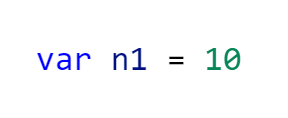
# Variáveis em JavaScript

Para declarar uma variável em JavaScript pode-se as palavras chaves **‘var’**, **‘let’** e **‘const’**. A sintaxe é a seguinte:

**var <identificador> = <valor da variável>;**

Por exemplo:



## Identificadores

O identificador de uma variável é o nome de referência da variável.

Os identificadores possuem algumas regras:

* Podem começar com uma letra, **‘$’**(cifrão) ou **‘\_’**(underline).
* Não podem conter espaços entre os caracteres.
* Não podem começar com números.
* Não podem ser palavras reservadas, como comandos em JS.
* Pode-se usar símbolos e acentos.
* Maiúsculas e minúsculas fazem diferença.

## *camelCase*

Assim é chamado a forma de escrever o nome das variáveis, onde o primeiro nome é escrito em minúsculo e os próximos em maiúsculo.

Por exemplo:

* javaScript
* firstName
* lastName

É comum também o uso de underlines (‘\_’) para a separação das palavras.

## Palavra-chave ‘let’

Declarar uma variável com a palavra-chave **‘let’** tornará possível um erro caso uma mesma variável seja declarada uma segunda vez. Ou seja, caso seja declarada uma variável existente, será executado um erro.

## Palavra-chave ‘const’

Variáveis que não devem ser reatribuídas devem ser declaradas com a palavra-chave **‘const’**, que as torna constante, não permitindo a alteração do valor.

Se caso houver a tentativa de reatribuir o valor da variável, ocorrerá um erro.

**NOTA**: É comum que o nome de variáveis imutáveis sejam MAIÚSCULAS.

# Tipos Primitivos

Em JavaScript, existem três tipos primitivos primordiais. São eles:

* Boolean (**true**, **false**).
* Number (**3**, **-2**, **0.2**, 5.9).
* String (**‘Olá, mundo’**, **“Olá, Mundo!”**, **`Olá, Mundo!`**)

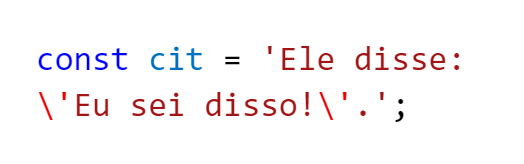
Com o comando **‘typeof’** é possível descobrir o tipo primitivo de uma variável, por exemplo. Irá retornar o tipo primitivo como valor.

## String

Em JavaScript, tudo entre aspas simples ou duplas são consideradas strings.

### Citações dentro da String

Para fazer uma citação dentro de uma string sem causar nenhum erro, use a barra invertida (**‘\’**) antes da citação da abertura de aspas da citação e antes do fechamento de aspas da mesma.



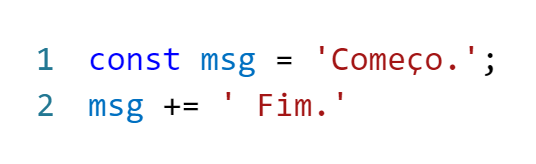
### Concatenação

O operador **‘+’** é usado também para concatenar strings.



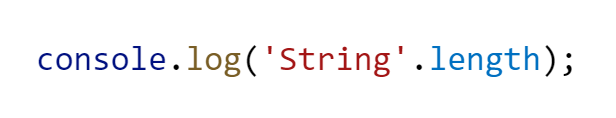
Perceba que na segunda string há um espaço. Esse espaço deve ser adicionado manualmente, ou o resultado será duas strings coladas uma à outra.

É possível usar também o sinal de mais junto ao de igual (**‘+=’**) para concatenar uma variável do tipo string com outra.



### Comprimento de uma String

Usar a propriedade **‘.length’** ao final da string retornará o comprimento do valor dessa string.

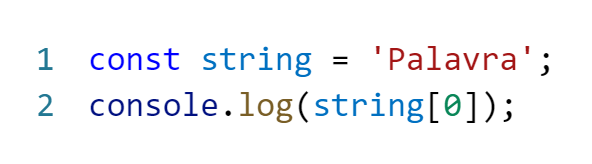


No caso acima, o resultado será 6.

### Fatiamento de String

Adicionando colchetes com algum número ao final de uma variável que tem uma string como tipo de valor, é possível especificar um caractere específico de uma string.

