```
//VAR:
//No Javascript podemos declarar variáveis de 2 formas, usando as palavras reservadas "var" e "let". Cada uma delas têm característic
as especiais quanto a sua aplicação. O var por exemplo, são variáveis sempre globais, o que significa que podem ser referenciadas de
qualquer lugar do código, não importa onde foram declaradas. O que não acontece com as variáveis let, que só podem ser referenciadas
de dentro do seu escopo. A única excessão quanto ao escopo de variáveis var é quando elas foram declaradas dentro de uma função, some
nte no caso da função elas não poderão ser chamadas de fora do escopo da função.
//Além disso, variáveis var podem ser tanto redeclaradas quanto sobrescritas, o que não acontece com as variáveis "let" que só podem
ser sobrescritas, nunca redeclaradas.




//VARS PODEM SER REDECLARADAS E SOBRESCRITAS...
var a = 2; //Perceba que redeclaramos a variável "a" logo abaixo...
let b = 2; //Já a variável "b" criada com let não a redeclaramos, simplesmente a sobrescrevemos...
var a = 3; //Perceba que redeclaramos var...
b = 3; //let não foi redeclarada, na verdade foi sobrescrita.
console.log(a, b);
//Veja no console que não é apresentado erro, pois usamos let a var como podem ser usadas, redeclaramos o var e sobrescrevemos o let.
..



//VARS PODEM SER CHAMADAS DE QUALQUER ESCOPO:
{
    {
        {
            {
                var seraQueVai = 'Foi!';//A variavel foi declarada dentro de 4 blocos
                console.log(seraQueVai);//Podem ser referenciada tanto dentro do bloco, quanto fora...
            }
        }
    }
}
console.log(seraQueVai);//Veja que mesmo no escopo global ela foi referenciada...
```

```
//CUIDADO AO USAR VAR:
var igual = 1; //Perceba que declaramos uma variável com o mesmo nome tanto fora quando dentro de um bloco...
{
    var igual = 2; //Mas a variável dentro do bloco teve o seu valor mudado...
    console.log('dentro = ' + igual); //Note que tanto dentro do bloco quanto fora, o valor de ambas será o mesmo...
}
console.log('fora = ' + igual); //Por isso devemos ter cuidado ao usar variáveis, se dermos um nome identico a ela, ela poderá trazer
 resultados que não desejamos...



//VAR DENTRO DE UM ESCOPO DE FUNÇÃO NÃO PODEM SER REFERENCIADAS DE FORA DA FUNÇÃO:
function funcao(){
    var essaSoVaiDentroDaFuncao = 'Essa só vai dentro da função';
    console.log(essaSoVaiDentroDaFuncao); //Quando chamada dentro da função ela funciona...
}
funcao(); //Perceba que a chamamos a função não temos erro nenhum
console.log(essaSoVaiDentroDaFuncao); //Mas quando chamamos só a variável á partir do contexto global o erro é apresentado...



//COMO UMA VARIÁVEL COM VAR PODE SER INFLUENCIADA PELO CONTEXTO GLOBAL:
for (var i = 0; i < 10; i++) { //Perceba como é forte a influência do var sobre o contexto global, mesmo dentro de um bloco de execuç
ão "for" ainda assim ele pode ser referenciado globalmente...
    console.log(i);
}
console.log('i = ' + i); //Veja que chamamos globalmente e ainda assim ele foi chamado...



//ESCOPO DE FUNÇÃO DO VAR:
//Outra coisa a ser levada em consideração quando se trata de escopo do var, é que o var não obedece ao escopo de função, ou seja, el
e não vai respeitar as regras do closure, se uma variável var for declarada dentro do tempo de execução dentro de uma função, os valo
res declarados sobre ela serão válidos segundo o último valor que foi declarado, o que não aconteceria com um let:
const funcs = []
for(var j = 0; j < 10; j++){
    funcs.push(function () { //usando a função push vamos incrementar valores dentro do array funcs dentro do escopo de função...
```

```
        console.log(j);
    })
}

funcs[2](); //Perceba que, quando chamamos a função juncs, os valores que são imprimidos são sempre os últimos...
funcs[8]();



//VARIÁVEIS PODEM SOFRER HOSTING:
console.log('1) ', hosting); //Referenciamos uma variável "a" que ainda não foi declarada, perceba que a variável existe, mas o seu v
alor ainda não pôde ser encontrado...
var hosting = 2; //Aqui aconteceu a declaração de "a"...
console.log('2) ', hosting); //Perceba que depois da atribuição de valores dela, nós conseguimos capturar o seu valor como resposta..
.



//HOSTING ACONTECE ATÉ DENTRO DE FUNÇÕES:
function teste() { //Perceba que o mesmo se repete na função abaixo, onde tentamos imprimir o valor de uma variável ainda não declara
da...
    console.log('3) ', hosting);
    var hosting = 'Eu tenho valor agora';
    console.log('4) ', hosting); //Somente depois da atribuição podemos ver o seu valor...
}
teste();
```

RESULTADO NO CONSOLE...

```
[Running] node "c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\014-manipulando_Var.js"
3 3
Foi!
Foi!
dentro = 2
fora = 2
Essa só vai dentro da função
0
1
2
3
4
5
6
7
8
9
i = 10
10
10
1)  undefined
2)  2
3)  undefined
4)  Eu tenho valor agora

[Done] exited with code=0 in 0.162 seconds
```

...

RESULTADO QUANDO SE TENTA REFERENCIAR UM VAR QUE ESTÁ DENTRO DE UMA FUNÇÃO...

```
[Running] node "c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\014-manipulando_Var.js"
3 3
Foi!
Foi!
dentro = 2
fora = 2
Essa só vai dentro da função
c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\014-manipulando_Var.js:49
console.log(essaSoVaiDentroDaFuncao); //Mas quando chamamos só a variável á partir do contexto global o erro é apresentado...
            ^

ReferenceError: essaSoVaiDentroDaFuncao is not defined
    at Object.<anonymous> (c:\Users\User\Documents\JAVASCRIPT\ARQUIVOS_DAS_AULAS\014-manipulando_Var.js:49:13)
    at Module._compile (internal/modules/cjs/loader.js:1072:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1101:10)
    at Module.load (internal/modules/cjs/loader.js:937:32)
    at Function.Module._load (internal/modules/cjs/loader.js:778:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:76:12)
    at internal/main/run_main_module.js:17:47

[Done] exited with code=1 in 0.171 seconds
```