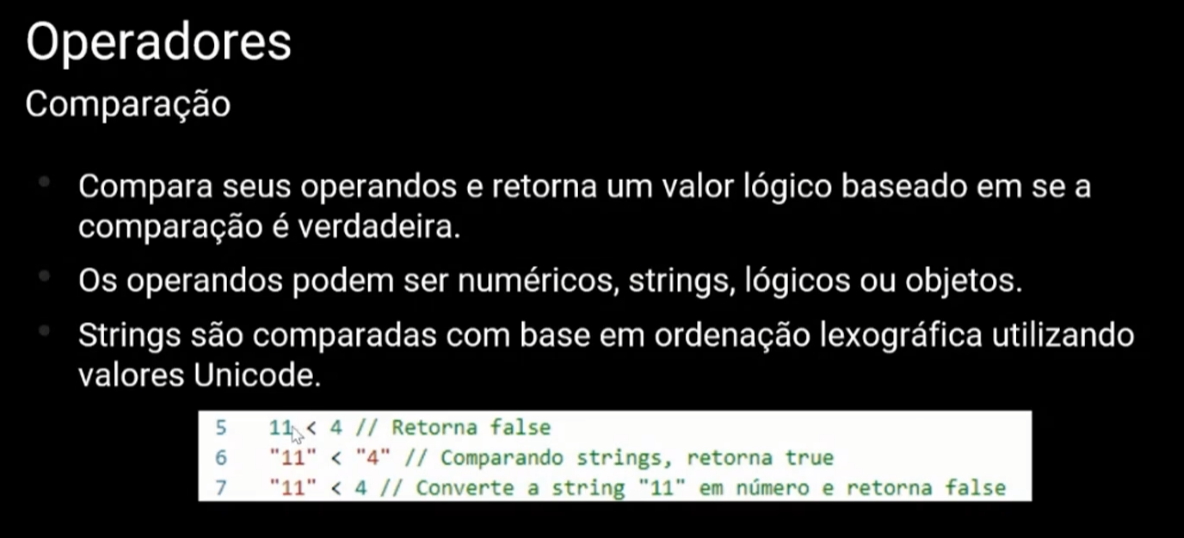
* Operador “=”.
  + Ex: x=y; onde o valor de y é atribuído a x.

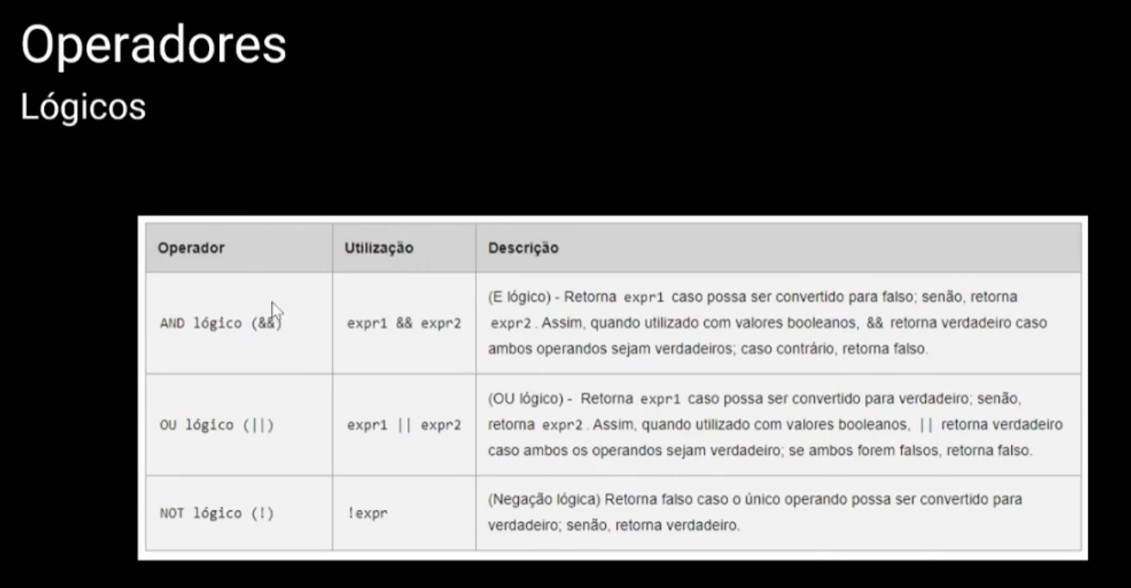




* *++ antes significa que* ***soma antes de atribuir******a J e atribui a I também***
* *++ depois significa que* ***coloca o I no J primeiro e depois soma ao I***



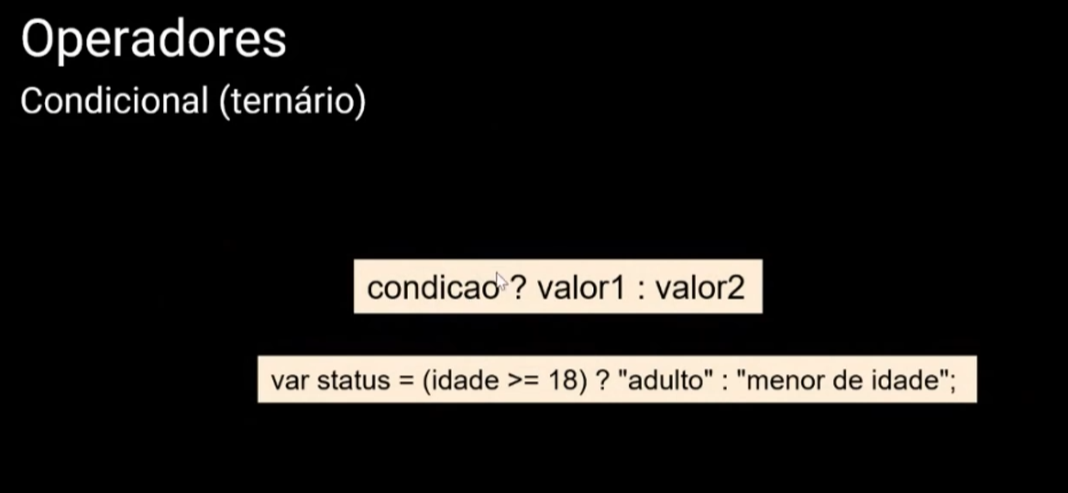




**&& - e**

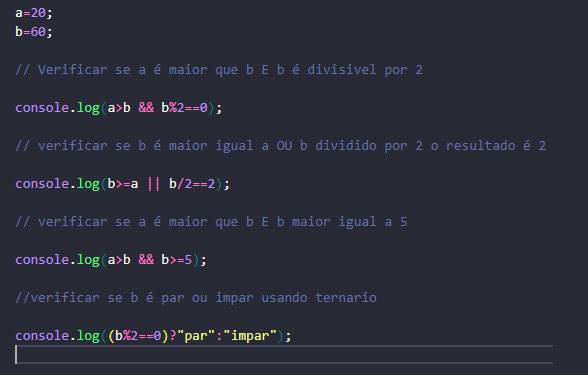
**| | -ou**

**! – NÃO**



**? :**

Traduzindo: Se a idade for maior igual a 18? se sim “Adulto”: se não “menor de idade”;

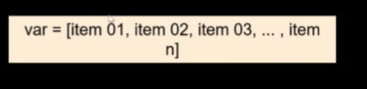


**Arrays**

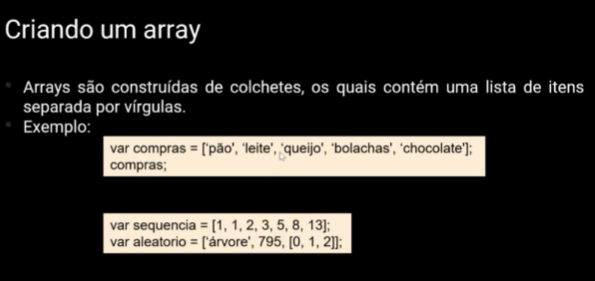
É como se fosse uma lista de objetos!

