Qual a diferença das variáveis:

var, let, const

var pode usar em todo código e ir mudando no decorrer da codificação

let – pode ser trocado seu valores, mas antes deve ser declarado

const – é uma variável que não pode ser mudada e tem que ser inicializada

[00:00] Ju: Nessa aula vamos nos aprofundar, um pouco, no tema de *booleanos*, falso e verdadeiro, porque ele tem algumas implicações, tem mais alguns usos que, se não sabemos no JavaScript como funciona, pode vir algum *bug* que nós não entendemos bem como é.

[00:16] Ju: Vamos, então, compartilhar a minha tela e vamos escrever alguns códigos.

[00:25] Ju: Nós conversamos, nos vídeos passados, sobre como o tipo *booleano* ou *boolean*, ele é, basicamente (ele tem dois valores), *true* ou *false*.

[00:40] Ju: E podemos atribuir variáveis a isso. Então, eu posso criar uma const usuarioLogado e ela começar como =true.

[00:54] Ju: Ou a mesma coisa, const usuarioLogado; usuarioLogado, não, vamos dar um outro exemplo. const contapaga começa sempre como = false.

[01:10] Ju: E, através desses valores nós conseguimos fazer comparação, então, se o usuarioLogado for igual à *true*, então, ele pode fazer coisas no nosso sistema, se contapaga for igual à *false*, então, posso mandar um aviso: "Sua conta não foi paga”.

[01:28] Ju: Só que, além do valor *true* e *false*, essas duas palavras; lembrando que no JavaScript elas sempre são escritas com letras minúsculas; quem veio de introdução a outras linguagens como o Python, ela pode estar em letra maiúscula. Isso é bastante importante (letra maiúscula, letra minúscula), sempre faz diferença. Se seu código tiver algum *bug*, dá uma olhada.

[01:49] Ju: Existem alguns outros valores de variáveis que não são nem *true* nem *false*, mas, que para o JavaScript, eles equivalem a ser tipo verdadeiro ou tipo falso, ou como costumamos dizer, *truthy* ou *falsy*.

[02:09] Ju: Porque eles não são, exatamente, *booleanos*, mas, podem se comportar como se fossem.

[02:15] Ju: Léo, posso continuar ou você quer adicionar alguma coisa?

[02:20] Leonardo: Não, nós podemos continuar.

[02:22] Ju: Então, vou dar alguns exemplos só para termos eles em mente.

[02:25] Ju: Vamos usar alguns exemplos sobre, por exemplo, alguns desses valores que o JavaScript pode considerar como falsos.

[02:34] Ju: Um deles é o 0.

[02:37] Ju: 0 e 1 são, em algumas linguagens (inclusive em linguagem de banco), eles fazem o papel do verdadeiro e do falso, sendo que o 0 pode ser igual o *false* e o 1 faz o papel do *true*.

[02:56] Ju: Então, o zero é um dos valores que o JavaScript considera como *false*. Vamos fazer algumas comparações só para entendermos melhor como que funciona esse valor que não é *booleano*, mas, é falso.

[03:11] Ju: Vou pedir para o JavaScript (console.log no terminal) comparar o 0 com o valor de *false*, mas, eu vou usar dois iguais ==, porque eu não quero comparar o valor e o tipo, eu quero comparar só o valor.

[03:32] Ju: Então, o 0 eu falei que é um desses valores de dados que o JavaScript considera como *false*, eu vou rodar o meu arquivo com o Node e ele retorna *true*. Ou seja, o *true* que aparece no console é o resultado desta comparação que eu estou fazendo entre 0 e o *booleano* *false*.

[03:53] Ju: Vamos fazer mais um console.log. Outro tipo de dado que o JavaScript considera como se fosse falso são as *strings* vazias, que só abrimos e fechamos aspas sem colocar nada no meio.

[04:07] Ju: Vamos fazer o mesmo tipo de comparação com dois iguais, entre *strings* vazias e *false*.

[04:12] Ju: Vou rodar de novo. Também deu verdadeiro, ou seja, o JavaScript considera esses dois valores como tipo falso.

[04:22] Ju: O que acarreta a um dado, zeros ou *strings* vazias serem considerados verdadeiro ou falso para o JavaScript?

[04:32] Ju: Isso acarreta que, às vezes, no nosso código podemos receber o tal do resultado não esperado. Porque nós esperamos que aconteça alguma coisa no nosso código, espera que o JavaScript reconheça que, por exemplo, existe algum valor e retorne *true*; mas, ele não vai retornar, ele pode errar uma comparação por causa disso. É muito importante termos isso em mente.

[04:53] Ju: Vamos passar um material extra para vocês com outros casos, inclusive o caso do *null* e do *undefined*, que são outros tipos de dados, para você já ter uma ideia melhor e fazer mais alguns testes, que são casos onde o JavaScript faz a equivalência com o falso.

[05:18] Ju: O *undefined* quando não temos uma variável declarada, mas ela não tem valor. No caso do *null* o Léo já vai explicar melhor o que é. Nós vamos entender mais detalhes, porque ele é um caso um pouco especial no JavaScript.

[05:32] Ju: Já que fizemos alguns testes como caso do falso, vamos fazer, também, por exemplo, o teste com o verdadeiro. Então, se o número 0 é considerado falso, *falsy*, então, o número 1 podemos dizer que é *truthy*, ele é do tipo verdadeiro.

