**JavaScript**

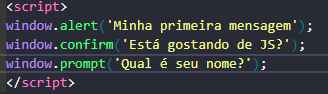
**HTML**: Conteúdo

**CSS**: Estilo

**JSS**: Interação

Geralmente em HTML o JS é feito no final do Body.// ou criado de forma externa.

No qual é criado um <script>



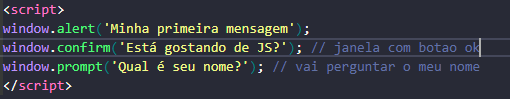
**Window.alert** (‘...’) --- aparece texto como mensagem para o usuário.

**Window.confirm** (‘...’) --- aparece texto para confirmar ou cancelar.

**Window.prompt** (‘...’) --- pergunta algo para a pessoa.

**Comentários nos códigos**:

1. //- uma única linha
2. /\* - mais de uma linha \*/



Para guardar os dados é utilizado **variáveis**.

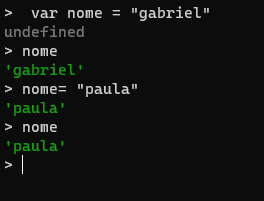
**= *(****significa recebe)*

Se eu quiser que receba nada =null

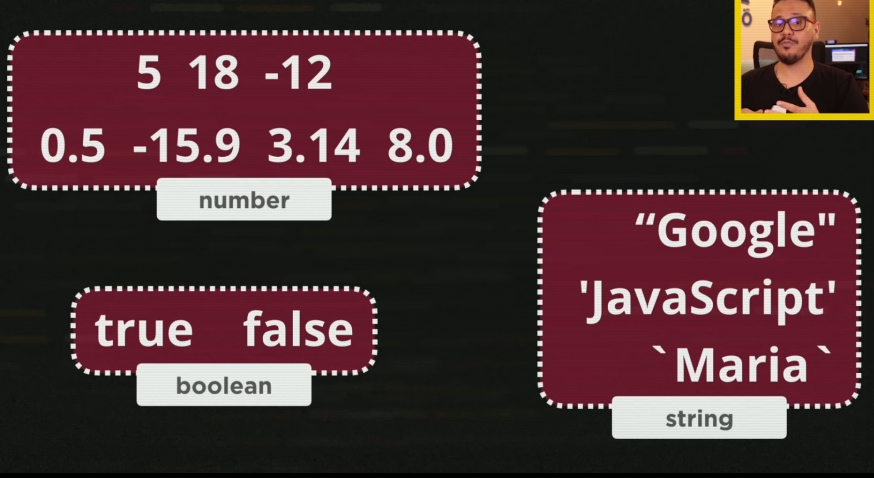
Para utilizar a variável coloca-se var ou let.

O nome depois do var é o **identificador**.

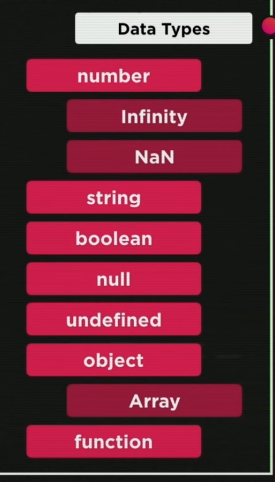
* Podem começar com **letra, $** ou **\_**
* **Não** podem **começar** com **números**.
* É possível usar **letras** ou **números**.
* É possível usar **acentos** ou **símbolos**.
* **Não** podem **conter espaços**.
* **Não** podem ser **palavras reservadas**.

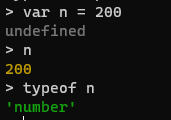


JS Data types (conhece):



Mas tem outras como: e tem o comando typeof para aprender cada tipo.