Um objeto array pode ser armazenado em variáveis e ser tratado de forma muito simular a qualquer outro tipo de valor.

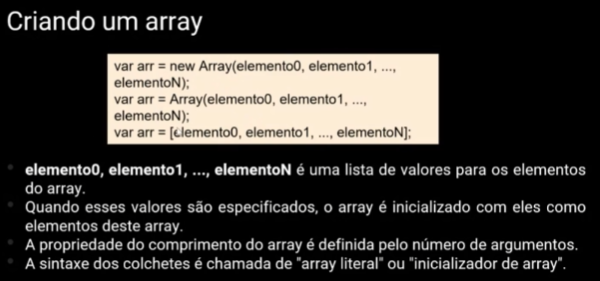
JS não possui um tipo de dados array especifico.



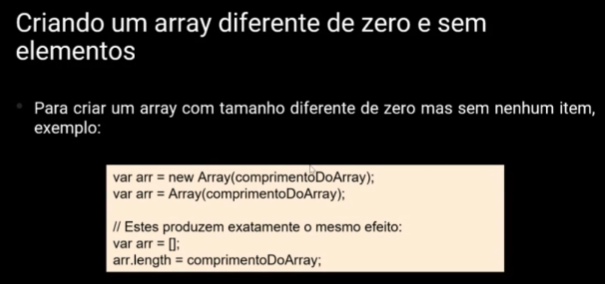
\* colocado entre [separado por “,”] – *cada item pode ser misturado (number, string)*

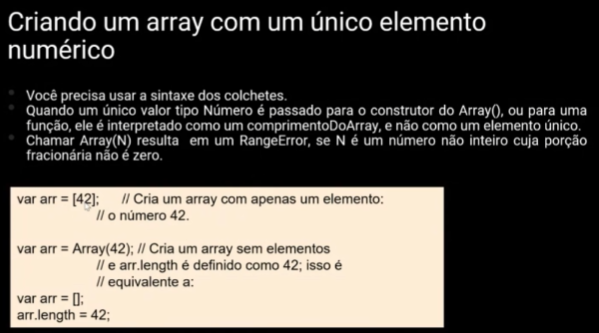


**Pode ser criado dessas 3 formas:**

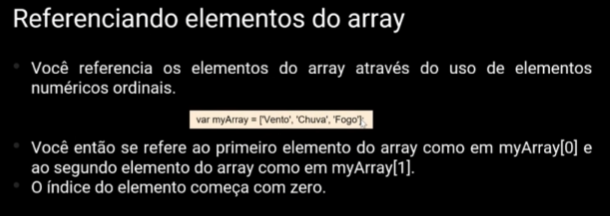


**Particularidades:**

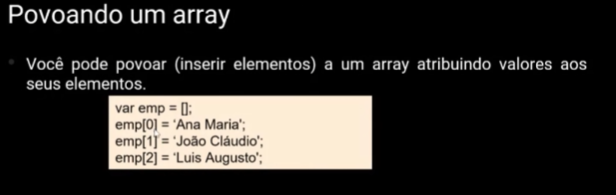


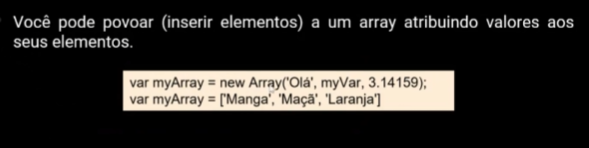


**Referenciando elementos do array**:



**Povoando um array**:





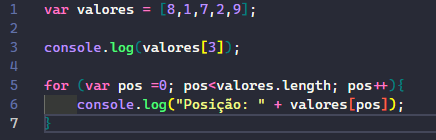
**Interação dos elementos do Array**

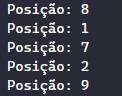
***For****: é forma de interar um array*

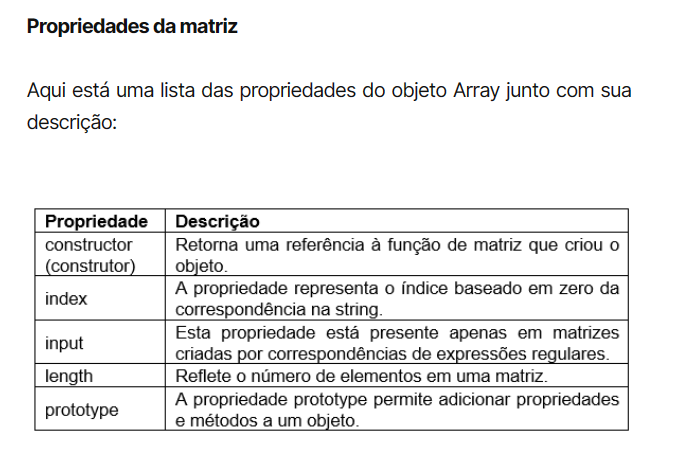
**For:** começa com uma variável de inicialização (por exemplo pos 0); coloca a condição; incremento/decremento.

Toda vez que o pos rodar ele vai ter um incremento (pos++) mas para isso precisa ter um elemento que é verdade!

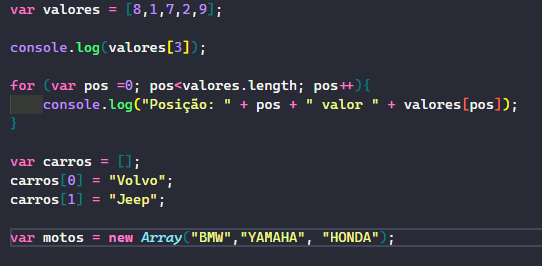
No caso em questão é o pos<valores.length (o pos é menor que o tamanho do array)



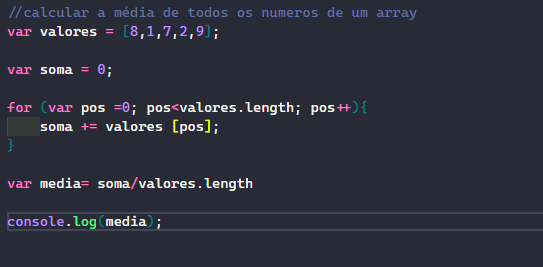




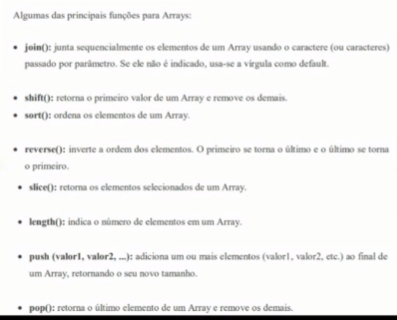
*Formas já vista anteriormente*:



Ex:

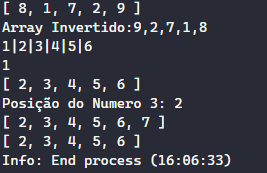


**Métodos de Array**:



* **join()**
* **shift() / unshift() add 1 elemento no inicio**
* **sort()**
* **reverse()**
* **slice()**
* **length()**
* **push(valor1,valor2,...)**
* **pop()**





Pratica:

**Dado um array de números inteiros**, o programa deve realizar duas verificações distintas. **Na primeira verificação**, é necessário identificar se pelo menos um número no array não é par ou é menor que zero.

