O que são variáveis?

Variável é um dos conceitos mais importantes no estudo de programação, independente da plataforma ou linguagem utilizada. Uma variável faz referencia a um espaço na memória do computador utilizado para guardar informações que serão usadas em seus programas.

Para elucidar o conceito, imagine que a memória de seu computador é um armário com 100 gavetas e você guarda cada tipo de objeto em uma gaveta diferente. Provavelmente você vai querer criar etiquetas para referenciar o que guarda em cada gaveta.

Mas por que o nome "variável"?

Porque uma variável pode ter o seu valor alterado durante a execução de um programa. Ainda na analogia anterior, suponha que há uma gaveta "Meias" com 2 meias; se necessário, pode-se adicionar mais 5 meias e, caso seja um colecionador, até 200. Assim a gaveta "Meias" muda o seu valor.

Exemplo1:

<script >

```
 var nome = "Gabriel Mendonça";
 var idade = 25;
</script>
```

Exemplo2:

```
<script >
 var nome = prompt('Digite seu nome: ');
 alert(nome + ', seja bem vindo!');
</script>
```

Exemplo 3:

```
<script >
 /* Este é um script para cálculo de idade! */

 // Declara o ano atual para fazer o cálculo
 var anoAtual = 2014;

 // Pede que o usuário digite o ano em que nasceu
 var anoNascimento = prompt('Digite o ano em que você nasceu.');

 // Calcula a idade do usuário e armazena na variável idade
 var idade = anoAtual - anoNascimento;

 // Mostra ao usuário a idade que ele possui
 alert("Sua idade é: " + idade + " anos");
</script>
```

Variáveis e constantes

Em JavaScript, uma variável é declarada usando-se a palavra *var*, seguida do nome da variável. O nome de uma variável deve começar por uma letra ou o símbolo "_", seguido por qualquer quantidade de letras, números e de caracteres "_" e $. Além disso, os nomes de variáveis são case *sensitive*, ou seja, o JavaScript

diferencia letras maiúsculas e minúsculas, assim como acontece por exemplo na linguagem Java.

O trecho que segue apresenta algumas declarações válidas de variáveis:

```
<script >
   var mensagem = "Alô Mundo!!!";
   var idade = 23;
   var altura = 2.32;
   var esta_contido = true;
</script>
```

Em JavaScript não é necessário haver a especificação do tipo da variável. Nessa linguagem a declaração de variáveis é feita apenas como indicado anteriormente, com o nome e a palavra-chave *var*. Isso cria a possibilidade de uma variável assumir valores de tipos diferentes ao longo da execução de um script! É isso mesmo! Em alguns momentos do programa uma variável pode estar armazenando um número, depois pode estar armazenando uma string e assim por diante.

No exemplo anterior, a variável mensagem é do tipo string, enquanto as variáveis idade e altura são do tipo inteiro e decimal, respectivamente. Veja a seguir um exemplo de código JavaScript que muda o tipo do dado armazenado em uma variável:

```
var num = 1;
num = “agora armazeno uma string”;
```

As constantes possuem a mesma função das variáveis, mas seu valor não pode ser alterado. Para se declarar constantes, usa-se a palavra *const* antes do nome da constante.

```
<script >
    const maximo = 100;
</script>
```

Assim como outras linguagens de programação, as variáveis podem ser locais ou globais. Variáveis globais podem ser usadas por qualquer trecho de código JavaScript presente na página Web (desde que o código venha depois da declaração da variável). Elas são declaradas fora e antes de todas as funções que a usam, mas podem estar contidas em qualquer local da página.