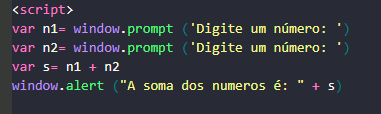


**Manipulação de dados**:

Para guardar o nome posso usar> var nome= window.prompt('Qual é seu nome?');

Para mostrar posso usar> window.alert ("é um grande prazer em te conhecer, " + nome);

**OBS**: Se eu apenas colocar assim:



\*ele me retorna o numero e o outro (tipo 1, 2 = 12)

**Dicas**:

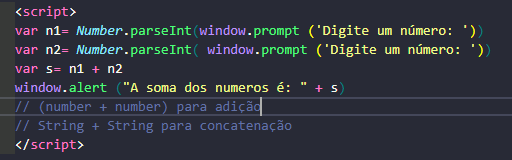
1. (number + number) para adição;
2. String + String para concatenação;

* **Fazer a conversão de String > Numero.**

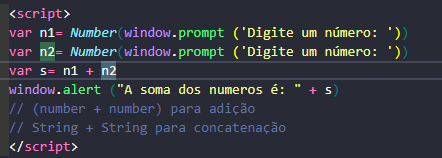
**Number.parseInt(n)** = conversão para número **inteiro**.

**Number.parseFloat(n)** = conversão para número **real**.

simplificando: só colocar **Number(n)**



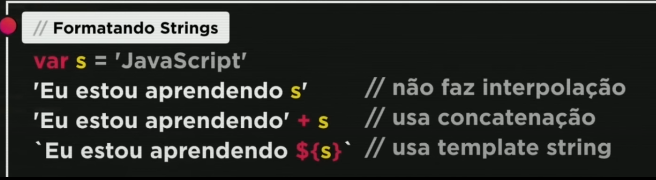
Se eu quiser real: coloco Float.



* **Fazer a conversão de Número > String.**

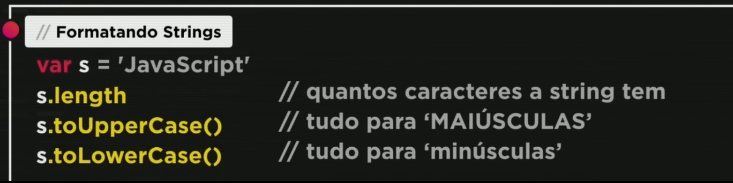
**String(n)**

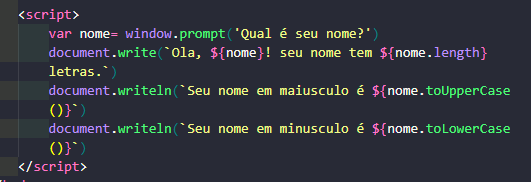
**n.tostring().**





**Obs**.: é **`**





**Document.write** = ele escreve o nome na página.

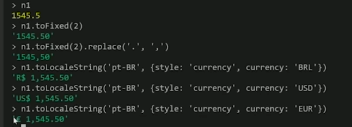
**Document.writeln** em outra linha.

**Formatando número:**

n1.toFixed(2) = fixar em 2 casas

n1.toFixed(2).replace(`.`,`,`) = para mudar o . para ,

n1.toLocaleString(`pt-BR`, {style: `currency`, currency: `BRL`}) = para colocar R$



**Operadores:**

* Aritméticos;
* Atribuição;
* Relacionais;
* Lógicos;
* Ternário;

**Aritméticos:**

Operadores: + - \* / % (- resto da divisão) \*\* (ao quadrado)

Operandos: número.

Tela de telefone celular com letras

Descrição gerada automaticamente com confiança média

Cuidado! **Ordem de prioridade.** Use parênteses () para colocar!

Texto

Descrição gerada automaticamente

Ordem precedência: **()** > **\*\*** > **\* / %** > **+ -**

Quando esta na mesma ordem, ver quem aparece primeiro da direita para esquerda.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**Auto atribuições**:

Var n=3

n=n+4 (soma o que esta depois do recebe e n torna aquele valor).

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente com confiança média

**Simplificando**: n+=4 (se n recebe ele mesmo pode colocar n+)

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente com confiança média

**Incremento**:

var x= 5

* x+=1 pode simplificar para x++ (pode colocar ++x)
* x-=1 pode simplificar para x-- (pode colocar --x)

Texto

Descrição gerada automaticamente com confiança média

**Relacionais: (vai da true or false)**

> maior

< menor

>= maior que

<= menor que

== igual

!= não é igual (diferente)

Calculadora preta com letras brancas

Descrição gerada automaticamente

Exemplo:

Texto

Descrição gerada automaticamente

**Identidade**: (o sinal de igualdade não testa o tipo!) então identidade serve para isso (**===**)

Ex: 5 == ‘5’ (verdade)

5===‘5’ (false)

Texto

Descrição gerada automaticamente com confiança média

**Lógicos**:

* ! negação – true ou false (1 valor logico).
* && conjunção (e) – (2 valores logico – true true / true false...) as 2 tem que ser true ou false.
* || disjunção (ou) – (2 valores logico – true false/ false true...) 1 tem que ser true ou false.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Idade entre 15 E 17

RJ OU SP

Salário maior que 1500 E sexo feminino

**Precedência em geral**:

Tela de um aparelho celular

Descrição gerada automaticamente

**Ternário:**

* ? : (?se verdade:se falso)

Condição? true : false

Ex: média>=6.0? ‘aprovado’: ‘reprovado’

Texto

Descrição gerada automaticamente

**DOM**

Document Object Model: Modelo de objetos para documento.

Árvore DOM: (hierárquica)

Diagrama

Descrição gerada automaticamente

Posso navegar dentro dessa arvore DOM entre os elementos! Até escrever na tela.

Posso selecionar esses elementos! Existe vários..

* Por marca;
* Por id;
* Por nome;
* Por classe;
* Por seletor;

**Por marca**: getElementByTagName () – selecionar mais de um objeto. Para selecionar o 1ª elemento por exemplo, posso usar [0] depois do comando

Texto

Descrição gerada automaticamente

\*obs quando usar **innerText** ele vem sem formatação, se usar **innerHTML** vem com a formatação.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

**Por ID**: getElementByID () – selecionar elemento por ID

**Por nome**: getElementByname () - selecionar elemento por nome

**Por classe**: getElementByClassName () - selecionar elemento por classe

Tudo segue o mesmo princípio!

Mais pratico>

**Por Seletor**: querySelector() / querySelectorall ()

Tela de celular com aplicativo aberto

Descrição gerada automaticamente com confiança média

**#** pois é ID se fosse classe seria **.**

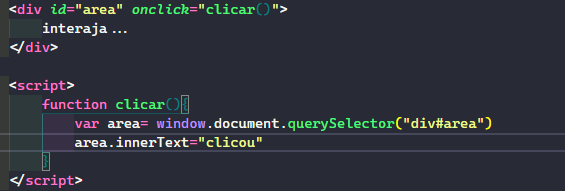
**Funções JS**:

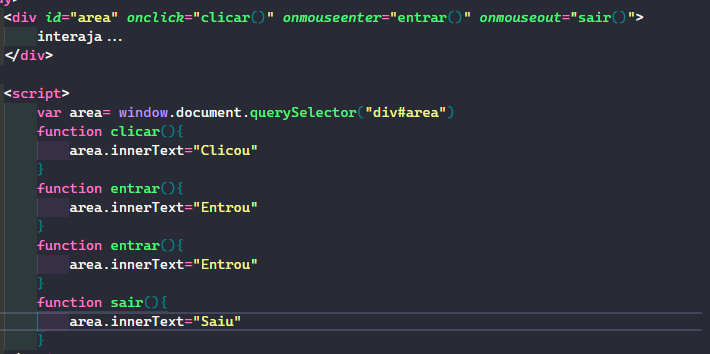
function ação (parametros){}

**Eventos DOM: (existe vários!)**

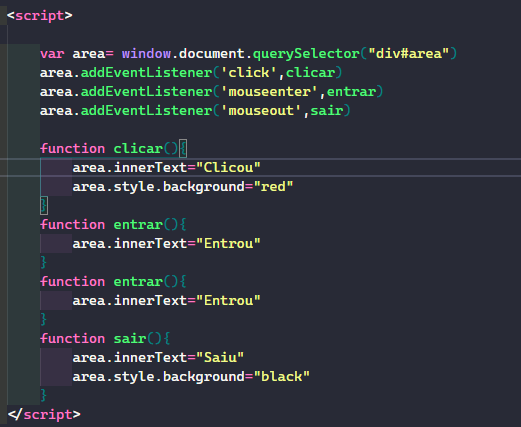
Mouse por exemplo!