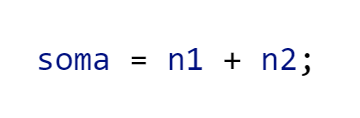
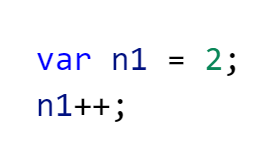
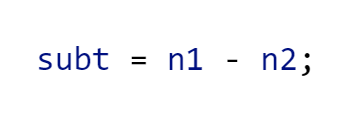
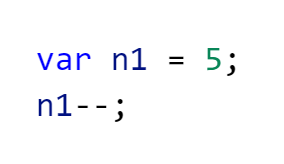
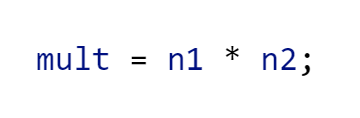
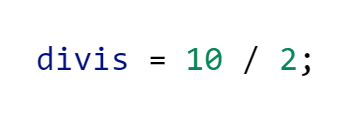
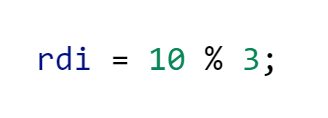
O primeiro caractere de uma string é zero, ou seja, o primeiro caractere de uma string seria: **‘string[0]’**.



## Number

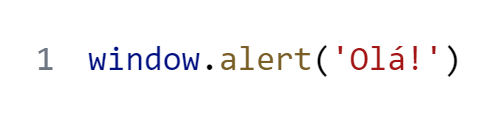
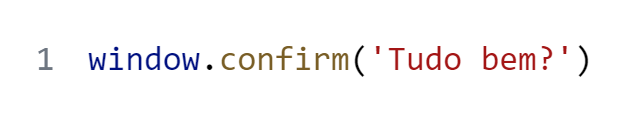
Os valores numéricos podem ser reais ou inteiros e com eles é possível fazer operações.

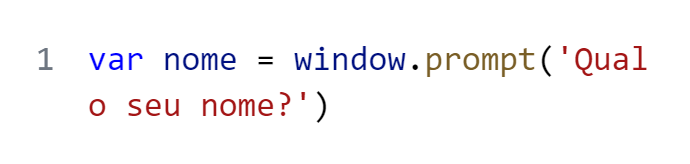
Os operadores são:

* **‘+’:** Adição.  
  
* **‘++’:** Soma mais um à variável.  
    
  O mesmo é equivalente a **‘n1 = n1 + 1’**. O resultado será 3.
* **‘-’:** Subtração.  
  
* **‘--’:** Subtrai um da variável.  
    
  O resultado será 4.
* **‘\*’:** Multiplicação.  
  
* **‘/’:** Divisão.  
  
* **‘%’:** Resto da divisão.  
  

# ‘window’

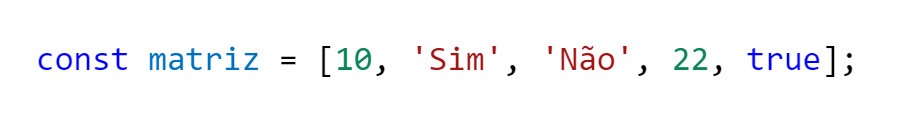
É um objeto em JavaScript que possui diversas funções. Algumas delas são:

* **‘alert()’:** Exibe uma caixa de texto.  
  
* **‘confirm()’:** Exibe uma caixa de texto com as opções “ok” e “cancelar”.  
  

**‘prompt()’:** Cria uma caixa de texto que possui uma caixa de entrada. Para usar o valor escrito pelo usuário deve ser usado em conjunto a uma variável.  


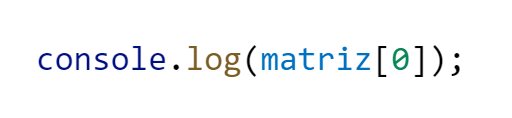
# Matrizes em JS (*Arrays*)

Permite armazenar vários valores em uma única variável. Todos os valores devem estar dentro de colchetes **‘[]’** e separados por vírgula.



Como pode ver, é possível armazenar diversos valores em uma matriz, ainda que diferentes uns dos outros.

Cada item na matriz é acessado por suas respectivas posições, considerando que a primeira posição é 0.