Já as variáveis locais são declaradas e invocadas somente dentro da função. Como pode ser visto, a variável mensagem é declarada na linha 2 e usada nas linhas 6 e 11, enquanto a variável numero1 e numero2 são locais, sendo definidas dentro das duas funções nas linhas 5 e 10, respectivamente.

```
<script >
   var mensagem = "O número é: ";
   function imprime1(){
      var numero1 = 1;
      alert(mensagem);
      alert(numero1);
   }
   function imprime2(){
      var numero2 = 2;
      alert(mensagem);
      alert(numero2);
   }
</script>
```

Declaração de Variáveis

Novas variáveis em Javascript são declaradas utilizando uma dessas três palavras-chave: *let, const* ou *var.*

O *let* permite que você declare variáveis com escopo de bloco. Dessa forma, a variável declarada é acessível somente dentro do bloco em que ela foi declarada.

O *const* permite que você declare variáveis cujos valores não serão modificados, ou seja, valores constantes. A variável estará acessível dentro do bloco no qual ela foi declarada.

Já o *var* é a palavra-chave mais comum para declarar uma variável. Ela não tem as restrições que as duas outras palavras-chave anteriores têm. Uma variável declarada com a palavra-chave *var* é acessível de qualquer lugar da função em que ela foi declarada, ou seja, ela não tem escopo de bloco, e sim escopo de função.

Se você declarar uma variável sem atribuir um valor a ela, seu tipo ficará como *undefined*.

O que é escopo?

Em poucas palavras, escopo é a propriedade que determina onde uma variável pode ser utilizada dentro de um programa, por exemplo, se você declara uma variável dentro de uma função, só aquela função consegue utilizar a variável, e se você declara uma variável fora de uma função, ela pode ser acessada de qualquer parte daquele mesmo script, pois ela foi declarada globalmente.

No Javascript existem 3 tipos de escopos, o global, local e de bloco (adicionado no ES6). Vamos ver em detalhes como cada um deles funciona para var e let a seguir.

Escopo global

Quando você declara uma variável fora de qualquer função, seja ela var ou let, ela tem o escopo global, pois qualquer função no script consegue utilizar esta variável. Veja abaixo um exemplo de uma variável global declarada com var:

```
var minhaVariavelGlobal = 10;
function func1(){
  console.log(minhaVariavelGlobal);
}
```

```
function func2(){
  console.log(minhaVariavelGlobal);
}

function func3(){
  console.log(minhaVariavelGlobal);
}
console.log(minhaVariavelGlobal);
```

Neste exemplo, todas as funções conseguem utilizar a variável, pois ela está declarada globalmente. Agora vejamos um exemplo com let:

```
let minhaVariavelGlobal = 10;
function func1(){
  console.log(minhaVariavelGlobal);
}
function func2(){
  console.log(minhaVariavelGlobal);
}
function func3(){
  console.log(minhaVariavelGlobal);
}
console.log(minhaVariavelGlobal);
```

O resultado é exatamente o mesmo, então isso significa que o escopo global serve tanto para var, como para let.

Escopo local

Uma variável é local quando ela é declarada dentro de alguma função, pois isso significa que apenas aquela função consegue enxergá-la. As demais funções do script não conseguem utilizá-la, pois a mesma foi declarada dentro de uma função específica. Neste ponto de vista, podemos dizer que o escopo local é completamente diferente do escopo global.

Abaixo vamos ver um exemplo de uma variável declarada localmente com var.

```
// Consegue manipular a variável victor

function func1(){
  var victor = "Lima";
  console.log(victor);
}

// NÃO Consegue manipular a variável victor

function func2(){
  console.log(victor);
}

// NÃO Consegue manipular a variável victor

function func3(){
  console.log(victor);
}
```

Como você pode ver nos exemplos de código acima a func2 e a func3 não conseguem acessar a variável victor, pois ela foi declarada dentro do corpo da func1. O mesmo comportamento se repete para variáveis declaradas com let, até agora, nada de diferente.

Escopo de bloco

E é aqui a grande diferença entre var e let. O escopo de bloco. Primeiro, para entender como esse escopo funciona, precisamos entender o que é um bloco na linguagem Javascript.

Podemos dizer que um bloco é tudo aquilo em um código que está entre chaves ({ }), ou seja, estruturas condicionais, estruturas de repetição, entre outras entidades que trabalham com blocos.