Pode ser feito no HTML ou na parte do JS



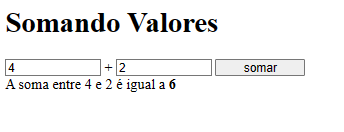


Fazendo dentro do próprio JS.



**Descobrir erro no JS: Inspecionar o elemento no navegador (aparece erro!)**





**Condições:**

* **IF e Else: (se-senao)**

If(condição){

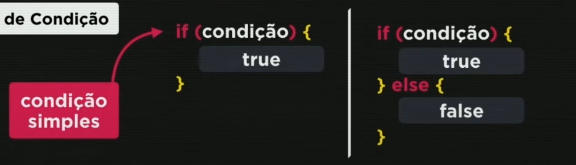
//verdadeiro

}else{

//falso

}

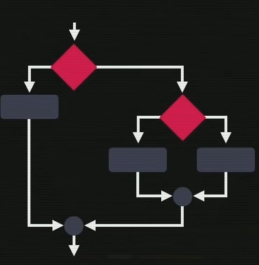
**Tipos de condições**:



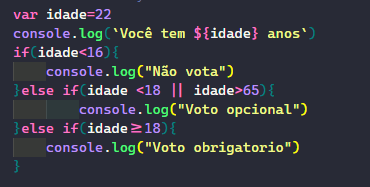
EX:



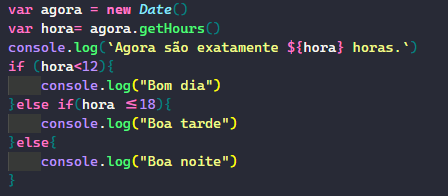
**Condições aninhadas**:



* **Elseif: (senão)**

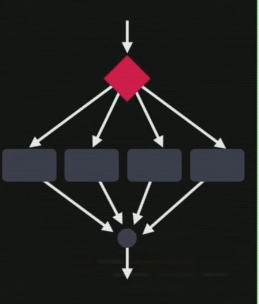


Pegar hora atual de um sistema: Utiliza o .gethours()



\*sempre vai ser new Date

**Condições múltipla**:



* **Switch: (caso)**

Switch(expressão){

Case valor 1:

//bloco

break

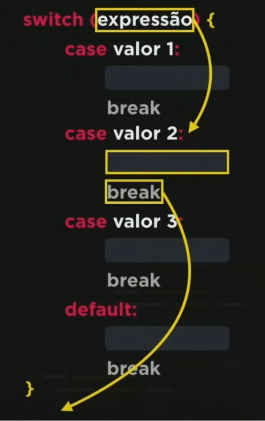
Case valor 2:

//bloco

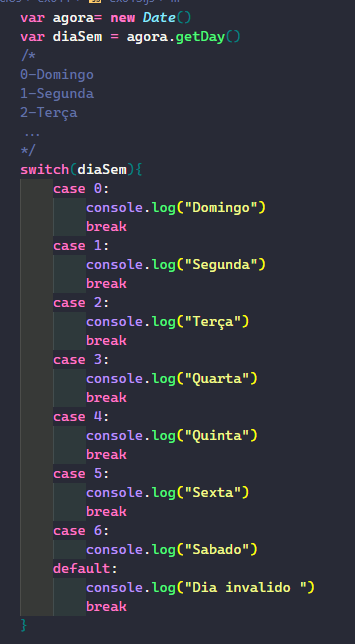
break

...Default:

}



Ex:

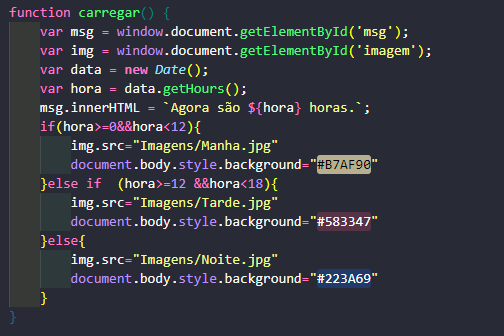


Fazendo o Exercício 012:

**HTML:**



**JS:**



Note: Estou pegando a hora do dia e alterando de acordo com a hora do computador a imagem de fundo e a cor do body.