Para isso, deve-se percorrer o array e aplicar as seguintes condições a cada elemento: **se o número não for par** (ou seja, seu resto da divisão por 2 é diferente de zero) **ou se for menor que zero, então a** **condição não é atendida.**

Caso ao menos uma dessas condições seja verdadeira para algum número do array, a resposta será ‘Sim’. Caso contrário, será ‘Não’.

**Na segunda verificação**, o programa deve determinar s**e todos os números no array são divisíveis por 3 e 5 simultaneamente**. Novamente, é necessário percorrer o array e aplicar as seguintes condições a cada elemento: **se o número não for divisível por 3** (ou seja, seu resto da divisão por 3 não é zero) **ou se não for divisível por 5,** **então a condição não é atendida. Se** todas as condições forem verdadeiras para todos os elementos do array, a resposta será ‘Sim’. Caso contrário, será ‘Não’.

* Resposta:

// Array de números

const numeros = [5, 10, 15, 20, 25];

// Verificar se pelo menos um número do array não é par ou menor que 0

let algumNaoParOuMenorQueZero = false;

for (let i = 0; i < numeros.length; i++) {

if (numeros[i] % 2 !== 0 || numeros[i] < 0) {

algumNaoParOuMenorQueZero = true;

break;

}

}

// Verificar se todos os números do array são divisíveis por 3 e 5

let todosDivisiveisPor3E5 = true;

for (let i = 0; i < numeros.length; i++) {

if (numeros[i] % 3 !== 0 || numeros[i] % 5 !== 0) {

todosDivisiveisPor3E5 = false;

break;

}

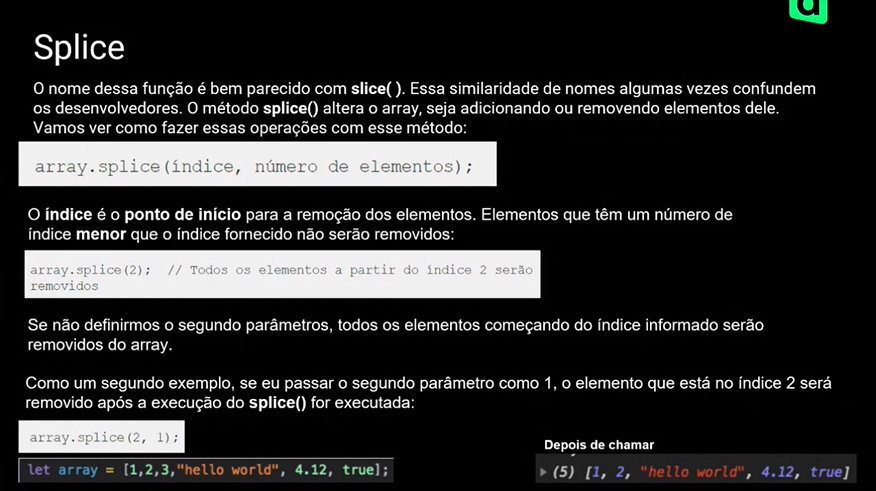
}

console.log(“Pelo menos um número não é par ou menor que zero?”, algumNaoParOuMenorQueZero);

console.log(“Todos os números são divisíveis por 3 e 5?”, todosDivisiveisPor3E5);

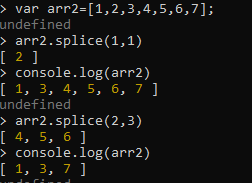
**Avançando com Array**

**Splice** – altera o array *(adiciona ou remove elemento)*



Quando coloca array.splice(2,1); ele remove o 2 elemento.

Primeira coisa que coloca é a posição que quer tirar e depois quanto quer tirar!

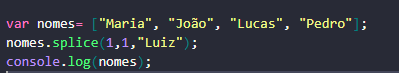


\*no primeiro eu pedi para ele tirar o 1º elemento e só ele

\*no segundo eu pedi para tirar o 2º elemento e 3 depois

**Se eu quiser substituir o elemento:**

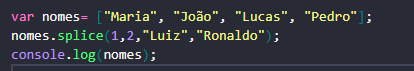
Primeira coisa que coloca é a posição que quer tirar e depois quanto quer tirar e depois o nome que vai substituir.

****

\* Eu pedi para ele trocar o João pelo Luiz



Posso fazer isso com quantos quiser, ex: colocar nomes.splice (1-posição, 2-elementos, “Luiz”, “Ronaldo”) o que ele vai fazer? Substituir o joão por luiz e lucas por Ronaldo.



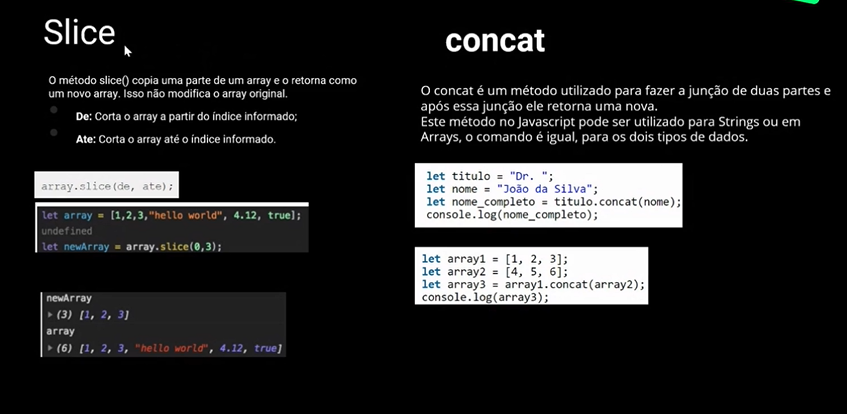


Se eu atribuir uma variável para saber quem saiu é só colocar

Var quemsaiu=nomes.splice (....)