Então, em resumo, uma variável com escopo de bloco, é uma variável que foi declarada dentro de um determinado bloco, e apenas pode ser manipulada dentro daquele bloco, e nenhuma outra parte do código pode manipulá-la. Dentre todos os tipos de escopo este é o mais restrito.

Este tipo de escopo só funciona com variáveis declaradas com a palavra chave let e esta é a grande diferença entre let e var.

O escopo de bloco abaixo é um exemplo de uma variável let declarada dentro de um bloco if.

```
var condicao = true;
if(condicao) {
  let minhaVariavel = 10;

  // É possível manipular a let, pois ela foi declarada neste bloco
  console.log(minhaVariavel);
}

// ERRO, Não é possível acessar a variável, pois ela tem escopo de bloco.

function func1(){
  console.log(minhaVariavel);
}

// ERRO, Não é possível acessar a variável, pois ela tem escopo de bloco.

function func2(){
  console.log(minhaVariavel);
}
```

```
// ERRO, Não é possível acessar a variável, pois ela tem escopo de bloco.

function func3(){
  console.log(minhaVariavel);
}

// ERRO, Não é possível acessar a variável, pois ela tem escopo de bloco.

console.log(minhaVariavel);
```

O Mesmo não se aplica quando você declara uma variável com var, pois o escopo de bloco não existe na palavra reservada var.

Abaixo vejamos um exemplo.

```
var condicao = true;
if(condicao){
  var minhaVariavel = 10;
  console.log(minhaVariavel);
}

// Funciona, pois o escopo de bloco não existe para var

function func1(){
  console.log(minhaVariavel);
}

// Funciona, pois o escopo de bloco não existe para var

function func2(){
  console.log(minhaVariavel);
}

// Funciona, pois o escopo de bloco não existe para var
```

```
function func3(){
  console.log(minhaVariavel);
}

// Funciona, pois o escopo de bloco não existe para var

console.log(minhaVariavel);
```

Definindo nomes de variáveis

Quando criamos variáveis temos de levar em consideração algumas regras específicas da linguagem:

Uma variável é case-sensitive

Isso significa que nomes com letras maiúsculas são diferentes de nomes com letras minúsculas: para um programa em JavaScript, Nome é diferente de nome.

Caracteres válidos

Palavras: embora a maioria dos navegadores já reconheça uma variedade de caracteres UTF-8 (como palavras com acentos e "ಠ_ಠ", por exemplo), o recomendado é o uso apenas de letras MAIÚSCULAS e minúsculas, sem acentos e espaços.

```
var meuNOME = "joão";
var meunome = "josé";
var MeuNoMe = "Thiago";
var ಠ_ಠ = "JavaScript é legal!"; // Não recomendado, entretanto
```

Números: desde que sejam precedidos por uma ou mais letras.

```
var camisa9 = "ronaldo";
var camisa230lateral = "joaozinho";
```

Underline e cifrão: "_" e "$" também são permitidos em qualquer posição, mas pouco usados.

```
var _nome = 'Gabriel'
var segundo_nome = "Mendonça"
var $ultimo_nome_ = ""
```

Nomes reservados pela linguagem

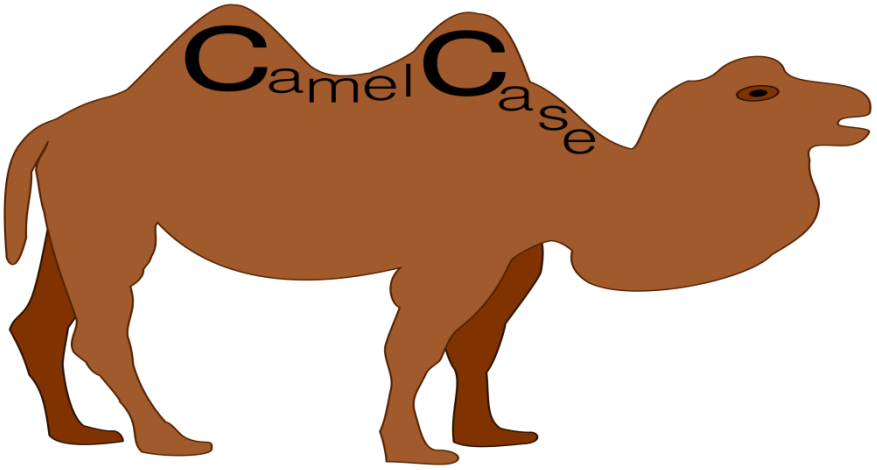
Alguns nomes não podem ser utilizados para criação de variáveis pois estão reservados de alguma forma à linguagem.