Fazendo o exercício 13:

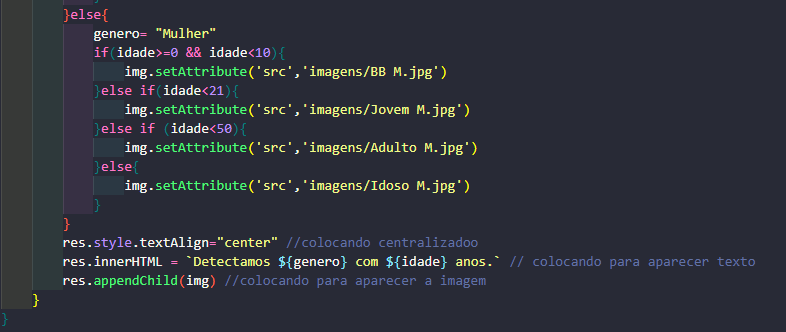
**HTML:**



Note: não tem o atributo IMG nesse html! Vamos criar de forma dinâmica.

**JS:**





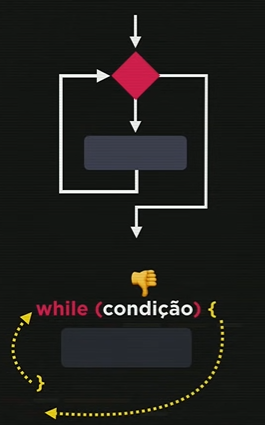
**Repetições:**

* **While: (enquanto)**

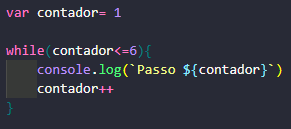
while(condição){

//

}



Ex:



* **do..while:**

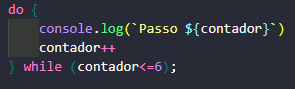
do {

//

} while (condition);

\*ele primeira testa e depois testa (ou seja, sempre testa 1 vez)

Ex:

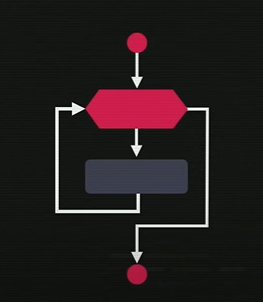


* **for: (para)**

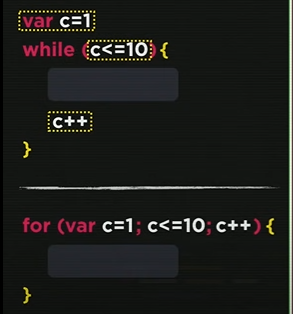
for (inicio;teste;incremento){

//

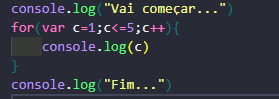
}



Comparação for e while:



Ex:



\*no executar e deburar eu consigo ver passo a passo, só colocar a variável no “inspecionar”

Fazendo o exercício 016:

**HTML:**

****

**JS:**

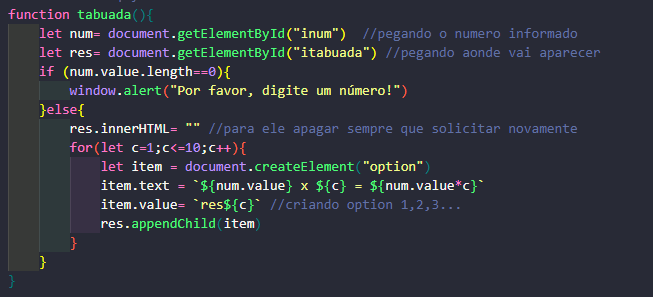


Fazendo o exercício 017:

**HTML:**



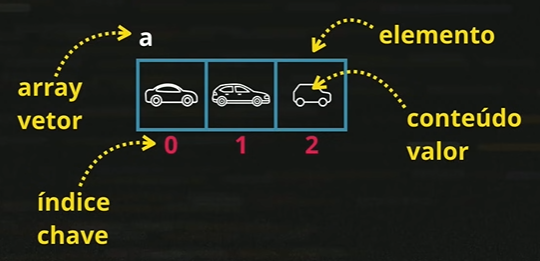
**JS:**

****

**Variaveis Composta:**

Armazena vários valores em uma mesma estrutura.