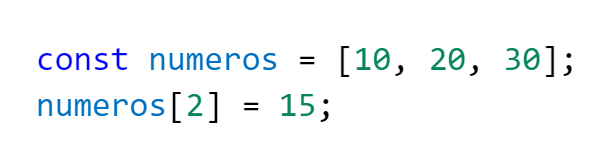


No exemplo acima, o console retornará o primeiro valor da matriz.

Caso pedido uma posição ou item inexistente na matriz, será retornado o valor **‘undefined’**.

## Mutação de matriz

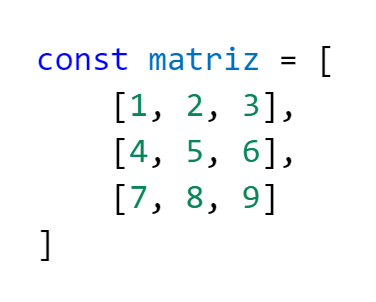
Mesmo que a matriz tenha sido declarada com ‘const’, ela é mutável. A variável em si é IMUTÁVEL, mas os valores podem ser modificados.



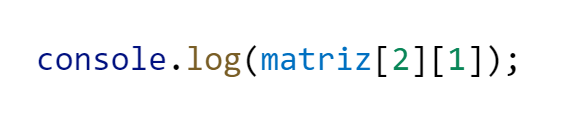
No código acima, o valor do terceiro item (posição 2) foi alterado de **‘30’** para **‘15’**.

## Matriz multidimensional

Assim se dá o nome quando uma matriz é composta de outras matrizes.