São eles:

- abstract
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- default
- do
- double
- else
- extends
- false
- final
- finally
- float
- for
- function
- goto
- if
- implements
- import
- in
- instanceof
- int
- interface
- long
- native
- new
- null
- package
- private
- protected
- public
- return
- short
- static
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- true
- try
- var
- void
- while
- with

CamelCase

O que é CamelCase? CamelCase é uma maneira de separar as palavras em uma frase colocando a primeira letra de cada palavra em maiúscula e não usando espaços. É comumente usado em URLs da web, programação e convenções de nomenclatura de computadores. É nomeado como caixa de camelo porque as letras maiúsculas lembram as corcovas nas costas de um camelo .



Definição dos Tipos de variáveis

Vamos começar pelos conceitos mais básicos de qualquer linguagem: os tipos de dados. Algoritmos em Javascript manipulam **valores,** e esse valores **pertencem a um tipo**.

Os tipos de dados em Javascript são:

- Números: 125, 14.24586;
- Strings: "JavaScript";
- Lógicos (Boolean): true, false;
- Objetos: que falaremos com detalhes em outra aula;
- null: valor primitivo especial;
- undefined: propriedade cujo valor não foi inicializado.

✔ ***Números*** suportam inteiros e números reais. Internamente são todos tratados como números reais, ou seja, são tratados nativamente como números de ponto flutuante.

- ✔ ***Strings*** são sequências de caracteres codificados em unicode. Definimos esse tipo utilizando aspas simples ou duplas. Para saber o comprimento de uma string, basta acessar a sua propriedade length.
- ✔ Valores ***booleanos*** são bastante simples, trata-se de um tipo de dado que só pode assumir dois valores: true ou false.
- ✔ ***Um objeto*** em Javascript é uma coleção de propriedades, e uma propriedade é uma associação entre um nome (ou chave) e um valor.
- ✔ O tipo ***null*** denota um valor nulo, atribuído deliberadamente.
- ✔ O ***undefinied*** normalmente indica uma variável não inicializada.

A tabela a seguir faz um resumo dos tipos de dados primitivos que as variáveis JavaScript podem assumir.

Tipo	Descrição
Números	Podem ser números inteiros ou números decimais (valores com ponto).
Cadeias de caracteres	Qualquer cadeia de caractere entre ' (aspas simples) ou " (aspas duplas).
Booleanos	As palavras true para verdadeiro e false para falso.
Nulo	Valor null que representa a ausência de um valor.

Quando falamos em tipos de variáveis, temos as linguagens chamadas de fortemente tipadas e fracamente tipadas.

Em linguagens fortemente tipadas, definimos o tipo da variável no momento de sua criação. Exemplos de linguagens do tipo: Java; C; C++.

Em linguagens fracamente tipadas, não precisamos definir o tipo da variável, ela é tipada automaticamente quando recebe um valor. Exemplos de linguagens do tipo: Python; Ruby; Javascript.

Exemplos:

Int - Variáveis com valores inteiros.

```
var idade = 17;
var graus = -3;
var pontos = 0;
var numeroGrande = 2000009283;
```

Float - Variáveis com ponto flutuante ou casas decimais.

```
var peso = 32.59345;
var PI = 3.14;
var meu_saldo = -1034.32
```

Observação:

Em linguagens de programação utiliza-se "." (ponto final) para separar casas decimais em vez da ',' (vírgula).