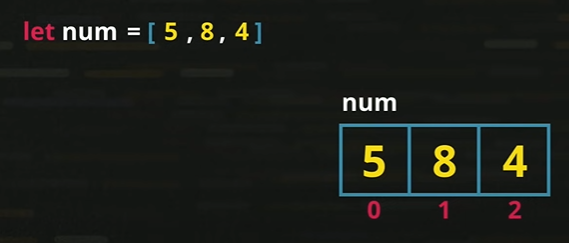
* **Array:**



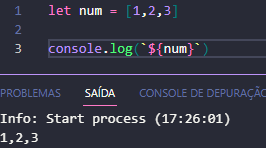
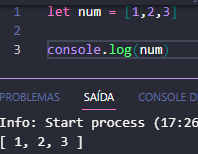
\* **Array é uma variável que tem vários elementos, cada elemento é composto por seu valor e por uma chave de identificação.**

Composição:

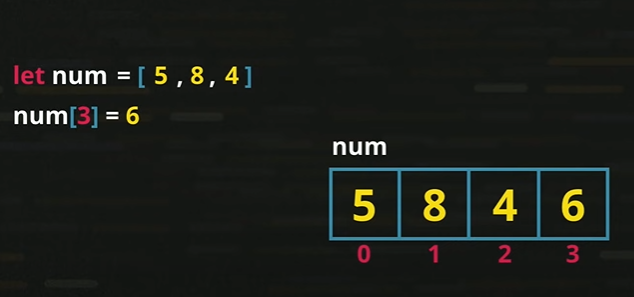
EX:



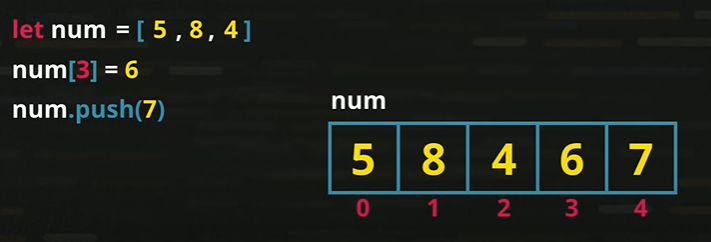
OBS: Se colocar: (`${num}`) ele mostra o resultado sem os [] se colocar só console.log num ele aparece.

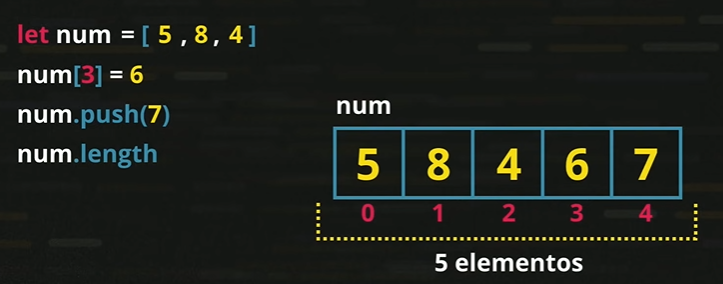
**Acrescentando**:



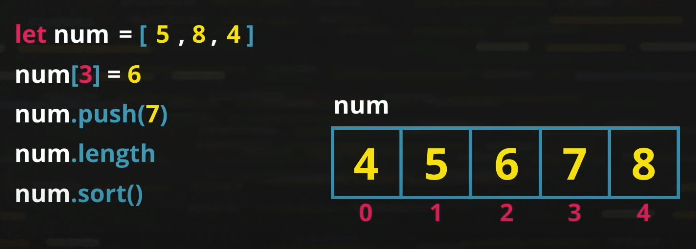
Agora se eu quiser acrecentar sempre na ultima posição (mesmo que não saiba qual é).  
Posso usar o **.push()**



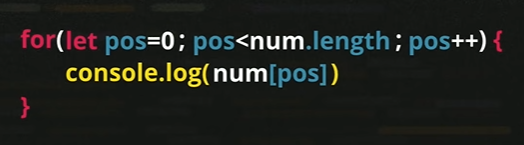
Para saber o comprimento de um array posso usar o **.length – sem parenteses.**