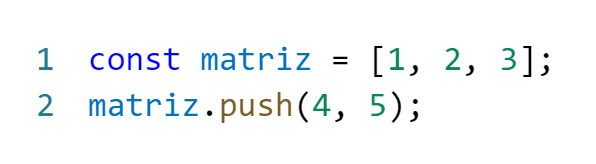
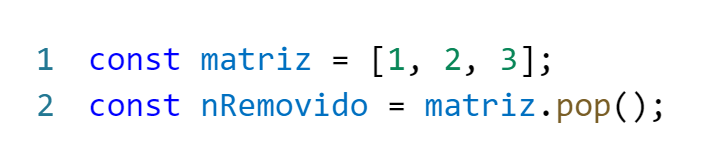
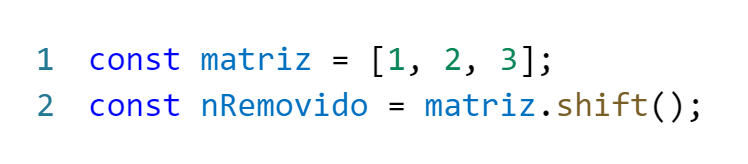
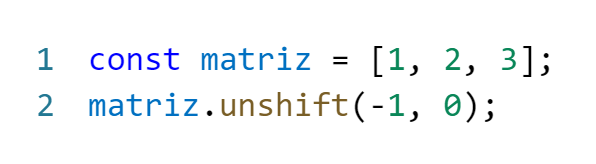
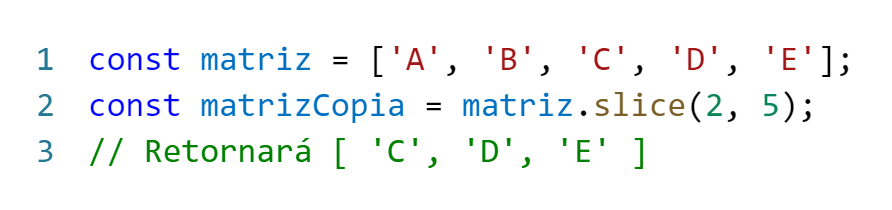
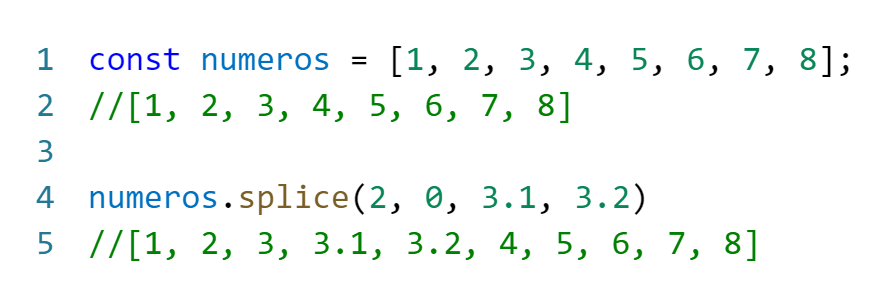