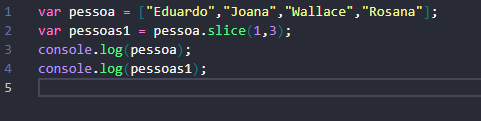
**Mais coisas a ver com array:**

**Manipulando mais array com filter, map, reduce e ForEach**



\***slice ele cria um novo array**. Com parâmetros **de.. até..**(o anterior)  
ex: slice (0,3) ele pega o elemento na posição 0 e os 2 elementos posterior: 1, 2

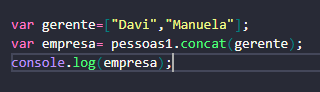
* **Para juntar array é só usar o concat.**



\*eu pedi para ele fazer um novo array da utilizando o nome da joana até rosana. (no caso vai pegar o anterior)

\*note que ele não vai excluir o 1 array, apenas criar outro.

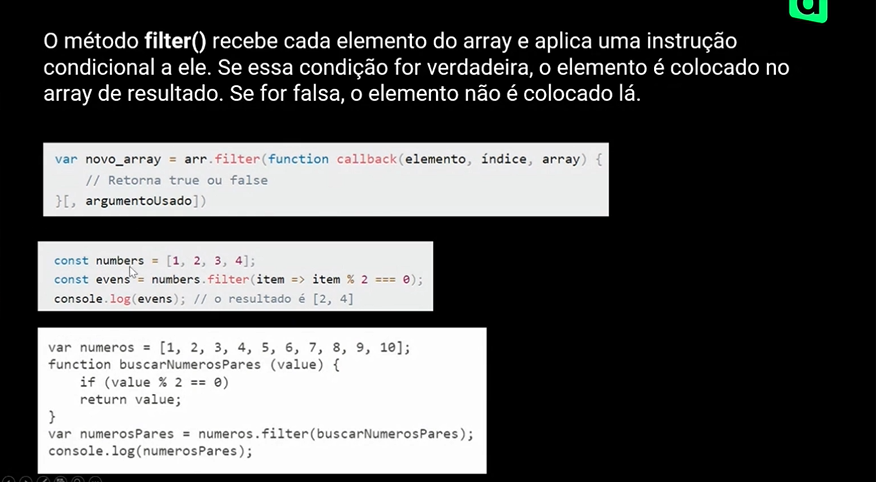




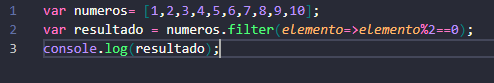
\*aqui eu pedi para ele juntar as pessoas1 com os gerentes para criar a empresa



**Filter:**

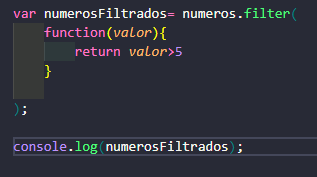


Ex: Filtar elementos no array que são divididos por 2 (ele vai pegar esses elementos e salvar em resultado)





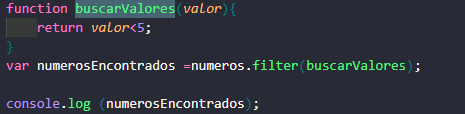
**Posso fazer por meio de uma função:**



\*Aqui pedi para ele filtrar todos os valores que são maiores que 5

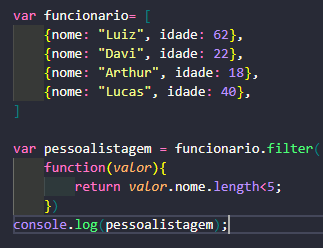


No caso posso fazer a função antes e pedir o filtro por ela. Assim:

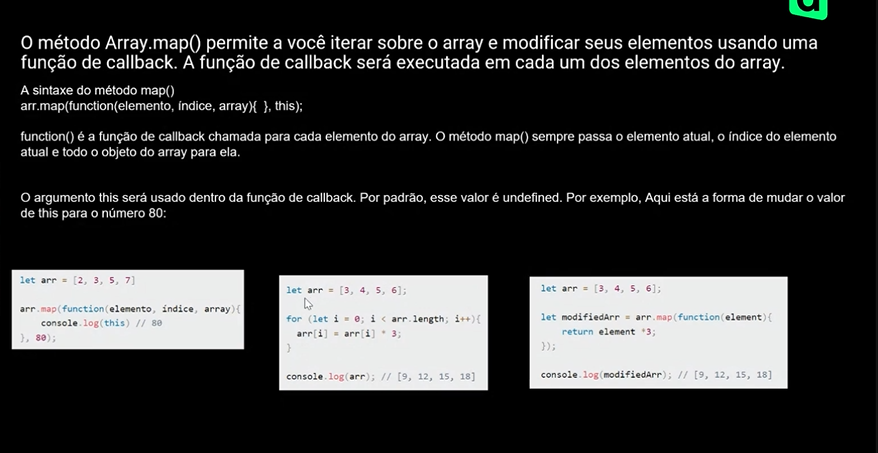




Da para fazer por objeto, if e etc.

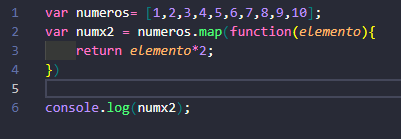


**Map:**

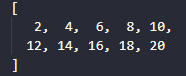
****

Serve para manipular o elemento do array! Pode ser feito por for ou pelo map.

Ex:

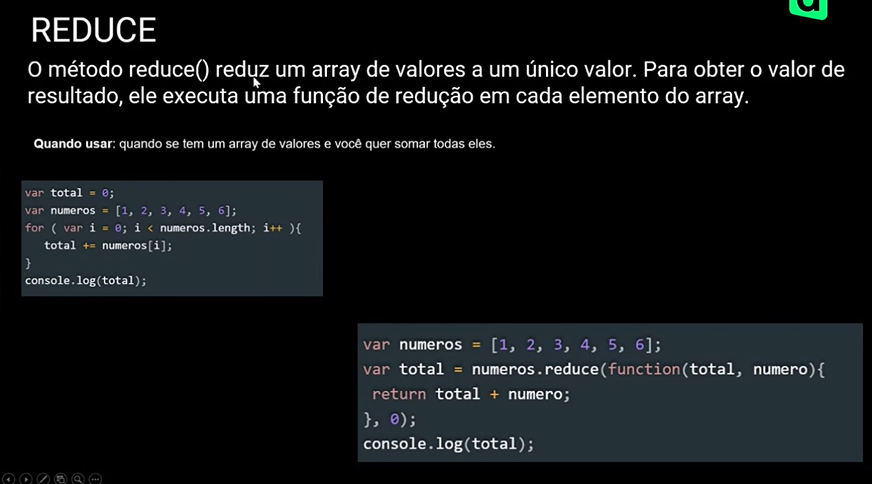


\*aqui eu quero que ele pegue todos os elementos e multiplique por 2



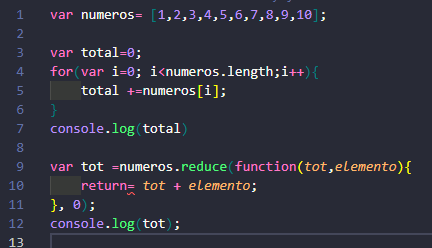
Tem como utilizar com objeto.