String - Variáveis de texto, normalmente chamada de "cadeia de caracteres".

Os valores desse tipo são atribuídos utilizando aspas duplas (") ou aspas simples (') como delimitador.

```
var nome = "Gabriel Mendonça";
var data_nascimento = "17 de Junho de 1988";
var email = "gabriel@host2.com.br";
var tempo = "20s";
```

Observação:

Tudo o que é declarado entre os delimitadores (") ou (') é entendido como parte da string, mesmo que sejam números.

Booleanos - Tipo de dado de dois valores: "true" (verdadeiro) ou "false" (falso).

```
var verdadeiro = true;
var verdadeiro2 = 1;
var falso1 = false;
var false2 = 0;
var falso3 = null
```

Observação:

Apesar dos valores true e false representarem, respectivamente, os valores "verdadeiro" e "falso", pode-se utilizar outros valores para essa representação, como exemplificado acima.

Arrays - Um array referencia a vários espaços na memória.

É um conjunto de valores e/ou variáveis organizadas por índice (que pode ser um valor inteiro ou string). O entendimento desse tipo é muito importante.

Retomando a analogia inicial: imagine que em uma gaveta você queira armazenar bebidas. Nela teria suco, refrigerante, café, água mineral... Perceba que esses itens formam uma lista e que todos os itens são bebidas, ou seja, itens de uma mesma categoria. Com o uso de arrays evitamos declarar diversas variáveis para um mesmo grupo (bebida1, bebida2), acessando os itens pelos seus respectivos índices: bebida[1], bebida[2]. Se você perceber, declarar várias variáveis para uma mesma coisa usaria desnecessariamente várias gavetas, enquanto declarar um array para armazenar todas as "mesmas coisas" usaria apenas uma.

Pense que, enquanto uma variável é uma casa, um array é um prédio; ou que um array é como um gaveteiro.

Operadores

Os operadores são úteis para manipular as variáveis, em JavaScript existem diversos operadores, veja quais são eles.

a. Operadores Matemáticos

São os operadores que permitem a realização de operações aritméticas sob as variáveis.

Símbolo	Significado	Exemplo
+	Adição	x + 7
-	Subtração	x - 7
*	Multiplicação	x * 8
/	Divisão	x / 8
%	Resto	x % 8
=	Atribuição	x = 8

b. Operadores lógicos

São os operadores que permitem a realização de operações lógicas sob as variáveis. São normalmente utilizados em comandos condicional (como o if) e repetições (como o while).

São aplicáveis a valores booleanos (true e false) e seu resultado também é um booleano.

Símbolo	Nome	Descrição	Exemplo
&&	E	Verdadeiro se as duas condições são verdadeiras	x && y
||	Ou	Verdadeiro se pelo menos uma das condições é verdadeira	x || y
!	Negação	Retorna o valor booleano inverso	!x

c. Operadores de Comparação

São os operadores que permitem a realização de operações de comparação sob variáveis. São muito utilizados em comandos condicional (como o if) e repetições (como o while), pois o resultado de sua avaliação é um valor booleano.

Símbolo	Significado	Exemplo
==	Verifica igualdade	x == 11
<	Verifica se valor menor que outro	x < 11
<=	Verifica se valor menor ou igual a outro	x <= 11
>	Verifica se valor maior que outro	x > 11
>=	Verifica se valor maior ou igual a outro	x >= 11
!=	Verifica se valor diferente de outro	x != 11

d. Operadores combinados

São operadores que simplificam a sintaxe de operações aritméticas bem comuns, servindo como atalho mais compacto da operação.