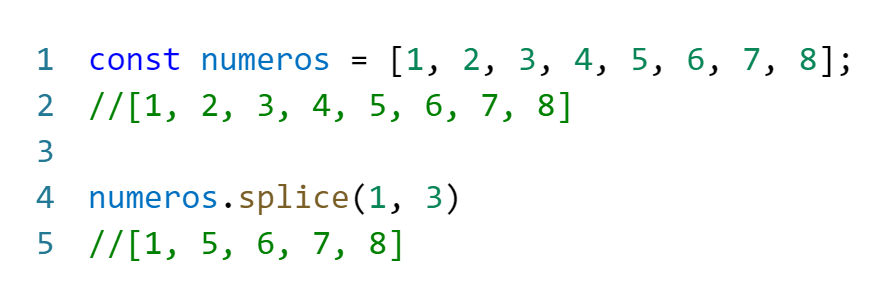
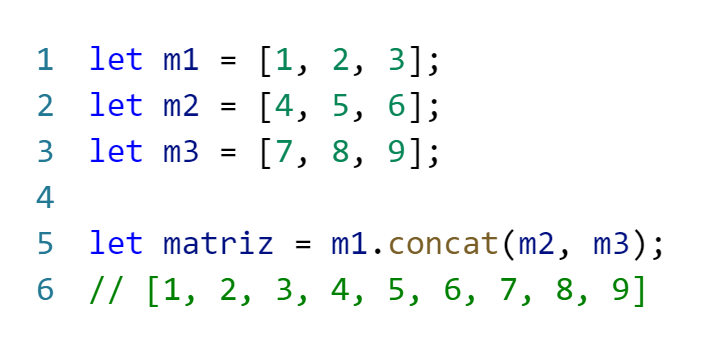
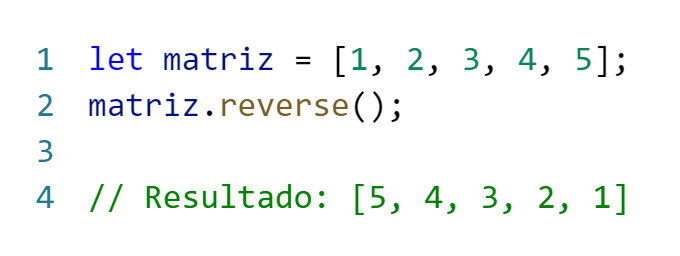
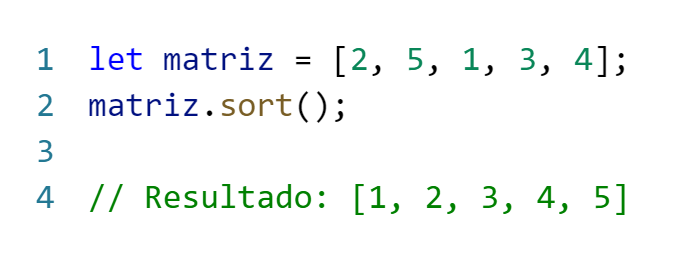
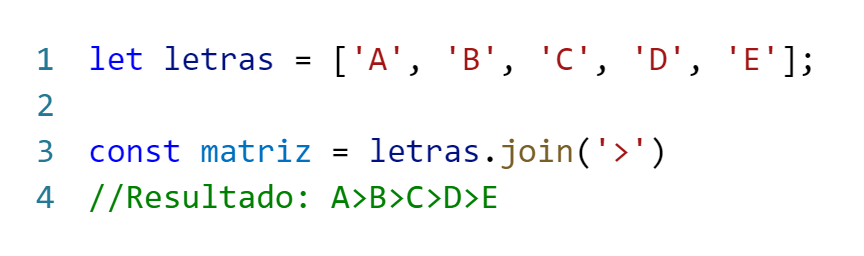
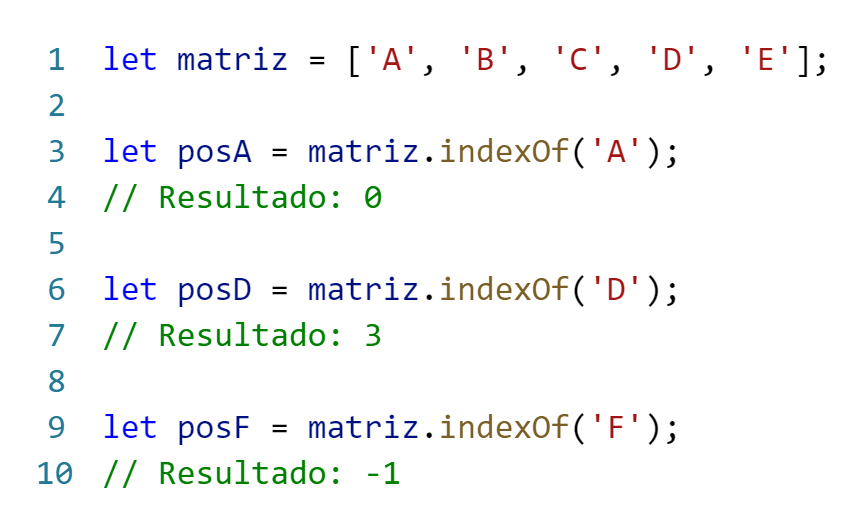