**Reduce**



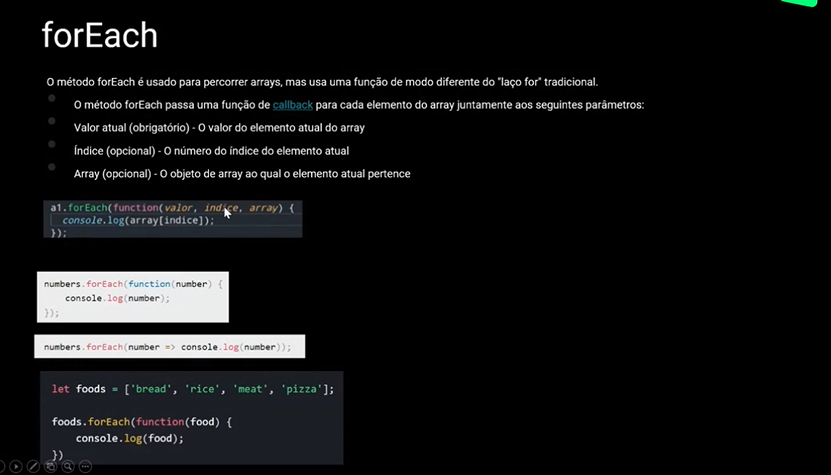
Reduz a um único valor.

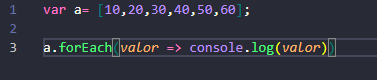
Da para fazer por for ou reduce.



\*aqui ele vai somar os elementos

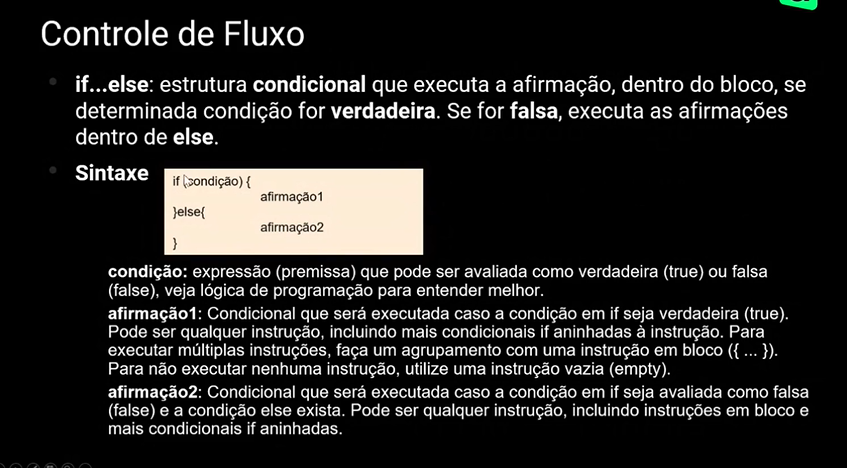
**ForEach**



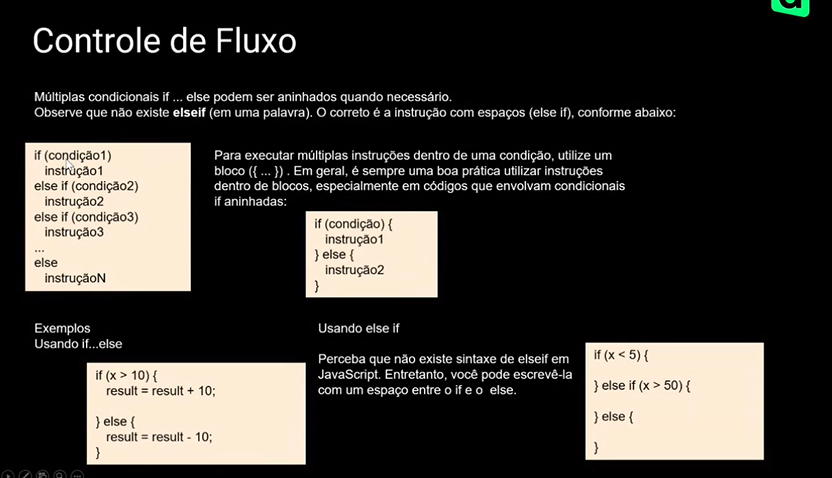


\*ele retorna todos os valores do array

**If**

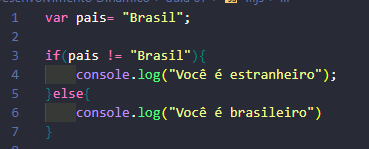


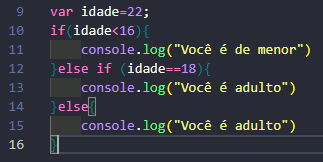
* If é o SE
* Se ela for verdadeira.... SE NÃO (else)



* **Else if** (se a primeira é falsa e testa a segunda...)

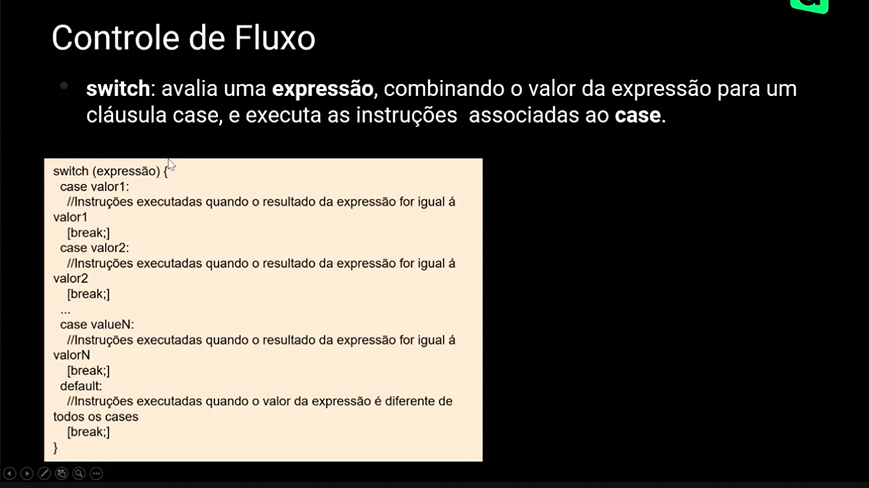
Ex:





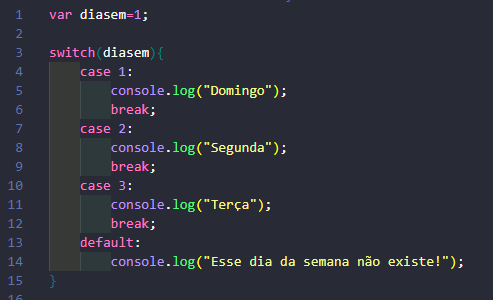
Pode colocar ou também | | em uma condição