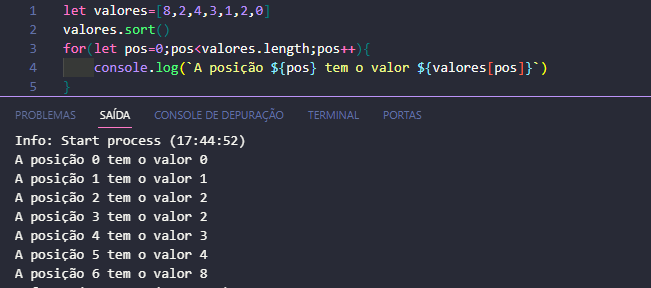


Para colocar em ordem crescente os elementos posso usar o **.sort()**

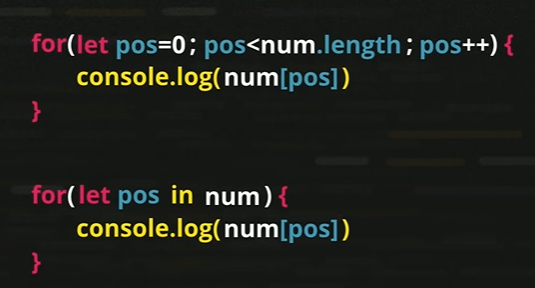


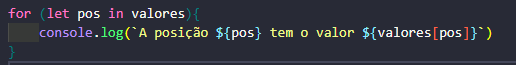
Para mostrar todos os elementos do array posso usar o **for**



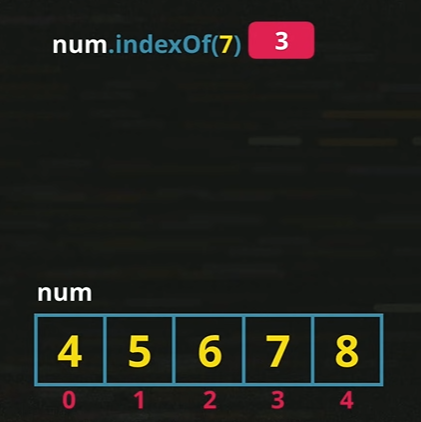


Simplificando: **for... in**



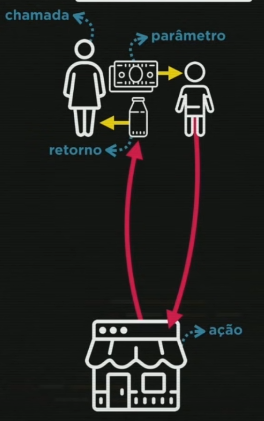


Para buscar o elemento posso usar o **.indexOf()**



Quando ele não encontra o elemento ele retorna -1.

**Funções:**



Toda função PODE TER uma chamada, parametro, uma ação e retorno.

São acões **executadas** assim que são **chamadas** ou em **decorrencia de algum evento**.

Pode receber **parametros** e retornar um **resultado**.