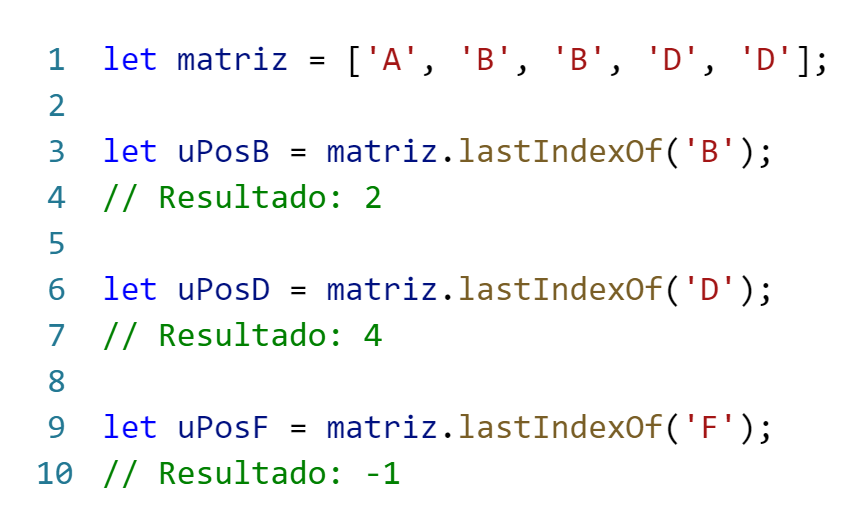
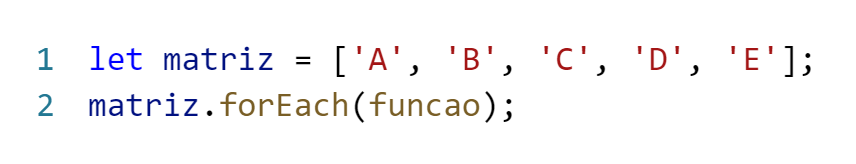
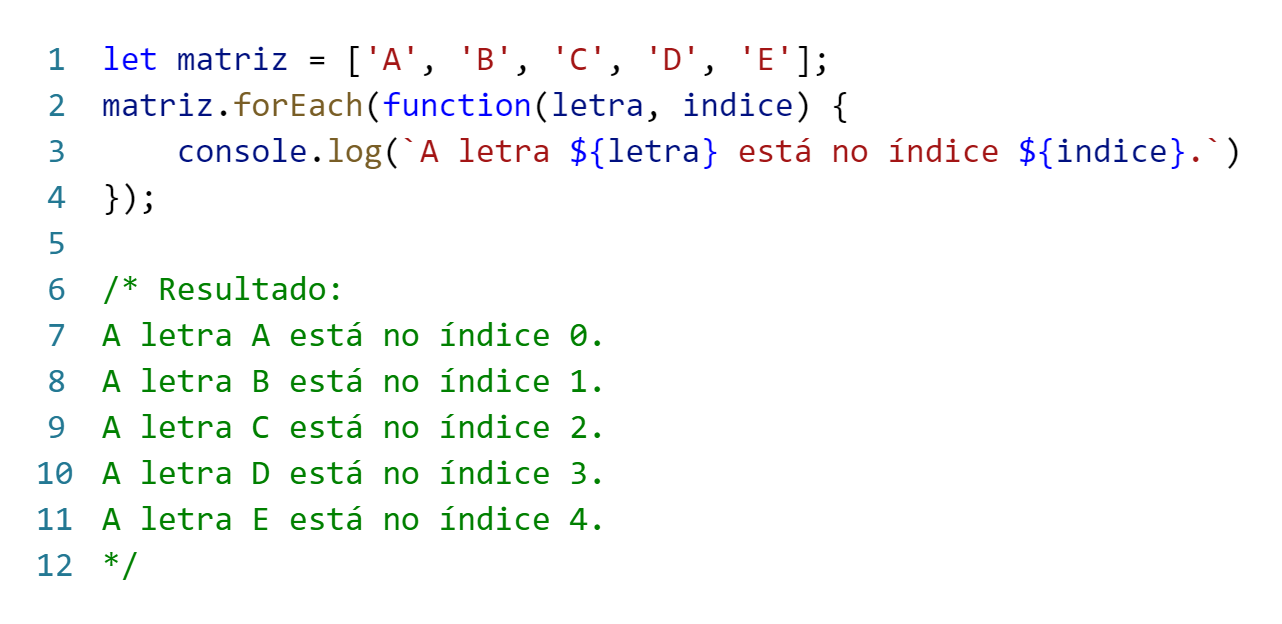
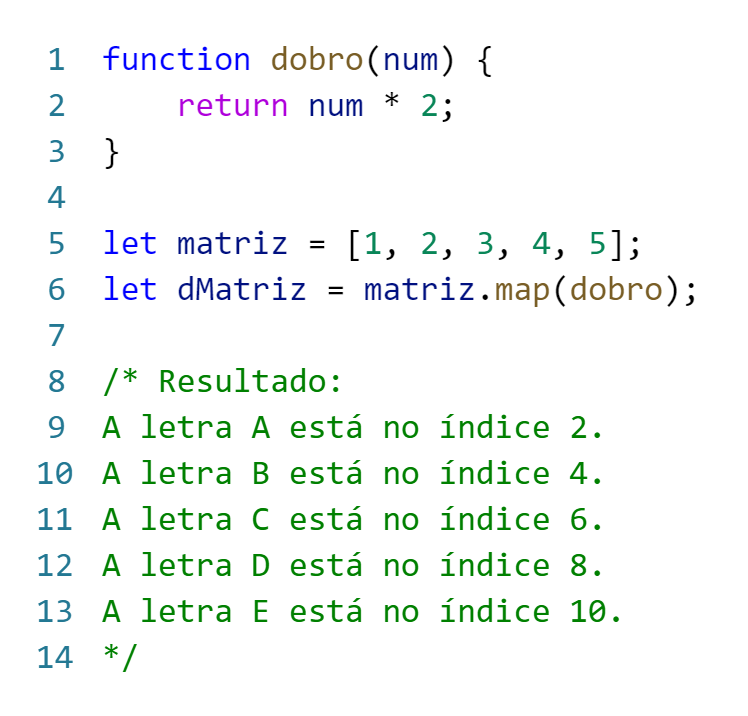
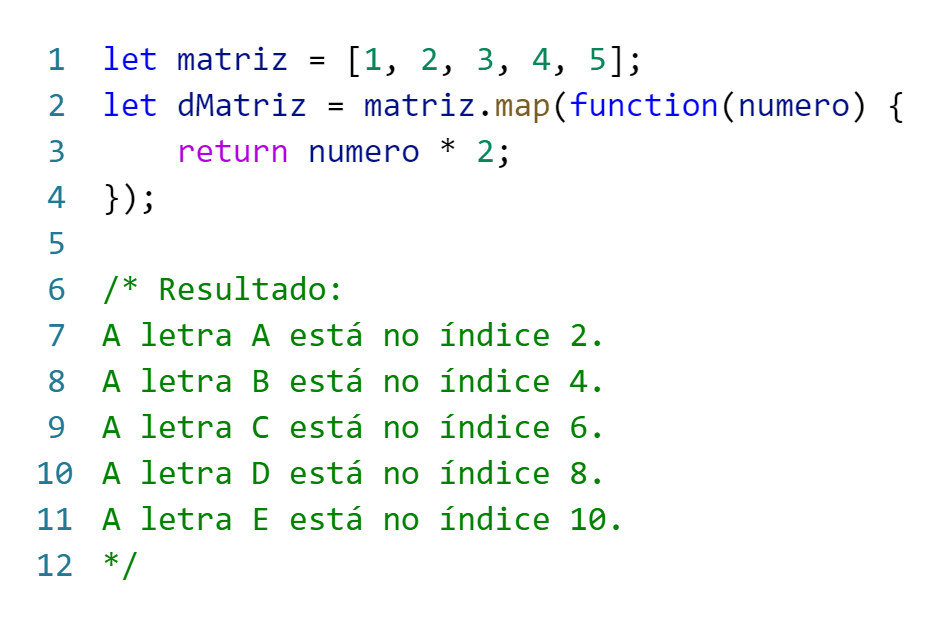
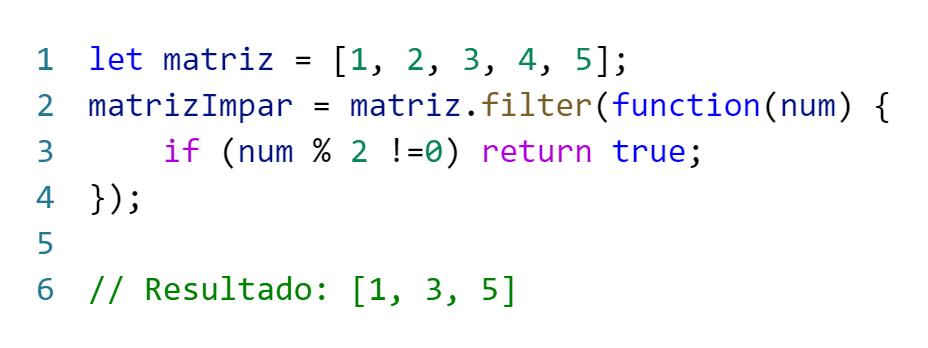
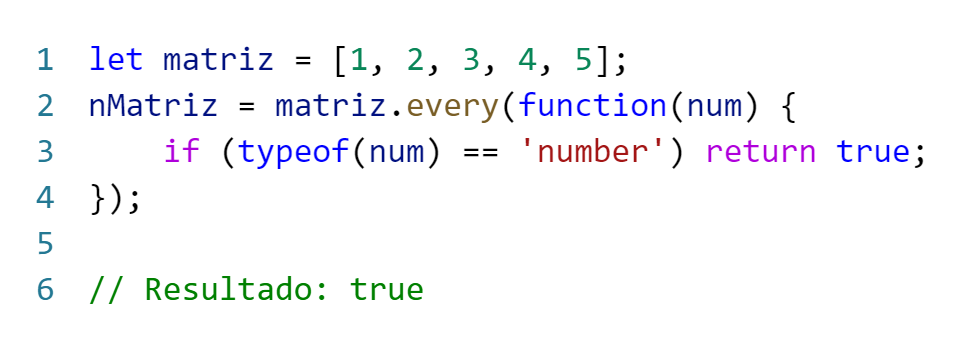
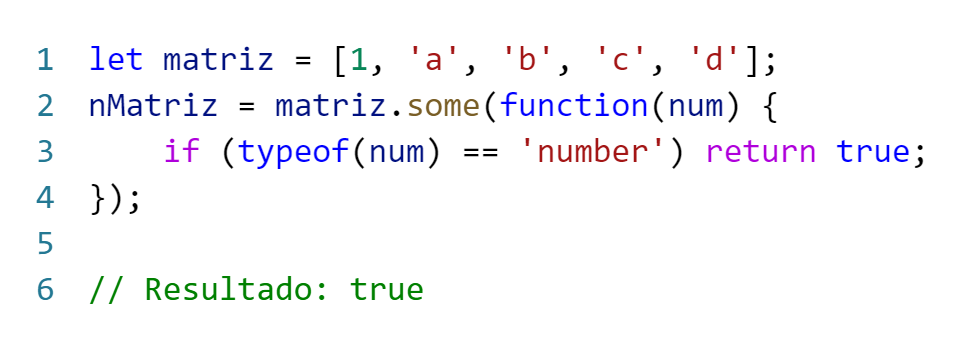
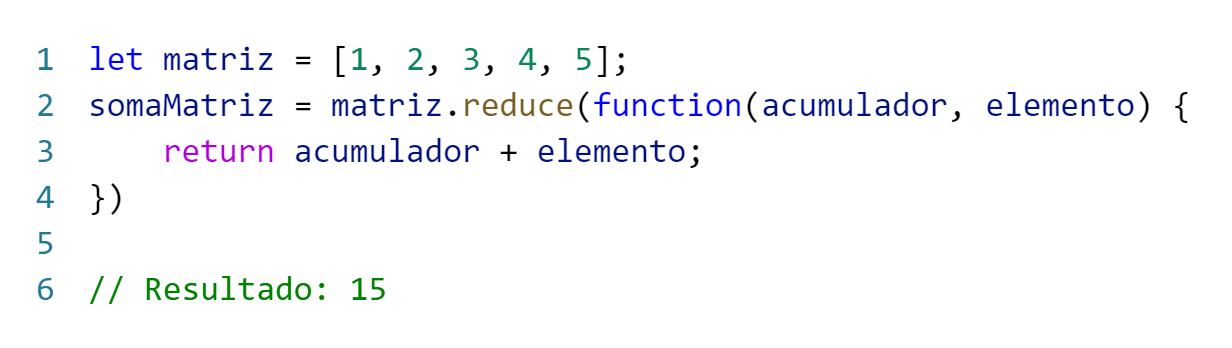
O acesso a um item dentro de uma matriz interna é feito pela posição, primeiro especificando a posição da matriz e depois a posição do item dentro dessa matriz.



Considerando a matriz multidimensional anterior, o código acima retornará no console o valor **‘8’**.

## Métodos de Matriz

Existem alguns métodos para matrizes que ajudam a adicionar, remover e manipular elementos dentro da matriz.

* **‘push()’:** Adiciona um ou mais elementos ao final da matriz original.  
  
* **‘pop()’:** Remove o último elemento da matriz e retorna o elemento removido.  
  
* **‘shift()’:** Remove o primeiro elemento da matriz e retorna o valor removido.  
  
* **‘unshift’:** Adiciona um ou mais elementos ao início da matriz.  
  
* **‘slice()’:** Cria uma cópia da matriz, começando e terminando pelas posições especificadas.  
    
  O primeiro valor é o **início** e o segundo é o **fim**. Caso não sejam especificados nenhum valor, será copiada a matriz inteira.
* **‘splice()’:** Adiciona, remove ou substitui elementos em posições específicas na matriz.  
    
  O primeiro valor é a posição em que começará as alterações.  
  O segundo valor é a quantidade de valores que serão excluídos.  
  Do terceiro valor em diante são os valores a serem adicionados.  
    
  Neste exemplo, os três primeiros valores a partir da posição 1 serão excluídos.  
  É importante ressaltar que tudo isso modifica a matriz original.
* **‘concat()’:** Combina duas ou mais matrizes, concatenando-as.  
  
* **‘reverse()’:** Inverte a posição dos elementos na matriz.  
    
  Isso altera a matriz original.
* **‘sort()’:** Organiza a matriz em ordem alfabética e numérica.  
  
* **‘join()’:** Cria uma string a partir dos elementos que compõem a matriz. Os elementos são separados por um caractere especificado.  
  
* **‘indexOf’:** Encontra a posição do primeiro elemento especificado, ou **‘-1’** caso elemento não exista dentro da matriz.  
  
* **‘lastIndexOf’:** Encontra a posição do último elemento especificado, ou **‘-1’** caso elemento não exista dentro da matriz.  
  
* **‘forEach’:** Executa uma função para cada elemento dentro da matriz.  
    
  Nesse caso, a função ‘funcao’ será executada cinco vezes, uma para cada elemento dentro da matriz.  
    
  Nesse caso, é criado um bloco com uma função.
* **‘map’:** Permite criar uma nova matriz com base em uma matriz existente por meio de uma função.  
    
  No exemplo acima, cada valor da matriz é multiplicado por 2, e ao final, a matriz **‘dMatriz’** possui todos os valores da primeira matriz, só que dobrados.  
    
  Essa é uma versão em que a função a ser executada foi criada junto à declaração. Ela funciona da mesma forma.
* **‘filter()’:** Retorna uma nova matriz apenas com os elementos verdadeiros em relação ao callback executado.  
  
* **‘every()’:** Retorna **‘true’** se todos os elementos forem verdadeiros de acordo com o callback.  
  
* **‘some()’:** Retorna **‘true’** se pelo menos um elemento for verdadeiro de acordo com o callback.  
  
* **‘reduce()’:** Reduz uma matriz inteira para um valor único de acordo com o callback.  
  
* **‘reduceRight()’:** É como o ‘reduce()’, porém atuando da direita para à esquerda.