**Switch**

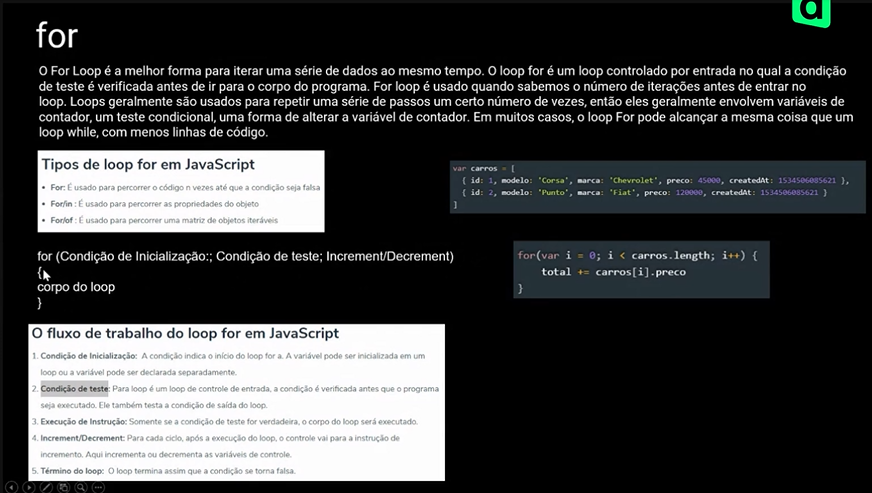


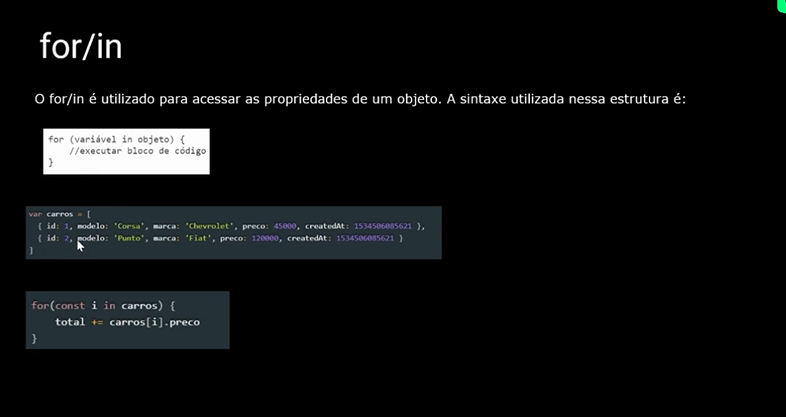


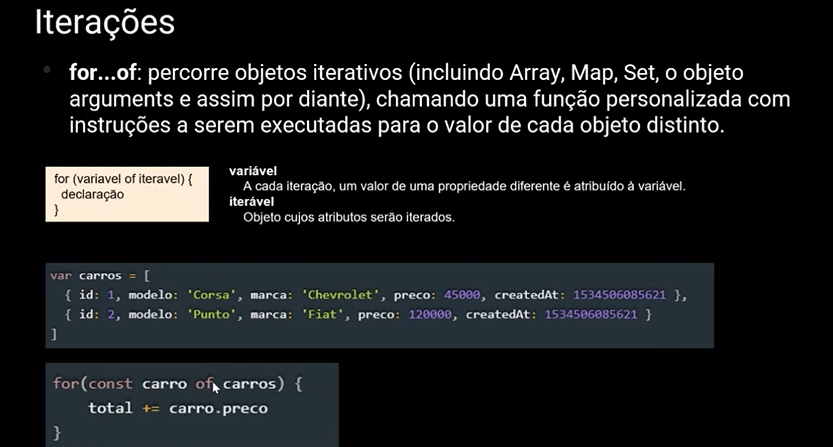
Se eu não tiver break depois do switch ele vai continuar para verificar.



**For**

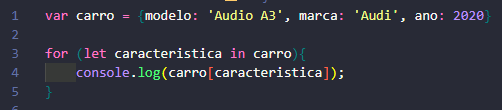






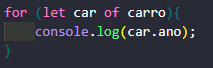
**O "forin" percorre os índices de um array, enquanto o "forof" percorre os valores.**

Ex- usando **for in**: Pedi para ele imprimir toda a característica do carro



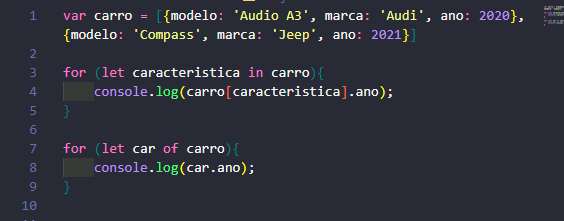


Ex- usando **for of:** Pegando uma propriedade especifica (o ano):

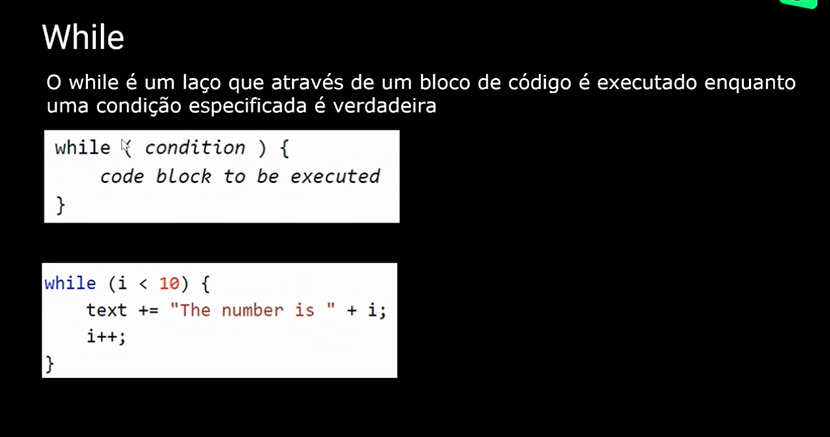


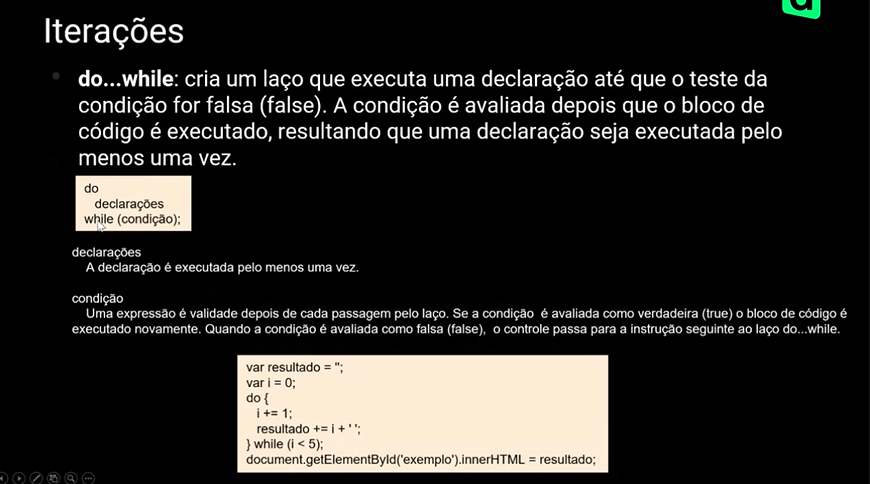


Vai da o mesmo resultado se fazer assim:

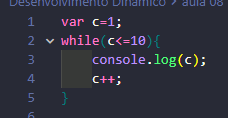


**While e do While**

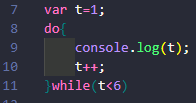




Ex – contar até 10:



Ex- se a condição t<6 for verdadeira faça isso... (ele vai fazer pelo menos 1 vez)



\*se eu colocar t<1 ele ainda vai rodar 1 vez.