[05:50] Ju: Vamos fazer um novo teste e sim, o JavaScript considera o número 1 como um valor tipo verdadeiro, *truthy*.

[06:01] Ju: Tem mais alguns casos, vamos deixar para vocês, mas, agora, podemos falar um pouco sobre o caso do *null*, para finalizarmos essa aula.

[06:10] Ju: Léo, o que você fala para nós sobre o *null*? Que ele existe, ele não é do JavaScript. O *null* é um pouco diferente no JavaScript, mas é um tipo de dado que existe em outras linguagens.

[06:22] Leonardo: O *null* é um tipo de dado que representa vazio, ele não representa nada, digamos assim.

[06:31] Leonardo: Vamos colocar em um exemplo: quando iniciamos uma variável com o *let* e deixa ela sem valor nenhum, ela ganha, automaticamente, pelo JavaScript o valor de *undefined*. Ou seja, ela não foi definida, mas, ela pode ser utilizada.

[06:49] Leonardo: Nós também podemos colocar o *null*, ou seja, vazio ou nada. Ela não está significando nada. Ambos, o *null* e o *undefined* são similares e podem ser usados, entre eles, sem problema nenhum.

[07:10] Ju: Eu vou colocar em console.log em (minhaVar) e um console.log , também, em (varNull). Só para vermos o que está acontecendo e ver o que elas retornam. Vamos lá.

[07:22] Ju: Eu vou apagar os primeiros consoles, para ficar mais fácil de visualizarmos, e vou rodar o programa, de novo.

[07:34] Ju: O primeiro resultado é *undefined*, de (minhaVar), que o Léo falou. A (minhaVar) inicializamos e não passamos nenhum valor. O espaço da variável está guardado na memória, mas, não atribuímos nenhum valor a ela. Ou seja, é um valor *undefined*, um valor indefinido.

[07:53] Ju: E, já na linha 19, a let varNull, nós atribuímos = null e *null* é um tipo de dado, então, retornou *null*.

[08:05] Leonardo: É isso!

[08:08] Leonardo: Mas, o *null* tem algumas características especiais. Vamos trabalhar nessas características que o *undefined*, por exemplo, é puramente *undefined*, é só ele. Para isso, Ju, coloca para mim uma variável do tipo número, então, let numero; e põe o número para ele, pode ser o número 3. Coloca, também, para mim, uma *string*.

[08:35] Ju: Vou escrever let texto = “Alura”.

[08:41] Leonardo: E agora vamos dar um console.log e vamos usar o comando typeof, para ver o que essa variável está armazenando.

[08:50] Ju: Certo. Então, vou fazer console.log em (typeof numero) e (typeof texto) para vermos. *Type of* é uma palavra chave que serve para perguntarmos ao JavaScript qual é o tipo de dado que está sendo guardado nessa variável.

[09:20] Leonardo: Ela acabou de rodar o programa para nós e vimos que o tipo de variável guardado em número é um tipo *number*, um número.

[09:27] Leonardo: A variável do tipo texto está guardando uma *string*, um texto, uma sequência de caracteres. Vamos fazer a mesma coisa, agora, pelas variáveis que você declarou anteriormente, que é o minhaVar e o varNull. Para vermos o que vai ter como resultado.

[09:46] Ju: Vou só substituir os consoles (typeof minhaVar) e (typeof varNull). Vou salvar, rodar e deu *undefined* em minhaVar. Só que, agora, o (typeof varNull) veio objeto, ele não veio null. E agora, Léo?

[10:10] Leonardo: Isso, ele veio como objeto. O que acontece? Como *null* foi criado no JavaScript, ele foi criado como um objeto. Ele deveria ter sido criado como *null*, só que ele é alguns dos *bugs* que tem dentro do JavaScript, que acaba ocorrendo, e que eles não pretendem consertar esses *bugs*.

[10:32] Leonardo: Por que? Porque se você consertar esses *bugs* você vai acabar quebrando muitos códigos já existentes. Então, acabou sendo deixado dessa maneira, porém, isso acaba influenciando, normalmente, no que fazemos. A menos, em casos específicos. Você tem alguma coisa para complementar, Ju?

[10:50] Ju: Não! Não tenho. Essa é uma anedota do JavaScript, alguns erros, realmente, não têm como voltar atrás. Mas, lembre-se sempre. E precisamos saber esses detalhes da linguagem, quando estamos começando, porque não saber os detalhes pode acontecer de termos *bugs* que não sabemos de onde vem e esse do *null* ser igual a um objeto é um caso muito clássico.

[11:15] Ju: E lembrando que as linguagens de programação são feitas por pessoas e as pessoas, às vezes, também cometem erros e esse foi um caso que não deu para consertar.

[11:26] Leonardo: É, infelizmente, às vezes, acontecem coisas que não esperamos.

[11:29] Ju: Sim. É isso.*Truthy*, *falsy* e o nosso caríssimo objeto *null* (ainda vamos ver bastante sobre o *null* e o *undefined* por aí).

Conversão implícita – converte um dado em outro ex. um numero em string

= -> atribuir valor a uma variável

== -> comparação somente do valor

456 == “456” = true

=== -> compara não somente o valor, mas tmb o tipo

456 === “456” = false

Conversão explicita

Transforma uma variável string em number e virse e versa

Ex. Number(varcontendoumastring)