Símbolo	Significado	Exemplo	Alternativa
+=	Incrementa valor	x += y	x = x + y
-=	Decrementa valor	x -= y	x = x - y
*=	Multiplica por	x *= y	x = x * y
/=	Divide por	x /= y	x = x / y
x++	Incrementa em uma unidade	x = y++	y = y + 1 e x = y
x--	Decrementa em uma unidade	x = y--	y = y - 1 e x = y

Os operadores aritméticos do Javascript são:

- adição: +
- substração: -
- multiplicação:*
- divisão: /
- resto da divisão (módulo): %

```
x = 3 + 2;
y = 17 - 4.5;
w = 3 * 2;
z = 14/2;
```

Valores são atribuídos utilizando o operador =. Podemos usar também atribuições compostas, como o += e o -=. Você também pode utilizar o ++ e o -- para incrementar e decrementar valores, respectivamente.

```
x = 15%2;
y += 5; // equivalente a y = y + 5;
x++;
```

```
y--;
```

Comparações com Javascript são feitas com <, >, <= e >=. Utilizamos o && para conjunção (e/and) lógica e o || para a disjunção lógica (ou/or).

A igualdade é feita utilizando os operadores == e ===. O operador de dupla igualdade (==), também conhecido como igualdade ampla, realiza a coerção de tipo, ou seja, convertendo ambos os valores para um tipo comum. Já com o operador de tripla igualdade (===), também conhecido como igualdade estrita, nenhum valor é convertido implicitamente para algum outro valor antes de serem comparados. Veja alguns exemplos:

```
123 == '123'; // true
1 == true; // true

123 === '123'; // false
1 === true;    // false
```

Sobre os Operadores

Os operadores vão nos permitir fazer operações matemáticas, de comparação e lógicas.

Aritiméticos

Para as operações matemáticas básicas são utilizados os seguintes, adição(+), subtração(-), multiplicação(*) e divisão(/).

```
//Adição

2+2 //4
2.3+4 //6.3
1.5+1.5 //3

//Subtração
```

2-2 //0

8-5 //-8

3.2-1 //2.2

//Multiplicação

*2*3 //6*

*1.5*2 //3*

//Divisão

1/2 //0.5

1.5/2 //0.75

2/0 //Infinity

Você notou que podemos ter resultados com casas decimais e que é retornada a constante Infinity em qualquer número divido por zero.

Além desses operadores básicos temos outros operadores bem úteis:

Resto (%)

Retorna o resto inteiro da divisão.

5%4 //1

4%5 //4

Incremento (++)

Adiciona um a variável. Se utilizado antes(++x) adiciona um e retorna o valor, caso o operador seja utilizado depois da variável(x++) retorna o valor e adiciona um.

var x = 1;

++x //2

x++ //2

Decremento (--)

O comportamento desse operador é parecido com o de incremento(acho que você já entendeu).Ele subtrai um da variável. Se utilizado antes(--x) subtrai um e retorna o valor, caso o operador seja utilizado depois da variável(x--) retorna o valor e subtrai um.

```
var x = 2;
--x //1
x-- //1
```

De comparação

Igual (==)

Retorna verdadeiro se os valores comparados forem iguais.

```
1=='1' //true
```

Não igual (!=)

Retorna verdadeiro se os valores comparados não forem iguais. Esse operador também pode ser chamado de diferente de.

```
4!=1 //true
```

Igual estrito (===)

Esse operador é mais severo, só retorna verdadeiro se o valor e o tipo comparados forem iguais.

```
3==='3' //false
3===3 //true
```

Não igual estrito (!==)

Não se engane, esse operador vai retornar verdadeiro se os valores e ou os tipos forem diferentes

```
3!=='3' //true
3!==3 //false
```

```
3!==4 //true
```

Maior que (>)

Compara se o operador da esquerda é maior que o da direita. Caso seja retorna verdadeiro

```
1>2 //false
5>3 //true
4>'1' //true
```

Maior ou igual que (>=)

Compara se o operador da esquerda é maior ou igual ao da direita. Caso seja retorna verdadeiro

```
1>=2 //false
5>=3 //true
4>='1' //true
2>=2 // true
```

Menor que (<)

Compara se o operador da esquerda é menor que o da direita. Caso seja retorna verdadeiro

```
1<2 //true
5<3 //false
4<'1' //false
```

Menor ou igual que (<=)

Compara se o operador da esquerda é menor ou igual ao da direita. Caso seja retorna verdadeiro

```
1<=2 //true
5<=3 //false
4<='1' //false
```

```
2<=2 // true
```

Lógicos

Operadores lógicos são utilizados normalmente com expressões que retornam verdadeiro ou falso, entretanto caso seja utilizado com valores não booleanos o retorno será não booleano.