Teste:

const carros = [

    { modelo: 'Audi A3', marca: 'Audi', ano: 2020 },

    { modelo: 'Jeep Compact', marca: 'Jeep', ano: 2021 }

  ];

  for (let indice in carros) {

    console.log('Carro ' + (parseInt(indice) + 1) + ':');

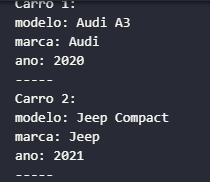
    for (let atributo in carros[indice]) {

      console.log(atributo + ': ' + carros[indice][atributo]);

    }

    console.log('-----');

  }



**Funções**

São blocos de construção fundamentais em JS.

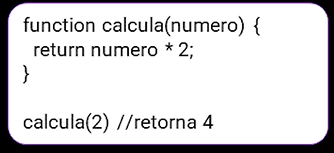
É um procedimento em JS que executa uma tarefa especifica. Ex: calcular um valor de um produto.

Quando vai repetir o código varias vezes. Utiliza-se função.

Para usar uma função, você deve definir, implementar e chamar em algum lugar no escopo do qual você quiser chamá-la.

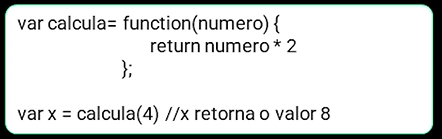
Existe 3 tipos de função:

1. **Função Regular**



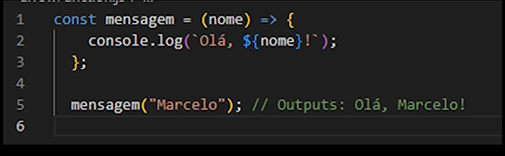
*Consiste em* ***function*** *seguido pelo* ***nome da função****, lista de argumentos para a função* ***entre parênteses******e separado por virgula*** *e a* ***declaração em JS*** *que define a função entre chaves.*

1. **Função Anônima**



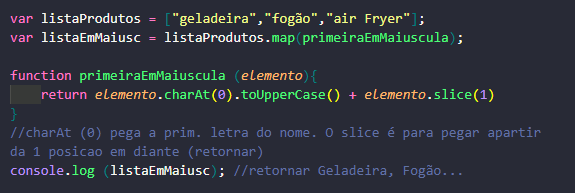
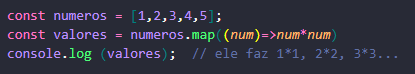
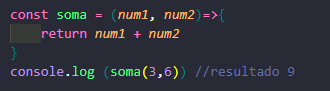
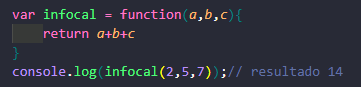
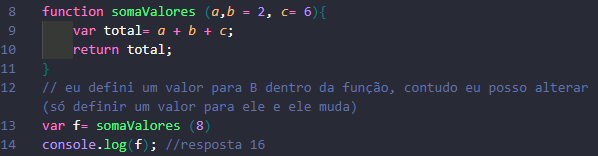
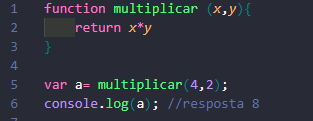
São funções sem nome, colocando o resultado da função já numa variável.

1. **Arrow Funcion**

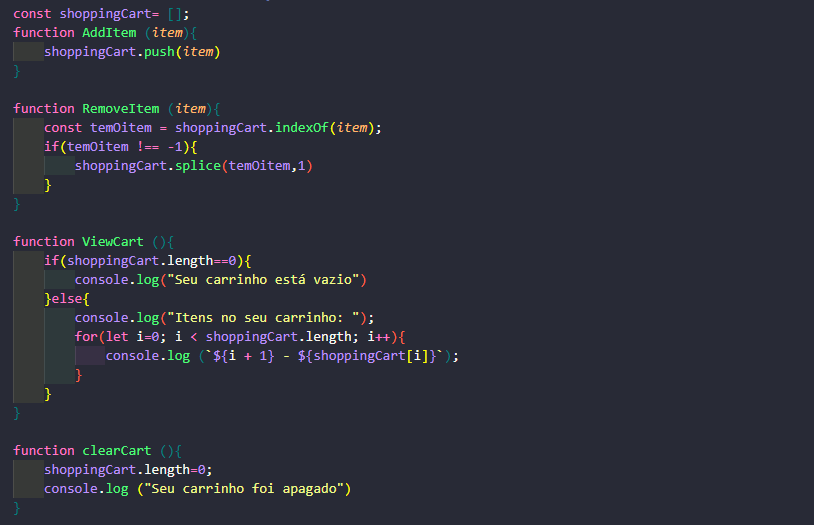


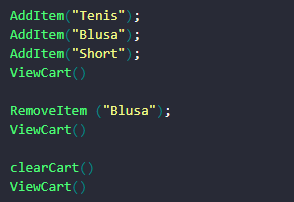
Sintaxe mais curta. (arrow pois tem =>) só coloca o argumento entre parênteses.

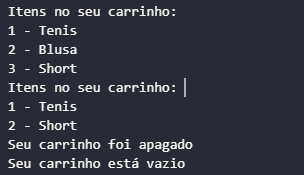
Pratica:



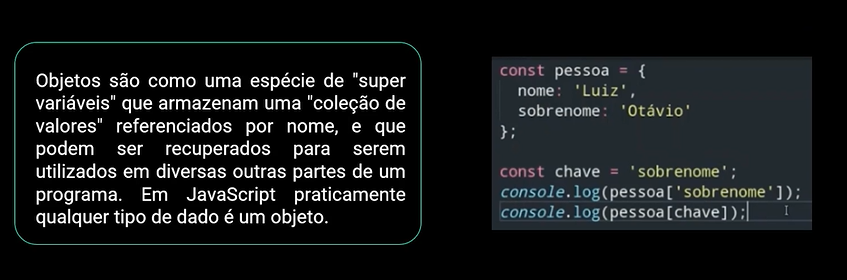
**Pratica do carrinho de compra**:





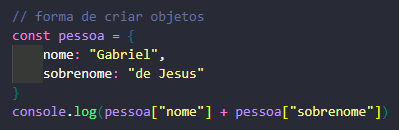


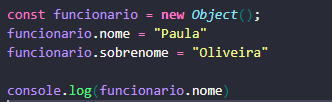
**Objetos**

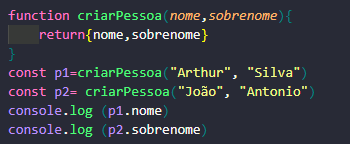


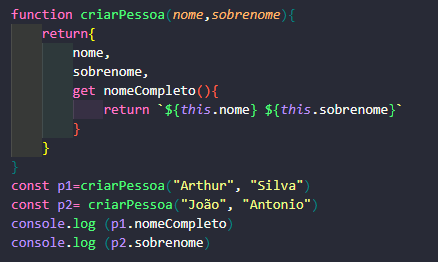
Cada item = propriedade.