* **Function:**

function **ação**(**parâmetro**){

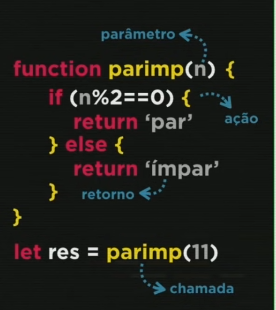
//

**return** resultado

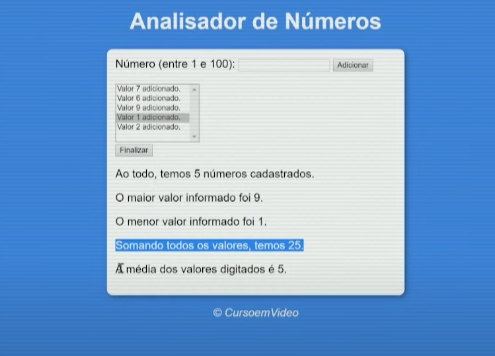
}

Chamada: **Ação(...)**

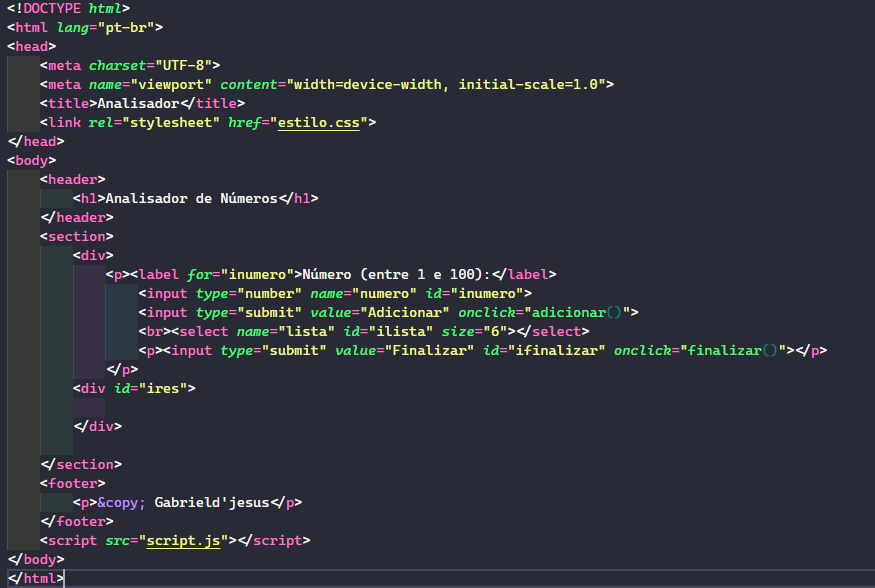
Ex:



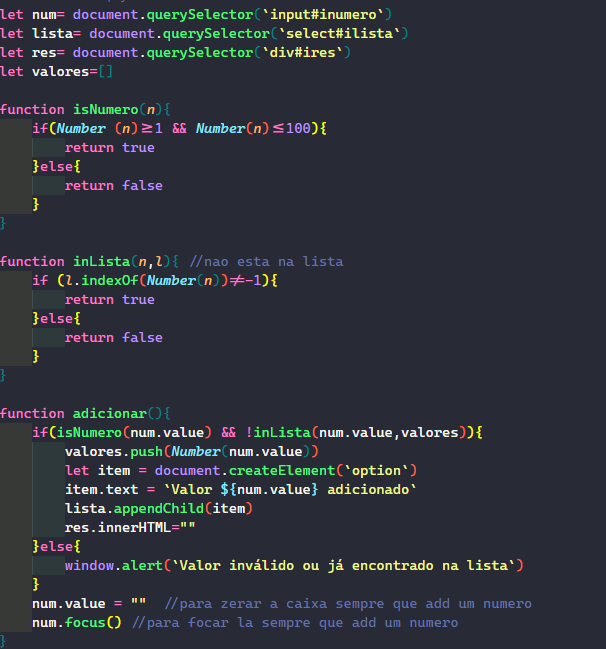
Exercicio:

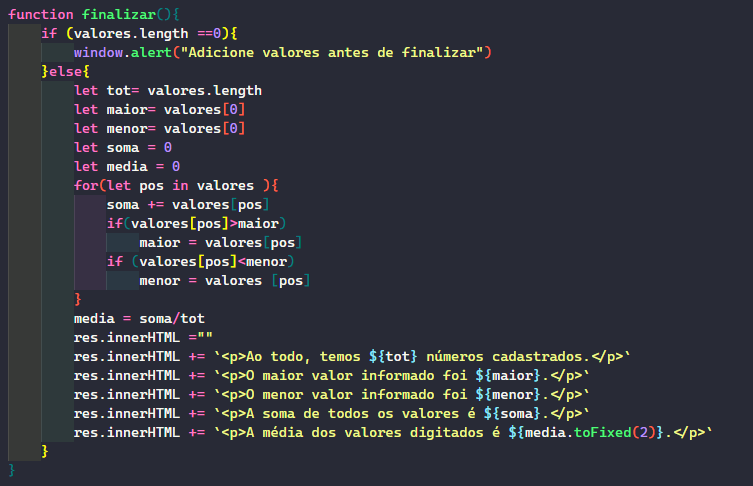


**HTML:**



**JSS:**





**Recomendado Estudar**:



Fundamentos de **Objeto**:

Declarado com chaves ({})