O JavaScript fornece três operadores lógicos:

- *!* Operador lógico NÃO (Logical NOT)
- *||* Operador lógico OU (Logical OR)
- *&&* Operador lógico E (Logical AND)

Os operadores lógicos são importantes no JavaScript porque permitem comparar variáveis e fazer algo com base no resultado dessa comparação.

Por exemplo, se o resultado da comparação for verdadeiro, você executa um bloco de código, se for falso, você executa outro bloco de código.

O operador lógico NÃO (Logical NOT)

JavaScript utiliza o ponto de exclamação ! para representar o operador lógico NÃO (negação ou valor contrário). O operador ! pode ser aplicado a um valor de qualquer tipo, não apenas a um valor booleano.

Quando você aplica o operador de negação ! a um valor booleano, o ! retorna true se o valor for false e retorna false se o valor for true, ou seja, vai retornar o valor contrário.

Por exemplo:

```
let  eligible = false;
let required = true;
console.log(!eligible) //true
console.log(!required); //false
```

Operador Lógico E (&&)

O JavaScript usa dois "e comerciais" && para representar o operador lógico E (Logical AND operator).

A tabela a seguir ilustra o resultado do operador && quando aplicado a dois valores booleanos.

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

O resultado do operador && será true apenas se ambos os valores forem true, caso contrário será sempre false. Por exemplo:

```
let  eligible = false;
let required = true;
console.log( eligible &&required) // false
```

Nesse exemplo a variável elegible é false, portanto o valor da expressão elegible && required é false.

Operador Lógico OU (||)

O JavaScript usa o pipe duplo || para representar o operador lógico OU (logical OR operator). Você pode aplicar o operador || para dois valores de qualquer tipo:

Exemplo:

```
let result = a || b
```

Se a pode ser convertido em true, então será retornado a, caso contrário, vai retornar b. Essa regra também se aplica a valores booleanos.

A tabela a seguir, ilustra o resultado do operador || com base no valor dos operandos (valores da esquerda e da direita):

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

O operador || retornará false se ambos os valores avaliados forem false. Caso qualquer um dos valores seja true, o operador || retorna true.

Por exemplo:

```
let  eligible = false;
let required = true;
console.log( eligible || required) // true
```

Nesse exemplo, a expressão (legible || required) retorna true porque um dos valores (nesse caso o elegible) é true.

Precedência lógica do operador

Quando você usa diversos operadores lógicos em uma expressão, o motor JavaScript avalia os operadores com base em uma ordem específica e essa ordem é chamada precedência do operador. Em outras palavras, a precedência do operador é a ordem em que um operador é executado.

A precedência de execução dos operadores lógicos seguem a ordem (de cima para baixo) da lista a seguir:

- ✔ Operador lógico NÃO !
- ✔ Operador lógico E &&
- ✔ Operador lógico OU ||

Resumo:

O operador lógico NÃO !, nega um valor booleano.

O operador lógico E &&, é aplicado a dois valores booleanos e retorna true se os dois valores forem true.

O operador lógico OU || é aplicado a dois valores booleanos e retorna true se pelo menos um dos valores for true.

A precedência lógica do operador da mais alta para a mais baixa é: !, && e ||.

Saiba mais:

https://youtu.be/sIXUNUqnoXI

https://youtu.be/7Isq6ekQT1k

https://www.youtube.com/watch?v=Vbabsye7mWo

https://youtu.be/hZG9ODUdxHo

https://www.w3schools.com/js/js_variables.asp

https://www.w3schools.com/js/js_datatypes.asp