Cada propriedade é composta por um par de “**nome**:**valor**”.

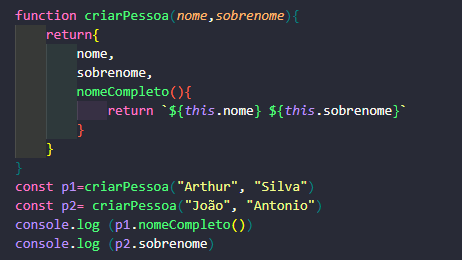




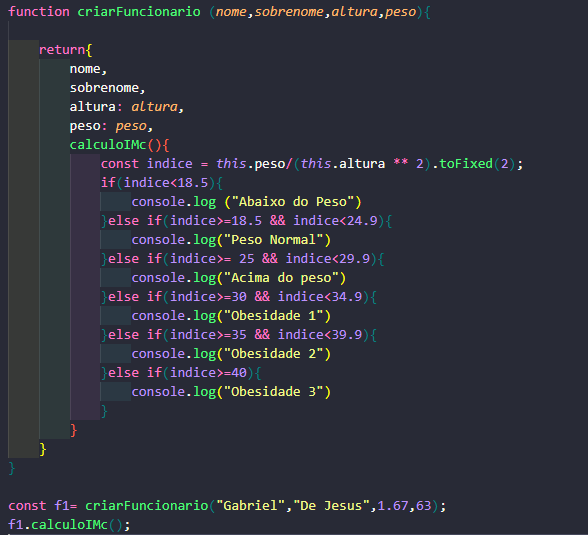




Ou sem get:



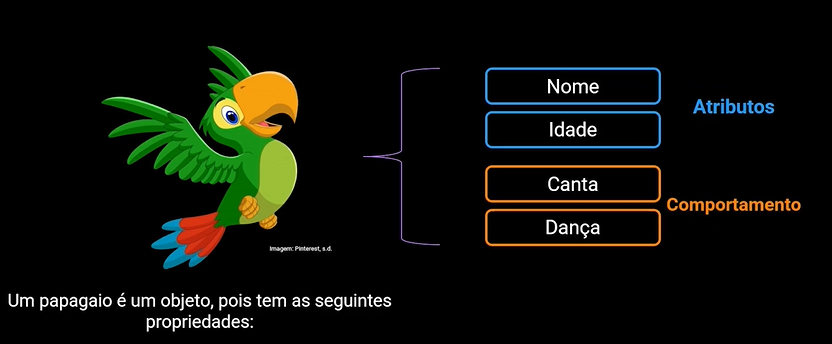
Pratica IMC:



**Classes**

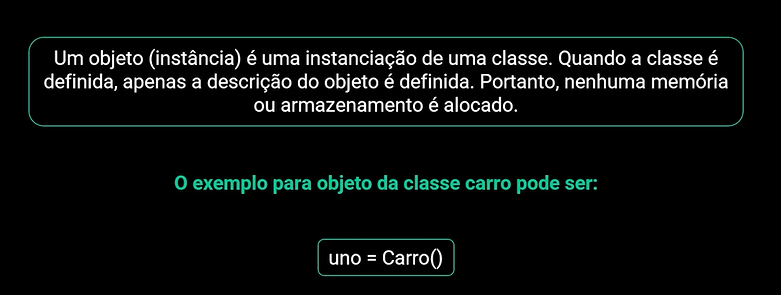
Modelo para representar objetos do mundo real. Dentro dele é comum declararmos atributos e métodos, que representam as características e comportamento desse objeto.

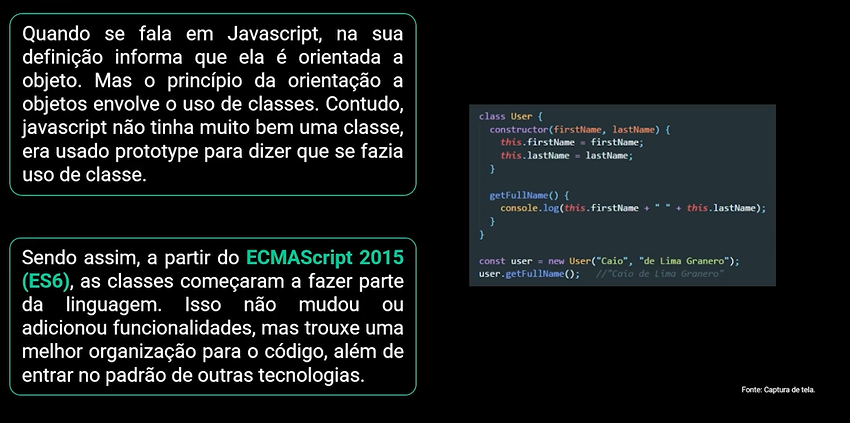




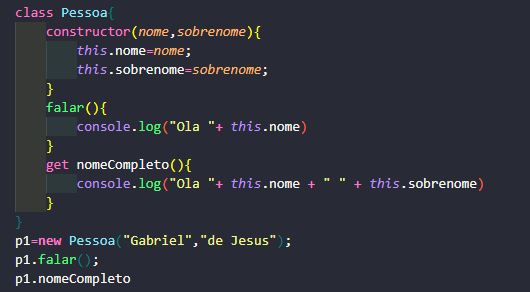


Para da vida a uma classe precisa dar nome a o objeto. (instanciar um objeto)

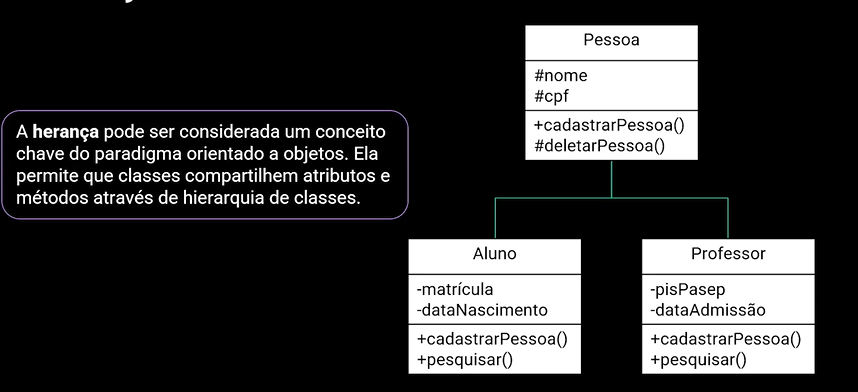


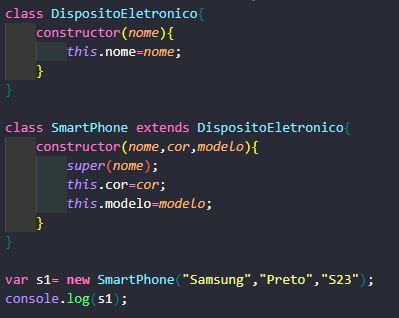






Herança





Eu aproveitei da classe dispositivo eletrônico com o **extends** e **super** (o nome)

Pratica:

