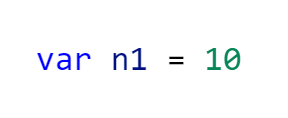
# Variáveis em JavaScript

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 alguns tipos primitivos. São eles:

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

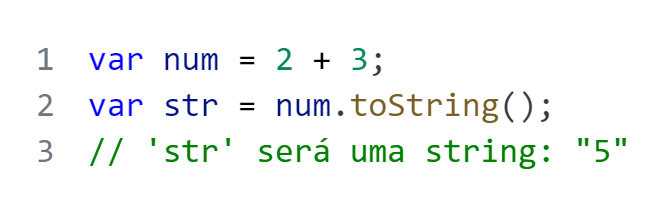
Com o comando **‘typeof’** é possível descobrir o tipo primitivo de uma variável, por exemplo. Irá retornar o tipo primitivo como valor.

## String

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

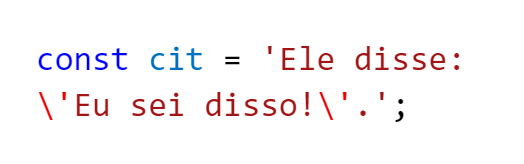
### Conversão para String

O método **‘toString()’** faz a conversão de uma expressão para string.



### 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.



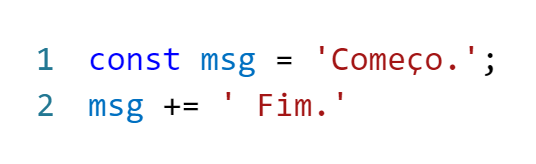
### Concatenação

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.

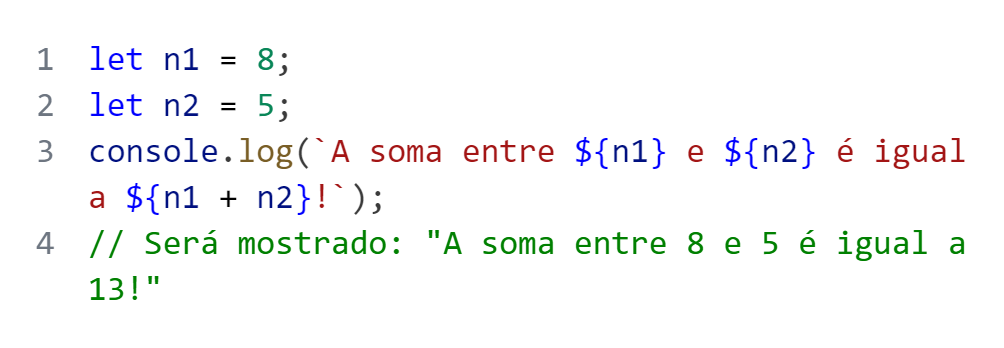
É possível usar também o sinal de mais junto ao de igual (**‘+=’**) para concatenar uma variável do tipo string com outra.



### Variáveis dentro da String

Usar crases (**‘``’**) ao invés de aspas simples ou duplas permite referenciar variáveis dentro da string.

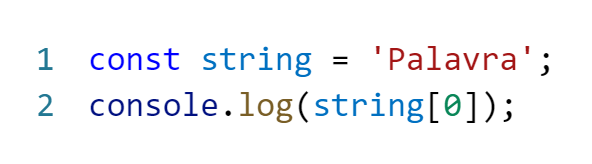
A variável (ou código) deve estar entre chaves, e antes das chaves deve haver um cifrão (**‘${}’**).



### 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.

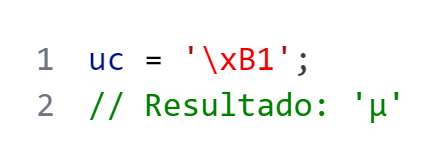
O primeiro caractere de uma string é zero, ou seja, o primeiro caractere de uma string seria: **‘string[0]’**.



### Caracteres especiais

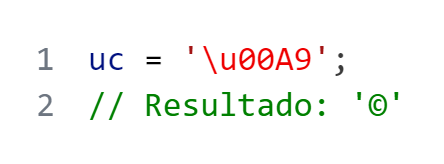
#### Sequência de escape hexadecimal

Os caracteres após **‘\x’** são interpretados como hexadecimal.

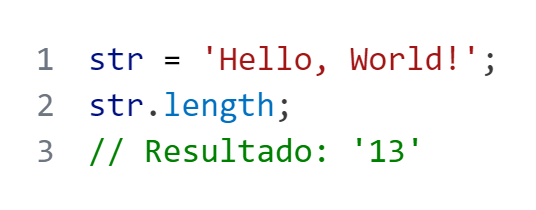
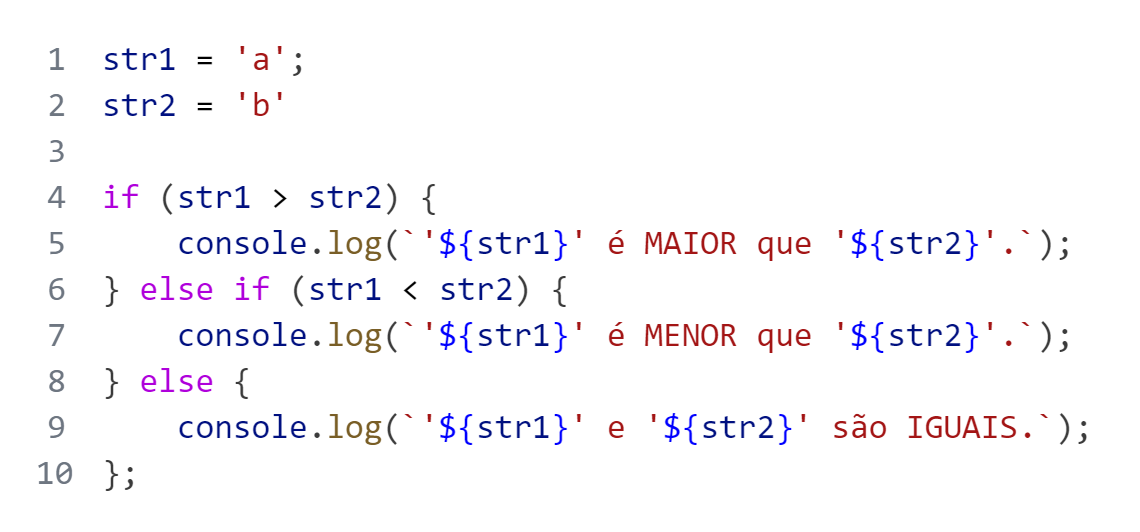


#### Sequência de escape *unicode*

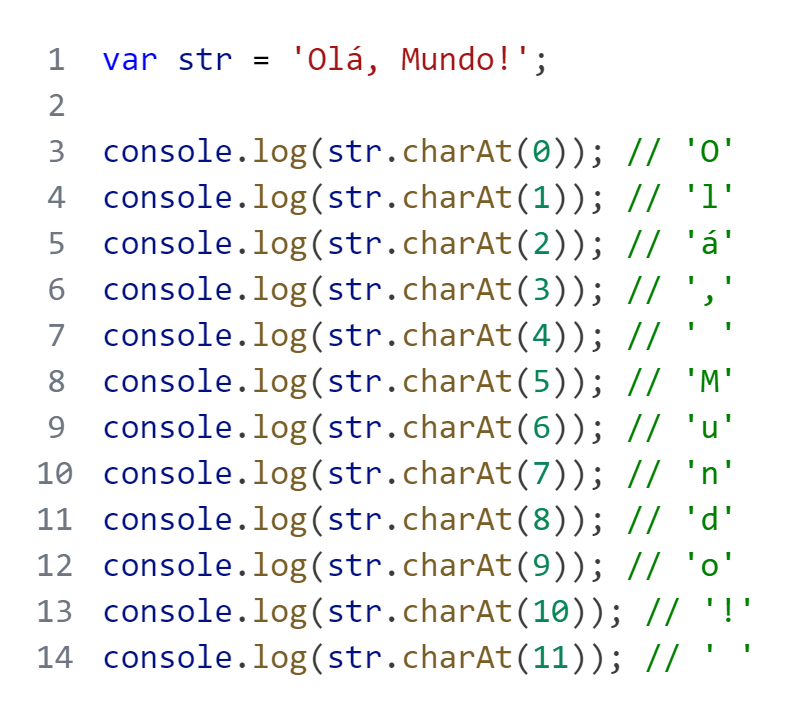
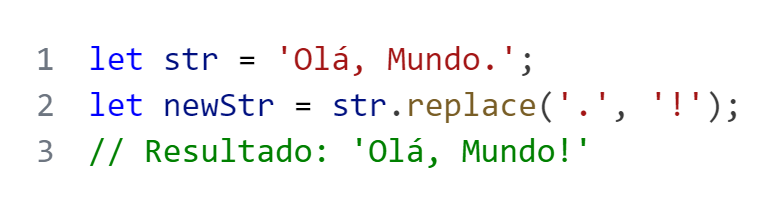
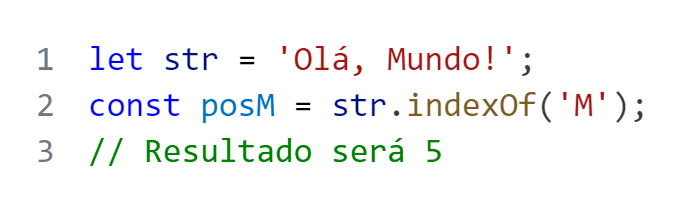
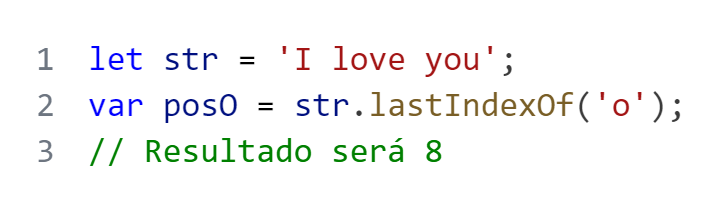
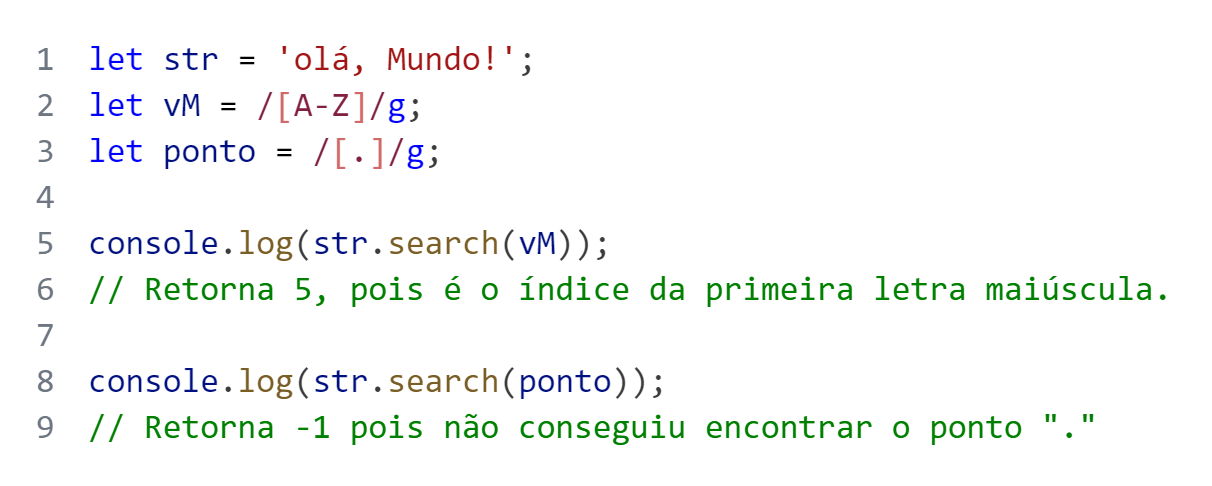
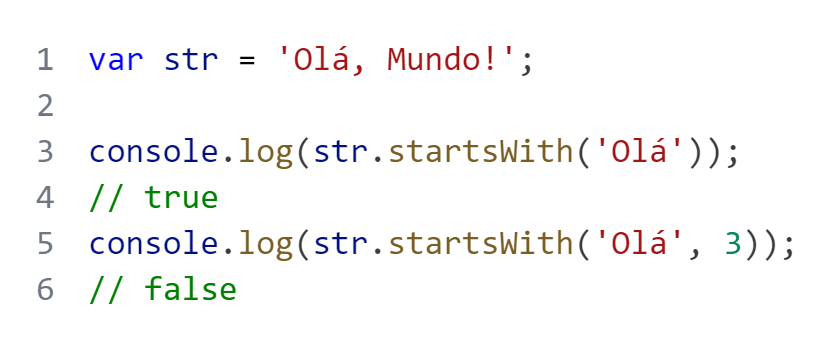
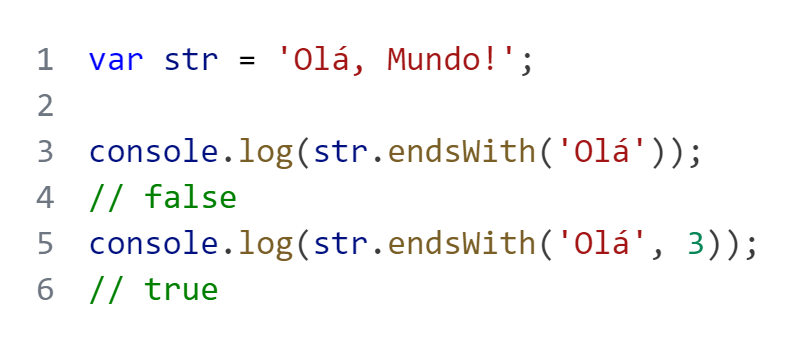
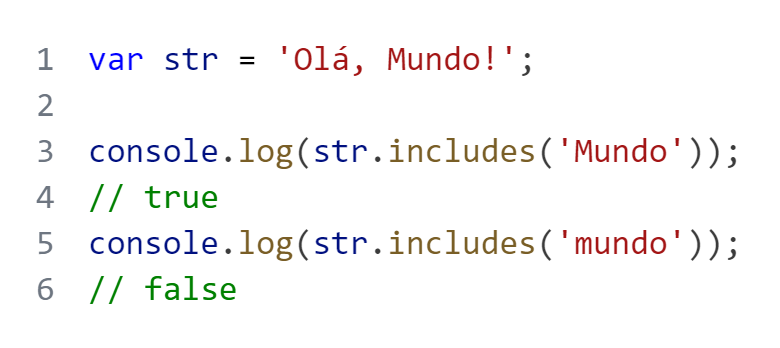
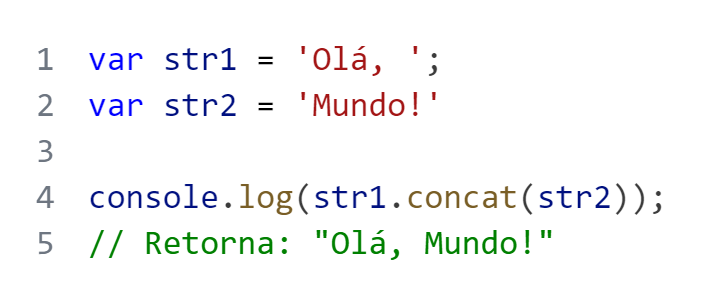
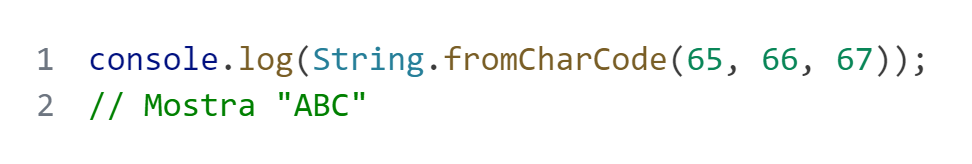
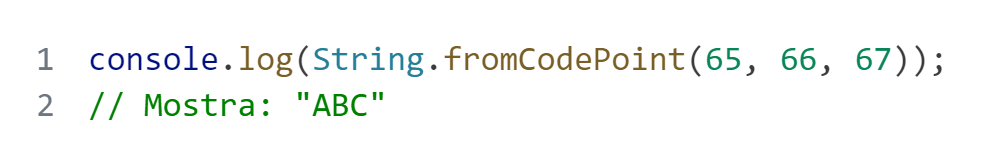
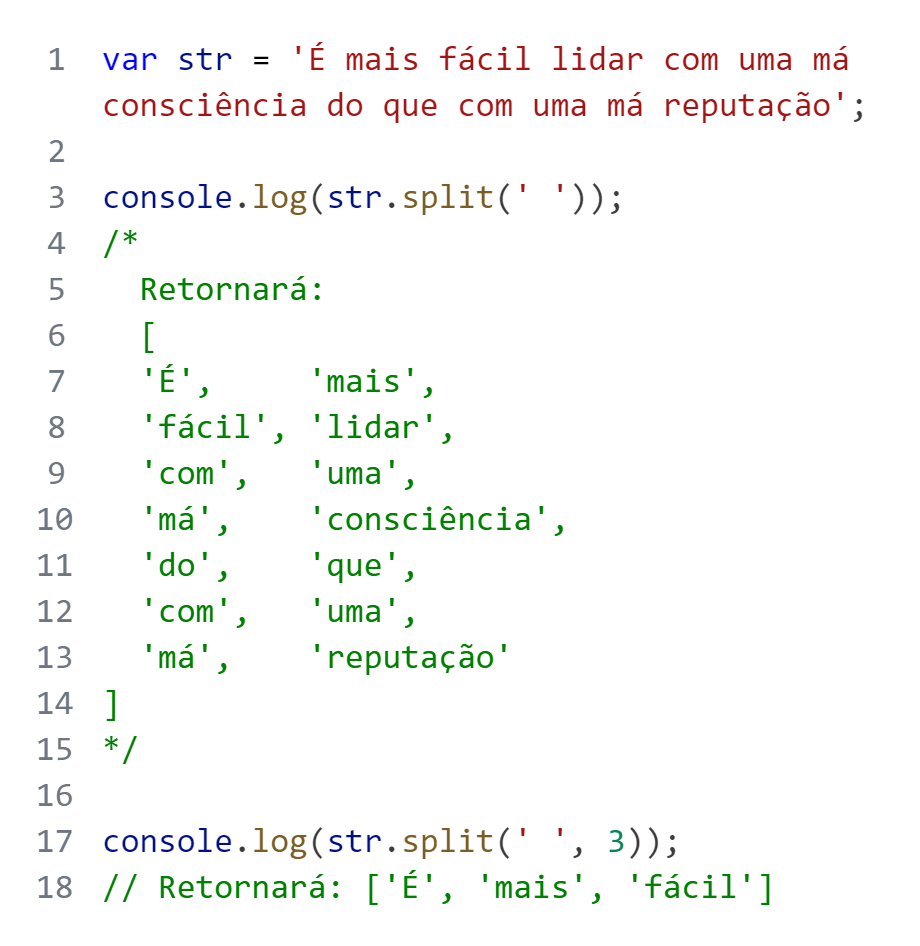
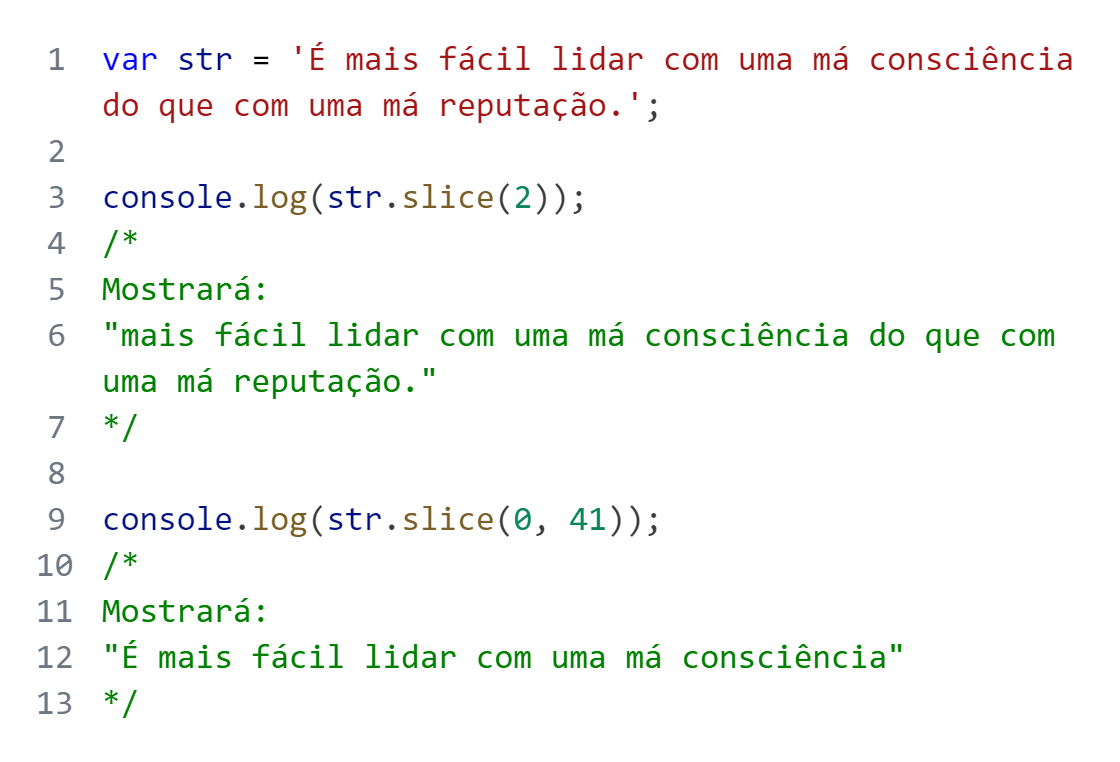
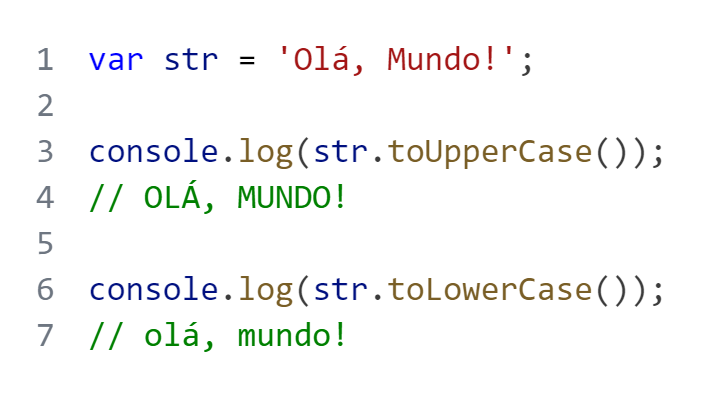
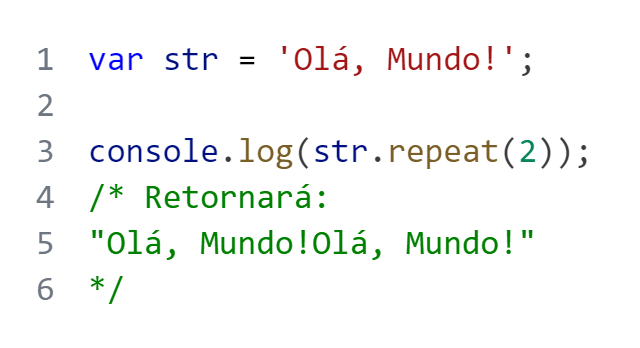
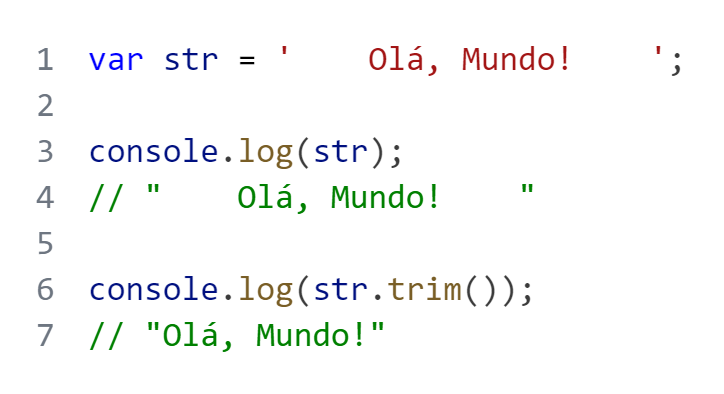
Os quatro caracteres após ‘\u’ são interpretados como *unicode*.



### Propriedades ‘String’

* **‘length’:** Retorna o comprimento da String.  
  
* **‘>’, ‘<’:** Compara strings.  
  

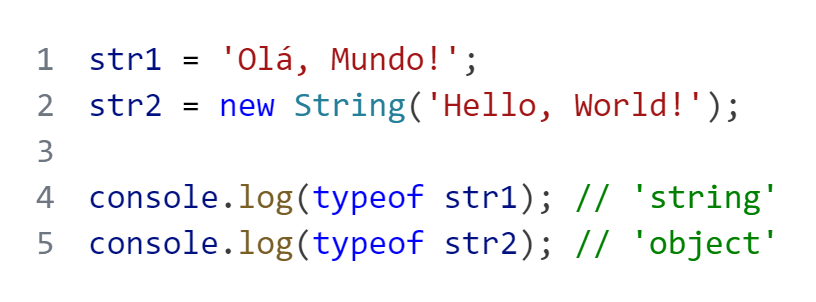
### Métodos ‘String’

* **‘charAt()’:** Retorna o caractere da posição especificada.  
    
  A funcionalidade é basicamente a mesma que colchetes para especificar o índice do caractere.
* **‘replace()’:** Substitui o primeiro argumento especificado pelo segundo dentro da string.  
  
* **‘indexOf()’:** Retorna a posição do primeiro caractere especificado.  
    
  Caso o caractere não exista dentro da string, retornará **‘-1’**.
* **‘lastIndexOf()’:** Retorna a posição do primeiro caractere especificado da direita para a esquerda.  
    
  Caso o caractere não exista dentro da string, retornará **‘-1’**.
* **‘search()’:** É usado para pesquisas dentro de uma string, parecido com ‘indexOf()’, porém permite o uso de **regexp**.  
  
* **‘startsWith()’:** Retorna verdadeiro caso a string especificada esteja no começo dentro da string, ou falso caso contrário.  
    
  São dois parâmetros:
  1. **‘searchString’:** A string a ser procurada.
  2. **‘posision’:** A posição a qual na qual se inicia a busca pela string especificada no parâmetro anterior. Este parâmetro é opcional.
* **‘endsWith()’:** Retorna verdadeiro caso a string termine com os caracteres determinados, ou falso, em caso contrário.  
    
  Seus parâmetros são:
  1. **‘searchString’:** A string a ser procurada.
  2. **‘endPosition’:** A posição final na qual se espera que o parâmetro anterior seja encontrado.
* **‘includes()’:** Retorna verdadeiro se a string especificada exista em qualquer lugar da string, ou falso caso não exista.  
    
  Seus parâmetros são:
  1. **‘searchString’:** A string a ser procurada.
  2. **‘position’:** A posição em que a string deve começar a ser procurada.
* **‘concat()’:** Combina o texto de duas ou mais strings e retorna uma nova string.  
  
* **‘fromCharCode()’:** Retorna uma string criada a partir de uma sequência específica de valores Unicode.  
  
* **‘fromCodePoint()’:** Retorna uma string criada a partir de uma sequência específica de pontos de código.  
  
* **‘split()’:** Cria uma Array com a string a partir de uma separador.  
    
  Seus parâmetros são:
  1. **‘separator’:** Especifica o separador das palavras da string, podendo ser inclusive mais de um caractere. Caso não seja especificado nenhum separador, se contida a string inteira em um único espaço na Array.
  2. **‘limit’:** Especifica o limite de palavras que serão contidas dentro da Array. Este parâmetro é opcional.
* **‘slice()’:** Retorna uma nova string a partir da string original.  
    
  Os parâmetros são o **índice de início(‘startIndex’)** e o **índice de fim(‘endIndex’)**, respectivamente.
* **‘toUpperCase()’, ‘toLowerCase()’:** Retornam uma string em maiúsculo e minúsculo, respectivamente.  
  
* **‘repeat()’:** Retorna uma nova string que repete o texto da string original determinadas vezes.  
  
* **‘trim()’:** Remove os espaços em branco no início e no fim da string.  
  

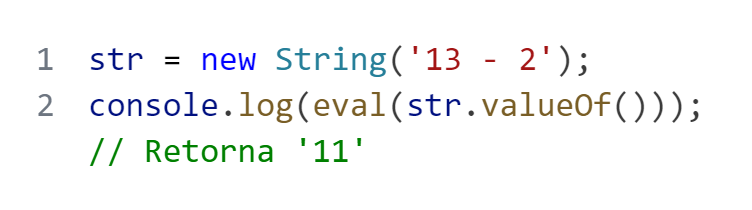
### String Primitiva x Objeto String

Objetos String são tratados como todos os outros objetos.

As Strings Primitivas são Strings “comuns”.



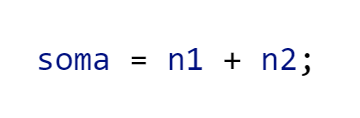
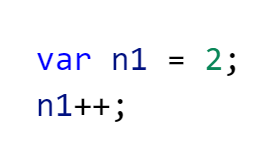
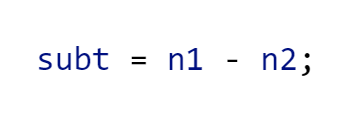
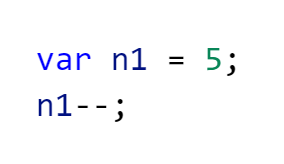
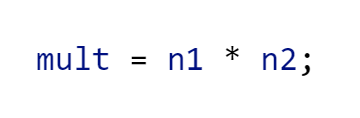
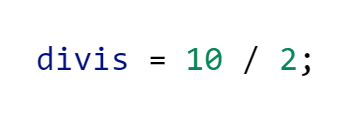
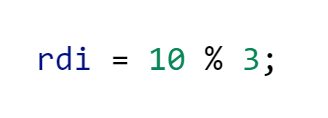
Para obter o valor de uma string primitiva sobre um objeto string, pode-se usar:



## Number

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

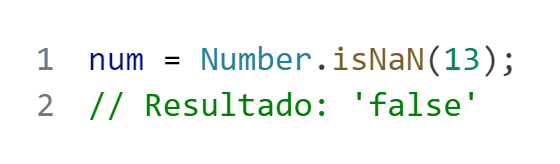
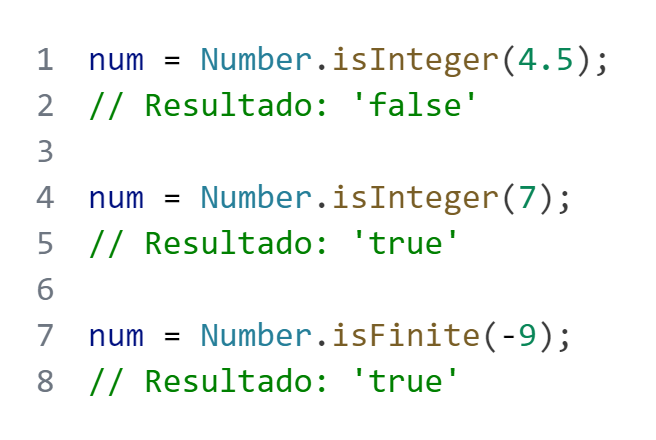
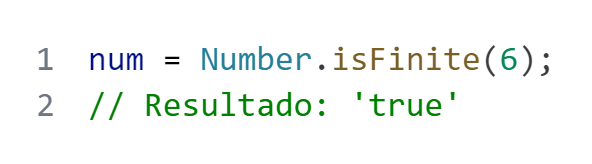
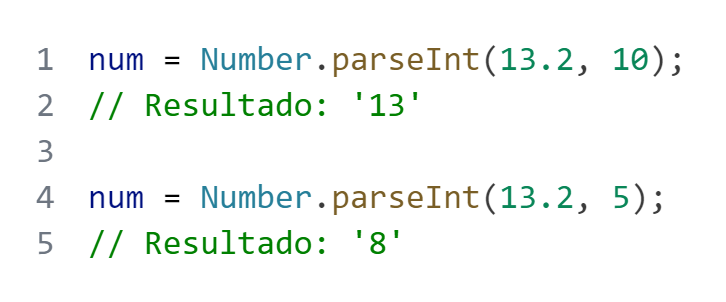
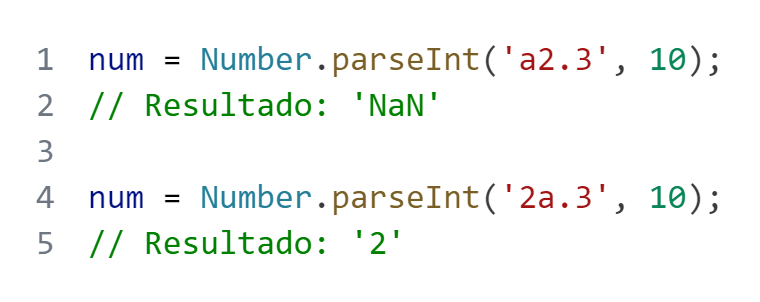
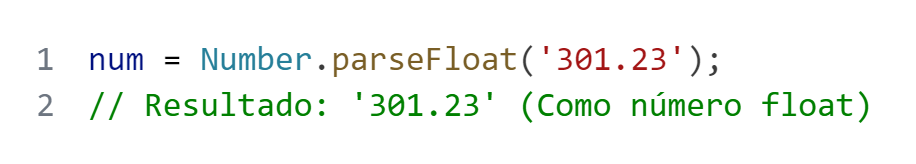
Os operadores são:

* **‘+’:** 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.  
  

### Propriedades ‘Number’

* **‘Number.EPSILON’:** Representa a menor diferença entre dois números de ponto flutuante padrão. Podendo ser usada para evitar problemas relacionados a erros de arredondamento em comparações numéricas.
* **‘Number.MAX\_SAFE\_INTEGER’:** É o inteiro máximo seguro em JavaScript (253-1).
* **‘Number.MIN\_SAFE\_INTEGER’:** É o inteiro mínimo seguro em JavaScript (-(253-1).
* **‘Number.MAX\_VALUE’:** O maior número representável positivo.
* **‘Number.MIN\_VALUE’:** O menor número representável positivo.
* **‘Number.NEGATIVE\_INFINITY’:** Representa o valor especial infinito negativo.
* **‘Number.POSITIVE\_INFINITY’:** Representa o valor especial infinito positivo.
* **‘Number.NaN’:** Valor especial que não é um número.

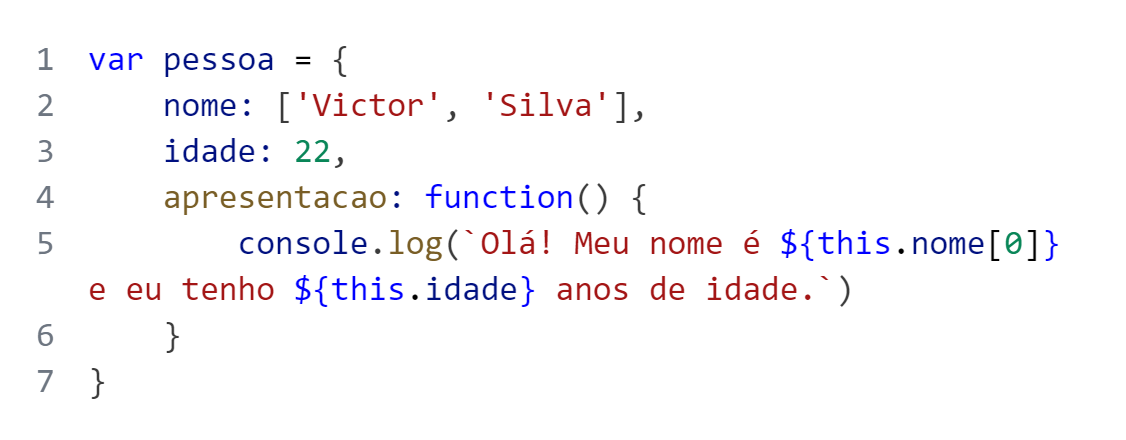
### Métodos ‘Number’

* **‘Number.isNaN()’:** Determina se o valor passado é NaN.  
  
* **‘Number.isInteger()’:** Determina se o tipo de valor é inteiro.  
  
* **‘Number.isFinite()’:** Determina se o tipo de valor é finito.  
  
* **‘Number.isSafeInteger()’:** Determina se o tipo de valor é um inteiro seguro, ou seja, está entre -(253-1) e 253-1.
* **‘Number.parseInt()’:** Converte um argumento em inteiro. Possui dois parâmetros: o primeiro é o valor a ser transformado em inteiro e o segundo é a base.  
    
  Caso o segundo parâmetro não seja especificado, o valor será o número inteiro na base 10.  
  Será retornado **‘NaN’** caso o primeiro caractere da string não possa ser convertido.  
  
* **‘Number.parseFloat()’:** Converte um argumento em real (Float).  
    
  Assim como ‘.parseInt()’, retornará **‘NaN’** caso o primeiro caractere da string não possa ser convertido.

## Object

Um objeto é uma coleção de dados e/ou funcionalidades relacionadas.

Variáveis e funções são exemplos de objetos, que são chamadas de **métodos** quando estão dentro de objetos.



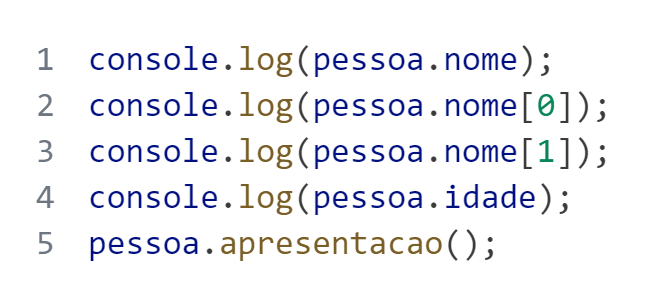
#### Membros e seus Valores

São chamados de **membros** os nomes que receberão os valores. No exemplo acima, os membros são **‘nome’**, **‘idade’** e **‘apresentacao’**.

Os **valores dos membros** do exemplo estão logo após os dois pontos após o membro.

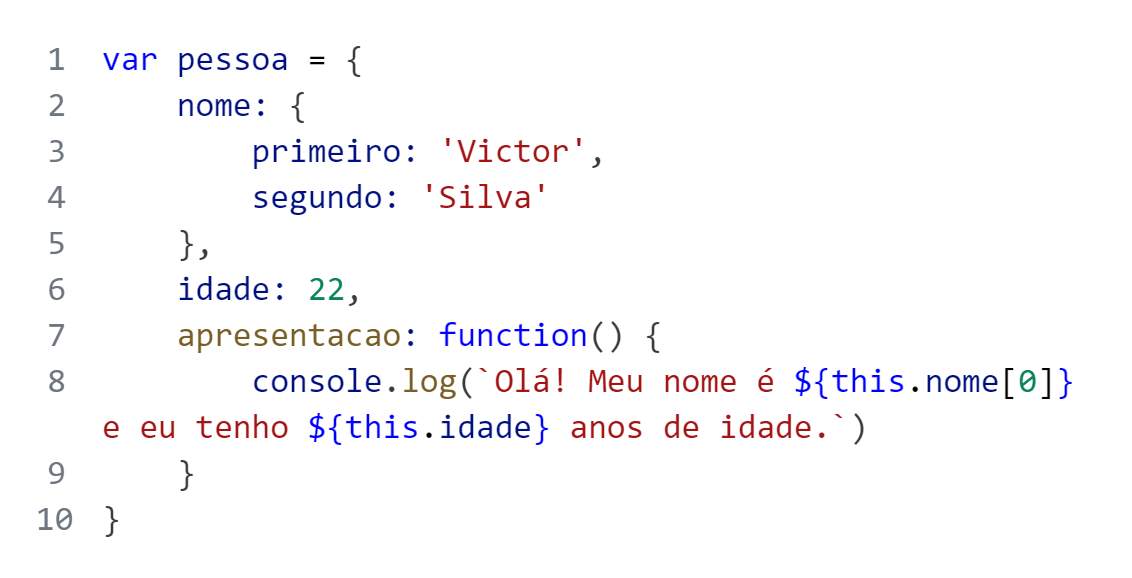
#### Uso de Dados de um Objeto

Para fazer o uso dos dados é usada a notação de ponto.

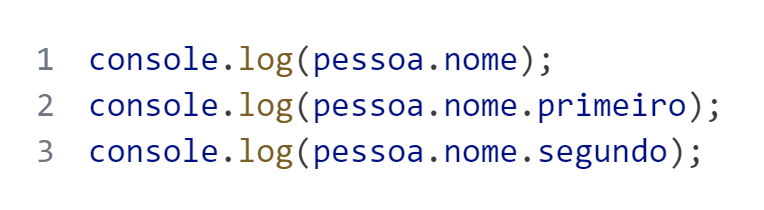


#### Objeto dentro de Objeto

É possível criar outros objetos dentro de um objeto:

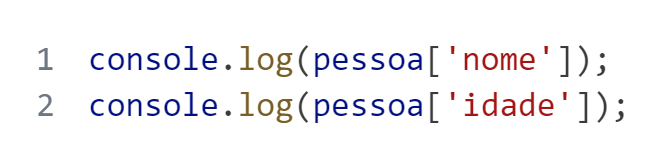


Para mostrar no console, por exemplo:



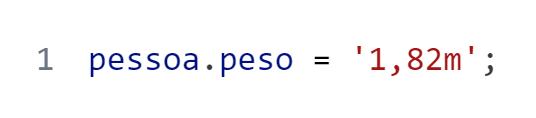
#### Referência por Nome do Membro

É possível referenciar o membro por seu nome também. Para isso é necessário o uso da notação de colchetes, como nas Arrays.

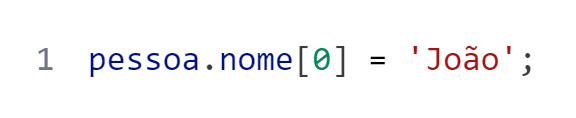


#### Criar e Modificar Membros.

É possível criar novos membros para o objeto, dessa forma:

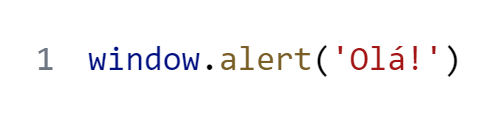
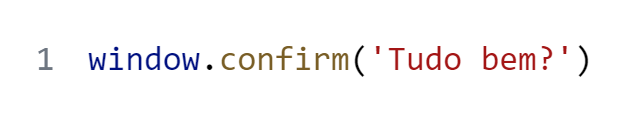


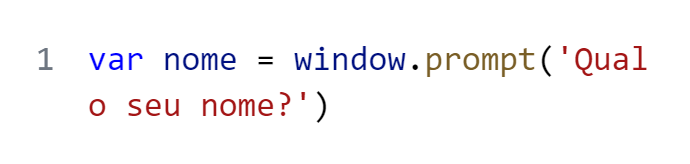
Dessa mesma forma é possível alterar o valor de um membro já existente.



# ‘window’

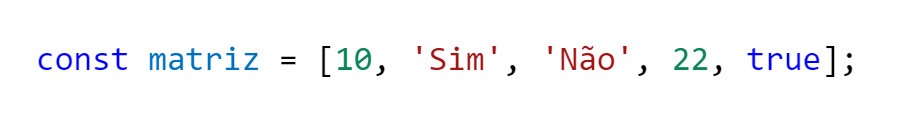
É um objeto em JavaScript que possui diversas funções. Algumas delas são:

* **‘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.  


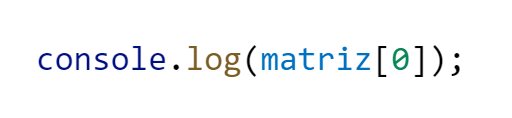
# *Arrays*

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.

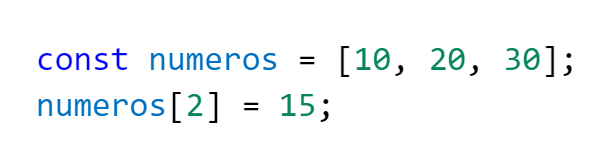


No exemplo acima, o console retornará o primeiro valor da matriz.

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

## Mutação de Array

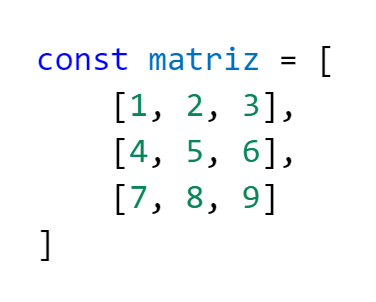
Mesmo que a matriz tenha sido declarada com ‘const’, ela é mutável. A variável em si é IMUTÁVEL, mas os valores podem ser modificados.



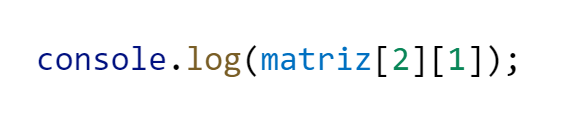
No código acima, o valor do terceiro item (posição 2) foi alterado de **‘30’** para **‘15’**.

## Array multidimensional

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



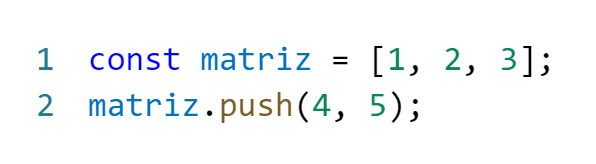
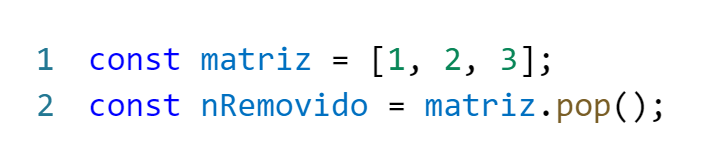
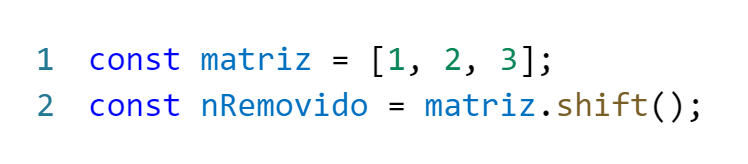
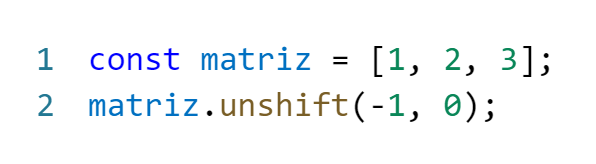
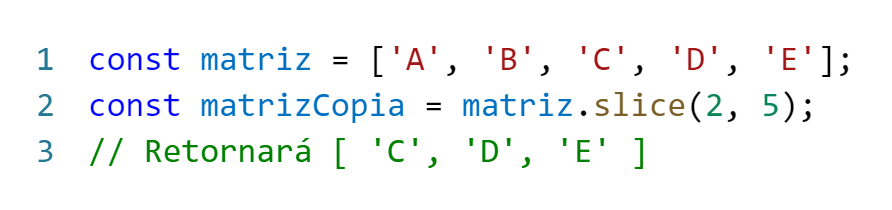
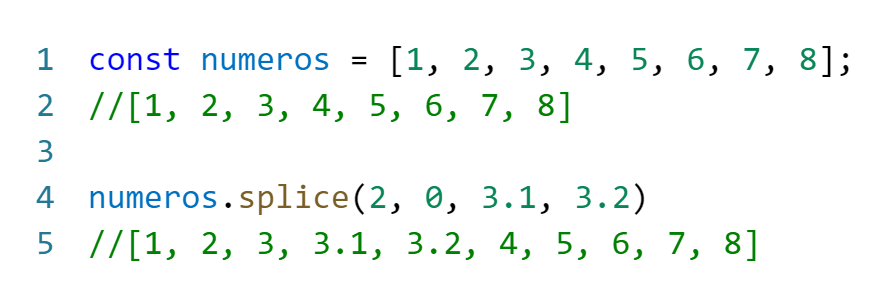
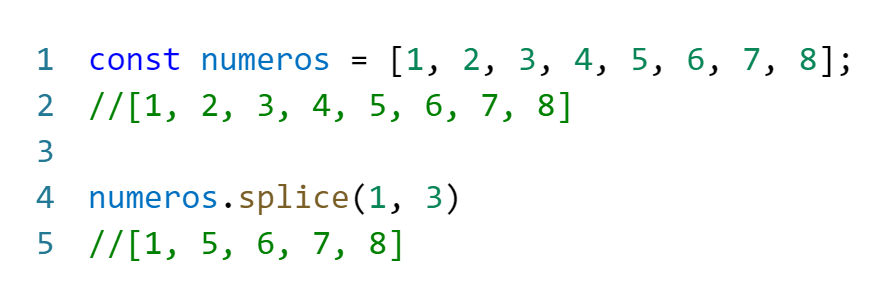
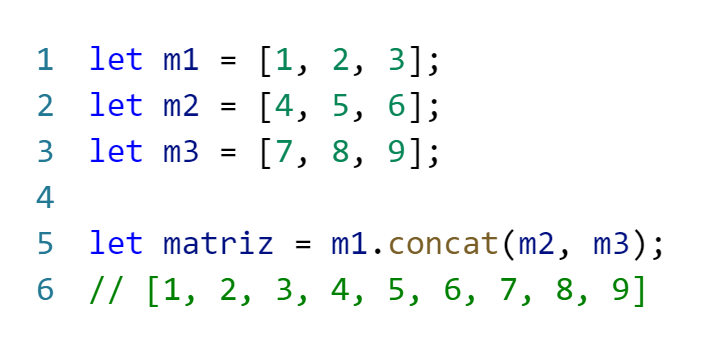
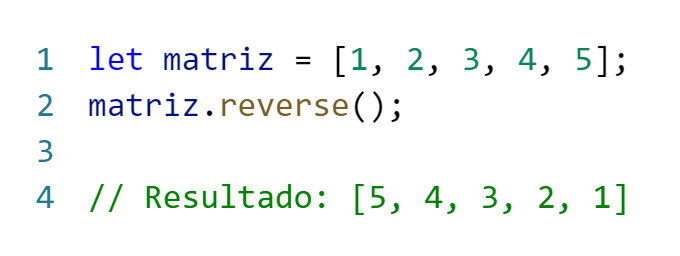
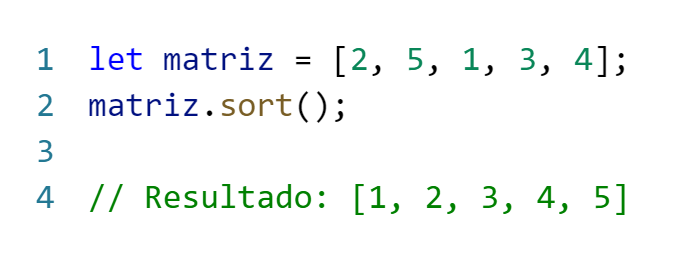
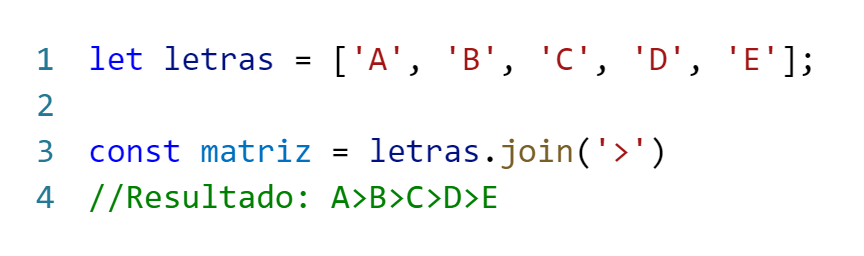
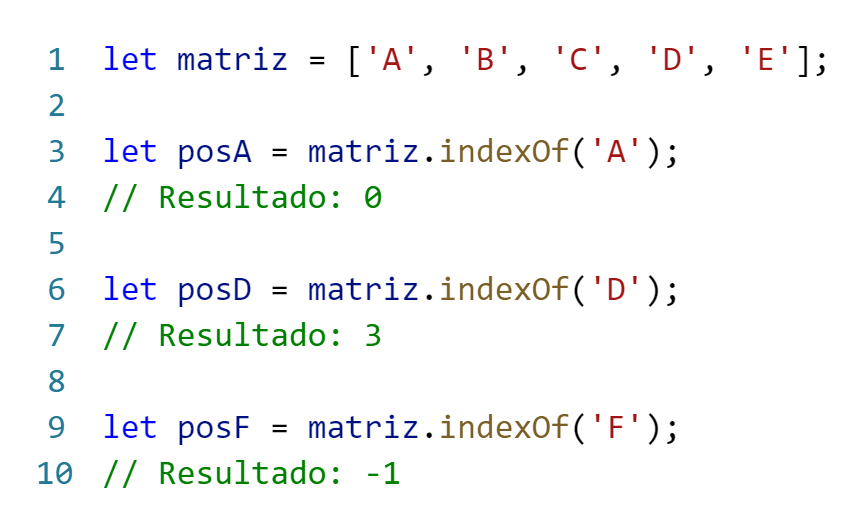
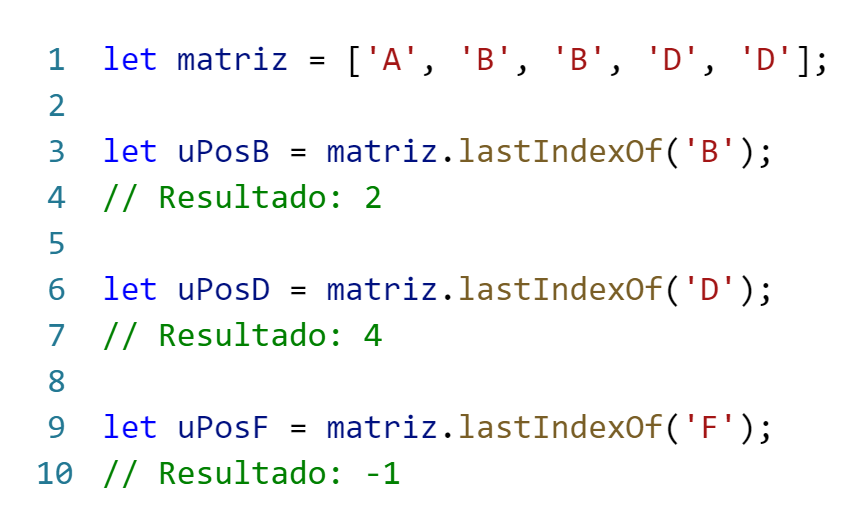
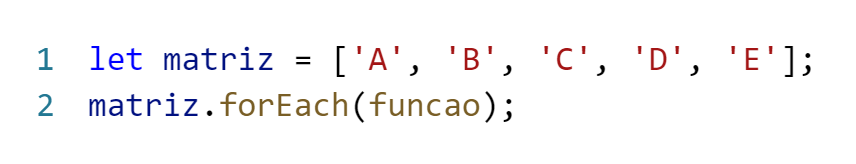
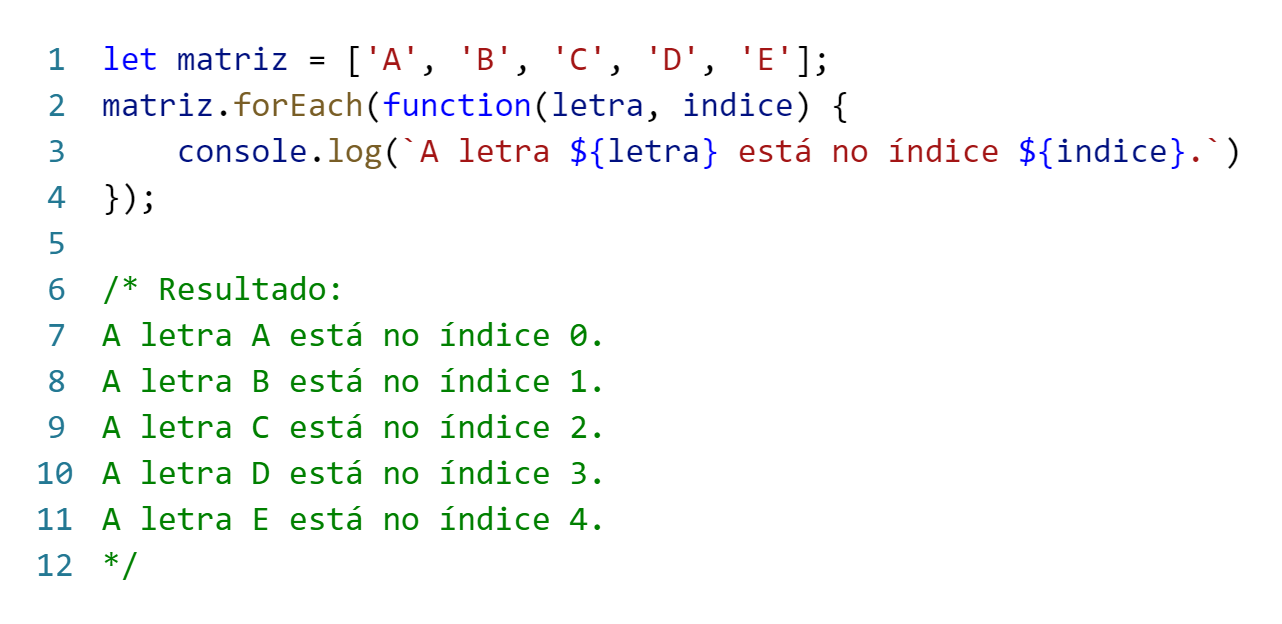
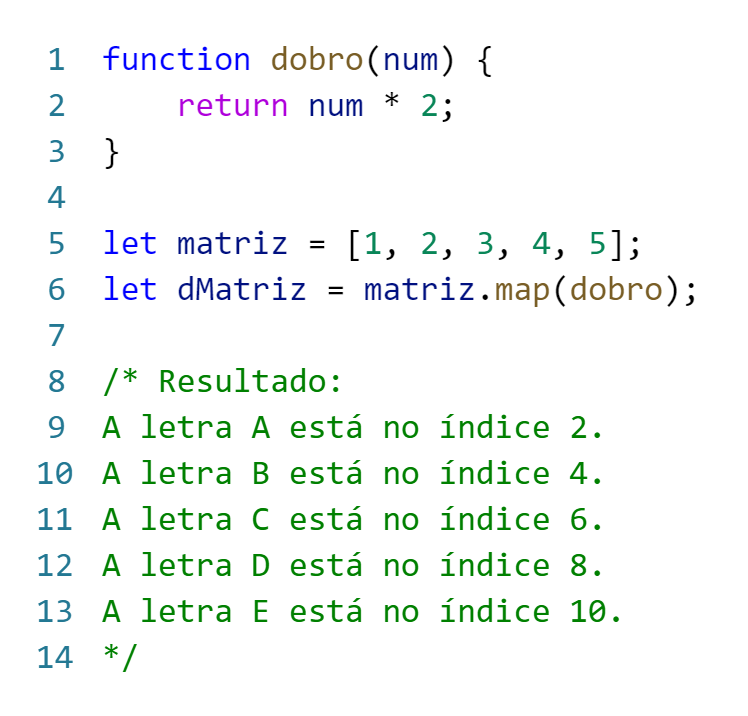
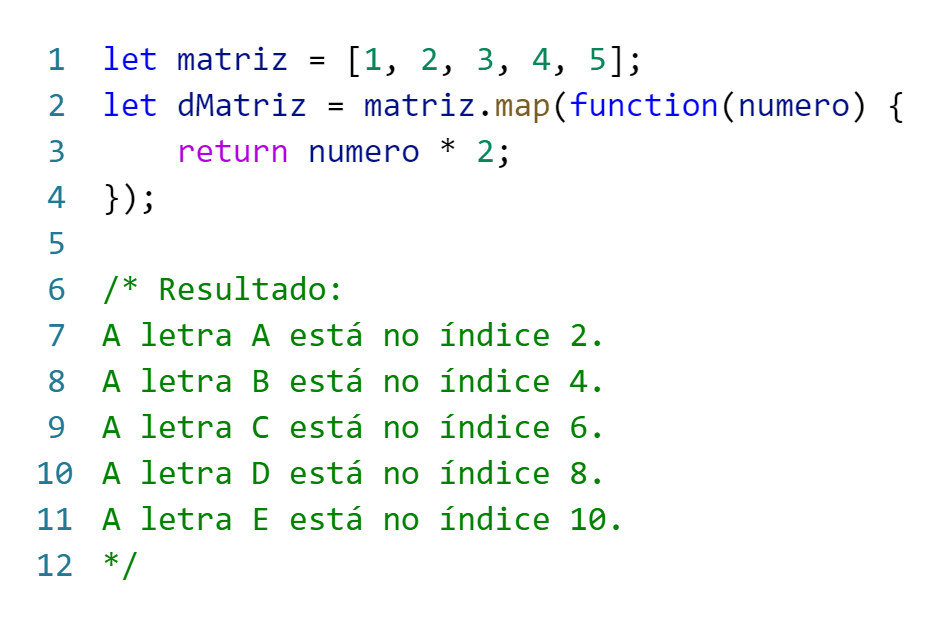
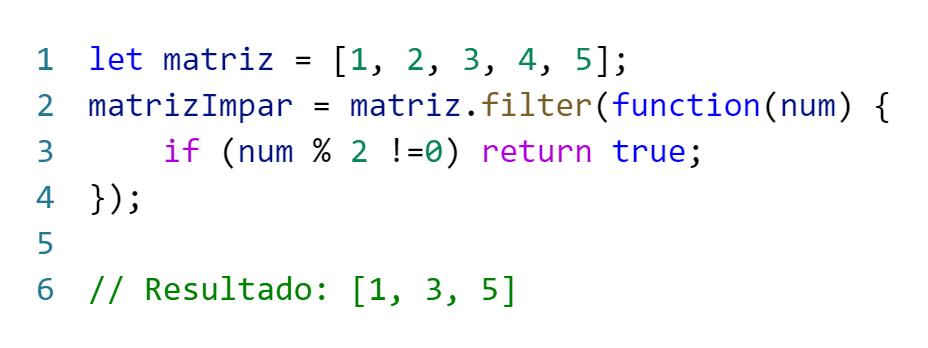
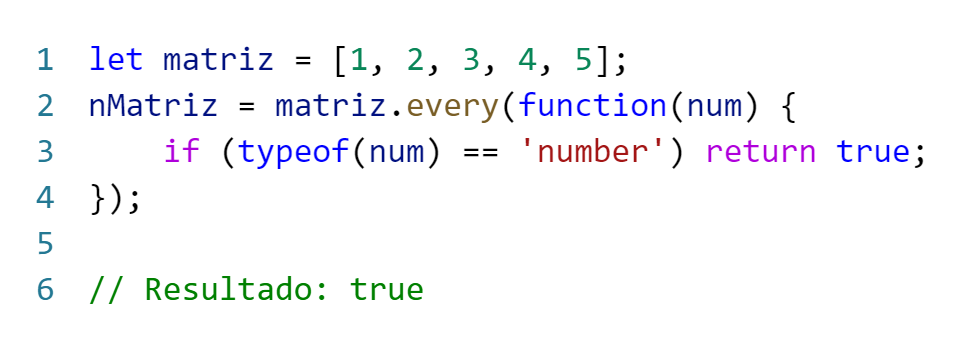
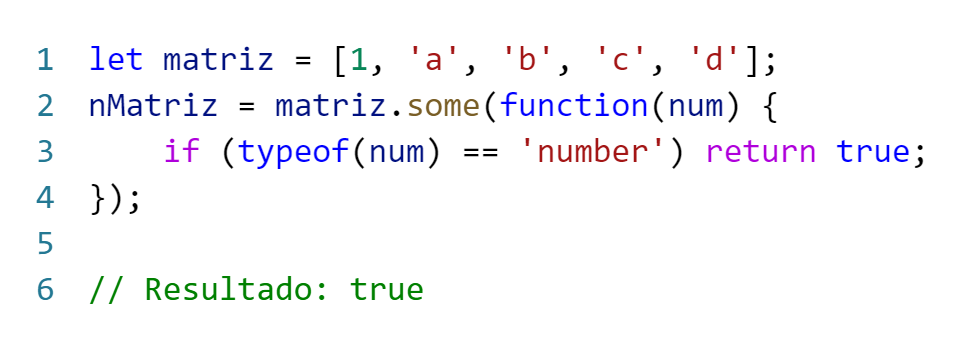
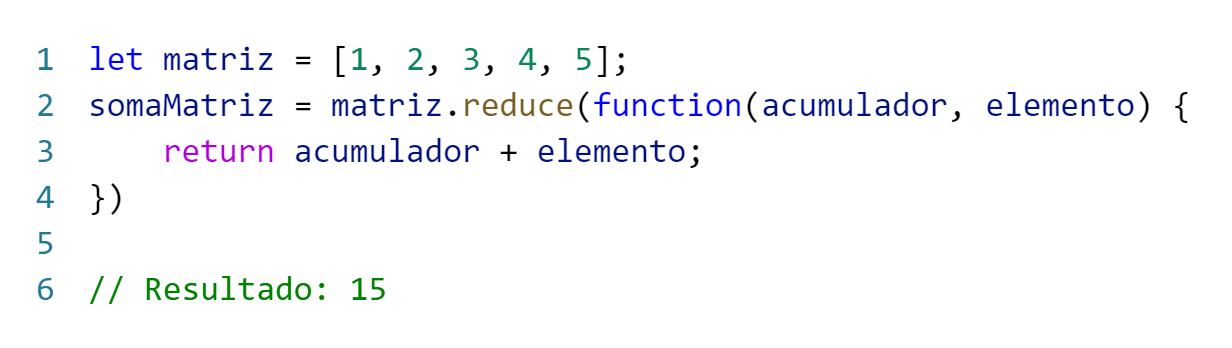
O acesso a um item dentro de uma matriz interna é feito pela posição, primeiro especificando a posição da matriz e depois a posição do item dentro dessa matriz.



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

## Métodos de Array

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

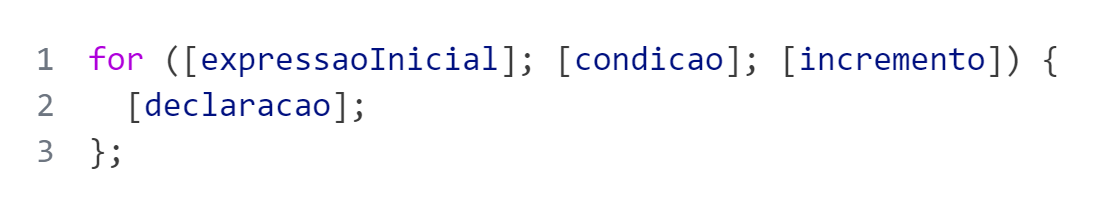
* **‘push()’:** Adiciona um ou mais elementos ao final da matriz original.  
  
* **‘pop()’:** Remove o último elemento da matriz e retorna o elemento removido.  
  
* **‘shift()’:** Remove o primeiro elemento da matriz e retorna o valor removido.  
  
* **‘unshift’:** Adiciona um ou mais elementos ao 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.

# Laços de Repetição

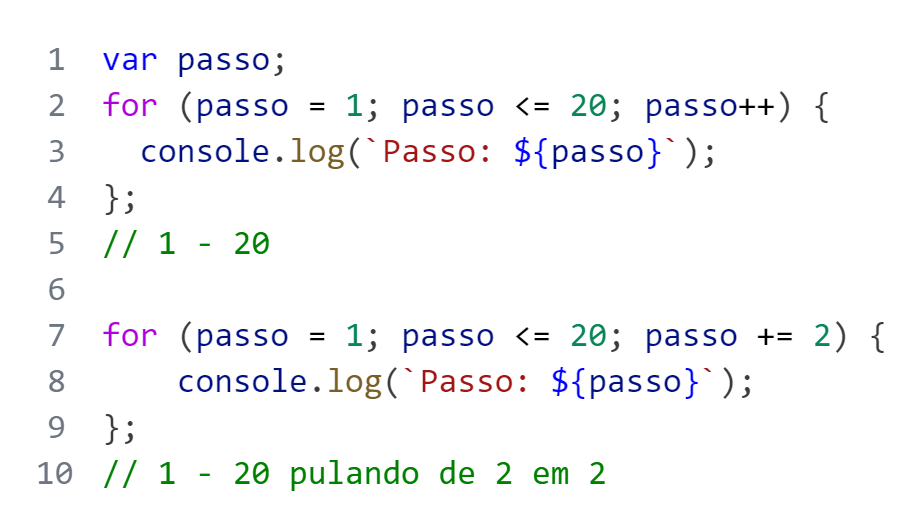
Em JavaScript existem diferentes formas de iterações.

## ‘for’

Um laço **‘for’** se repete até que a condição especificada seja falsa. Sua sintaxe é a seguinte:

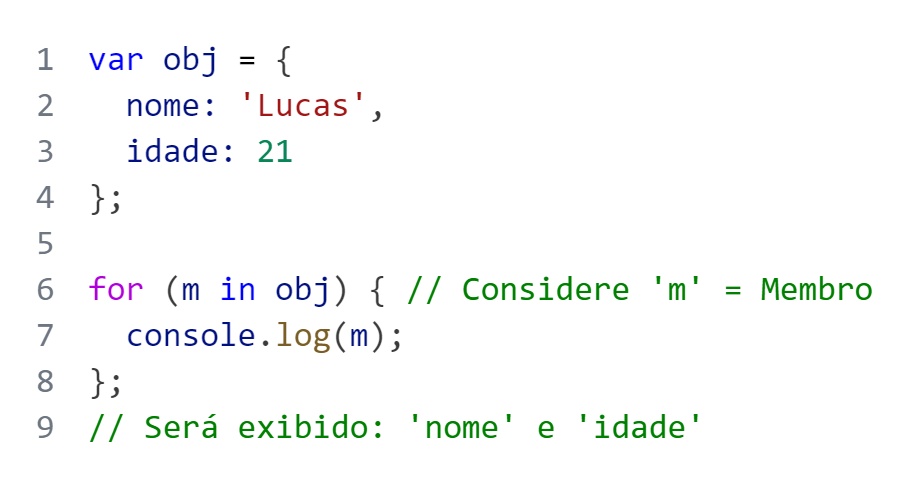


Um exemplo é:



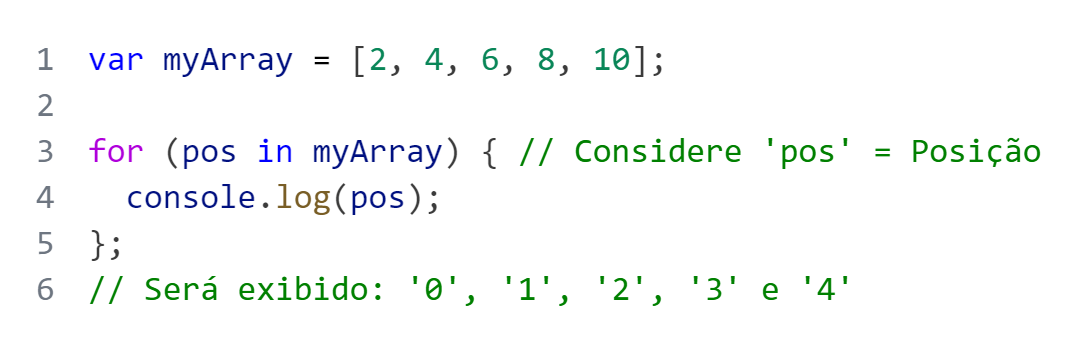
### ‘for...in’

A repetição será feita de acordo com a variável especificada, levando em conta todas as propriedades do objeto.



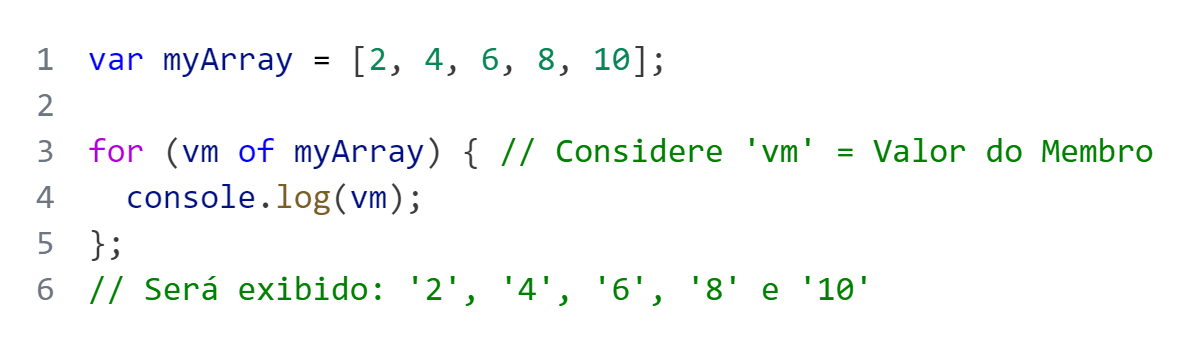
#### Interação com Arrays

São especificadas as posições de cada item de uma Array.



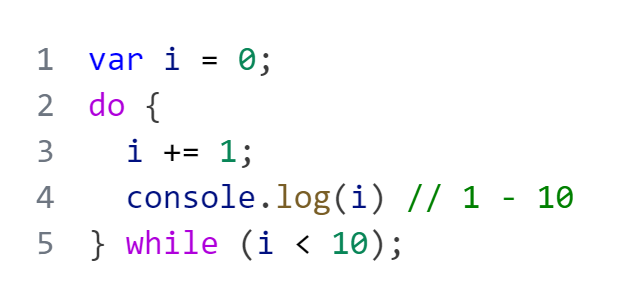
### ‘for...of’

Interage com o valor das propriedades.



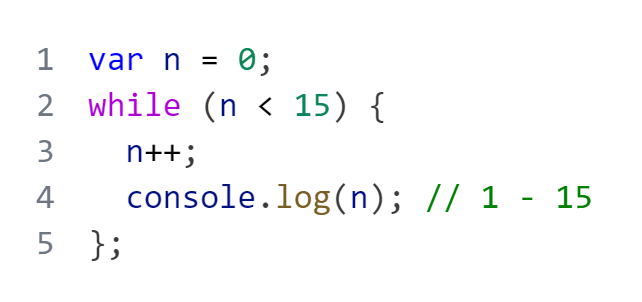
## ‘do...while’

Executa a declaração até que a condição se torne falsa.



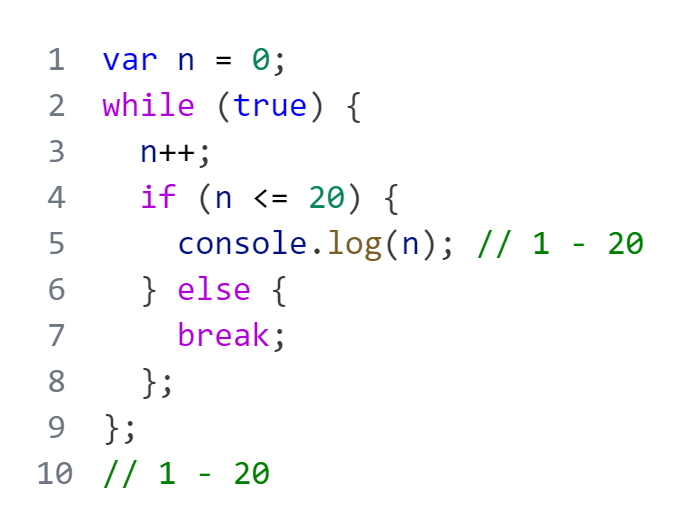
## ‘while’

Executa uma declaração enquanto sua condição for verdadeira.



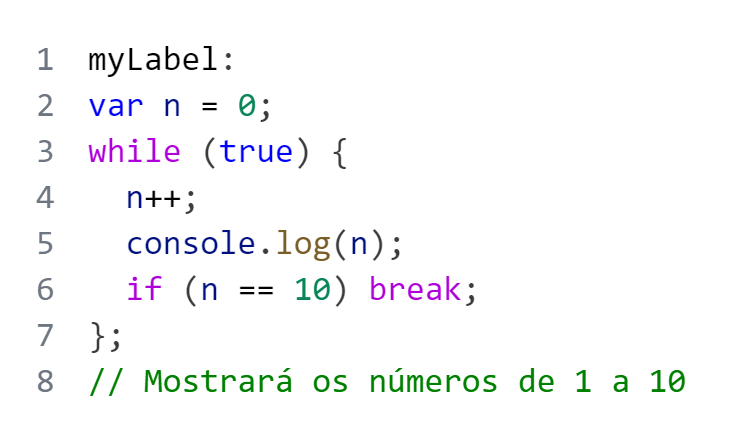
### Laço infinito

Se sua condição for **‘true’**, ou seja, verdadeira, o laço não terá fim até que haja uma declaração de parada (‘break’).



## ‘label’

Cria um identificador para uma repetição.



No código acima, a primeira linha indica o **‘label’**.

**ATENÇÃO**: Os labels não são comumente utilizados. Além disso, os mesmos devem ser evitados por tornar os códigos mais difíceis de ler.

## Declarações dentro de Laços

### ‘break’

Encerra os laços assim que executado, além do ‘switch’.

Caso seja utilizado em um **‘label’**, ele encerrará o **‘label’** específico.

Suas sintaxes são:

* **‘break;’**
* **‘break label;’**

### ‘continue’

Retorna o laço para o início.

No **‘for’**, ele contará a partir do próximo incremento. Já no **‘while’**, ele voltará para a revisão da condição.

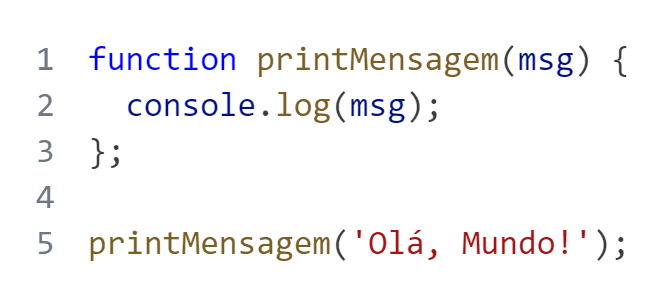
Suas sintaxes são:

* **‘continue;’**
* **‘continue label;’**

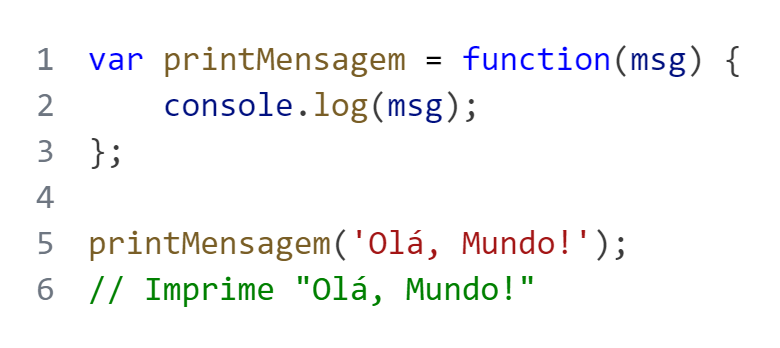
# Funções

A sintaxe é a seguinte:

**‘function <nomeFuncao>(<parm1>, <param2>..., <param5>) {<corpoDaFuncao>};’**



A mesma função também poderia ser:

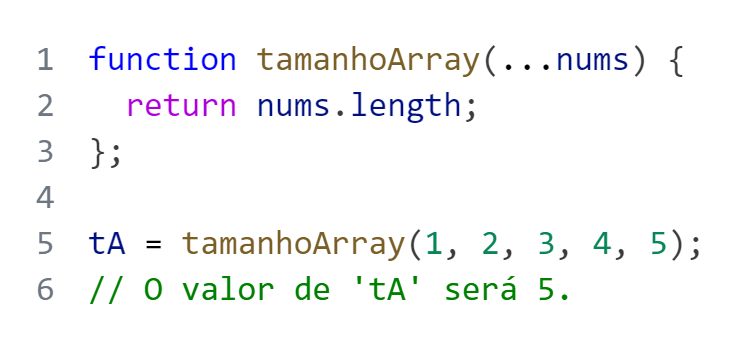


## Parâmetros

Podem ser passados um ou vários parâmetros, ou nenhum se não houver a necessidade.

### Parâmetros ‘rest’

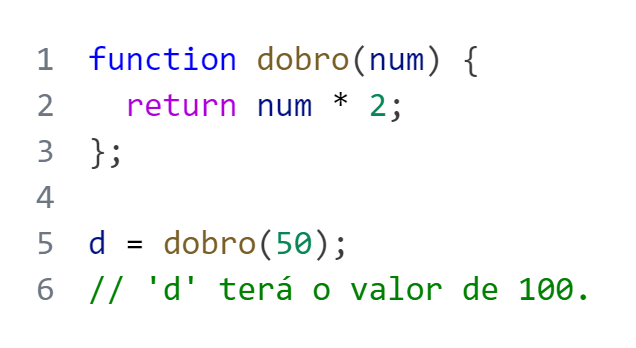
São adicionados três pontos (**‘...’**) antes do nome do parâmetro. É basicamente uma Array, e útil caso o número de parâmetros possa variar em diferentes execuções.



Módulos internos de Arrays funcionam normalmente.

## ‘return’

Retorna um valor para a variável que chama a função.

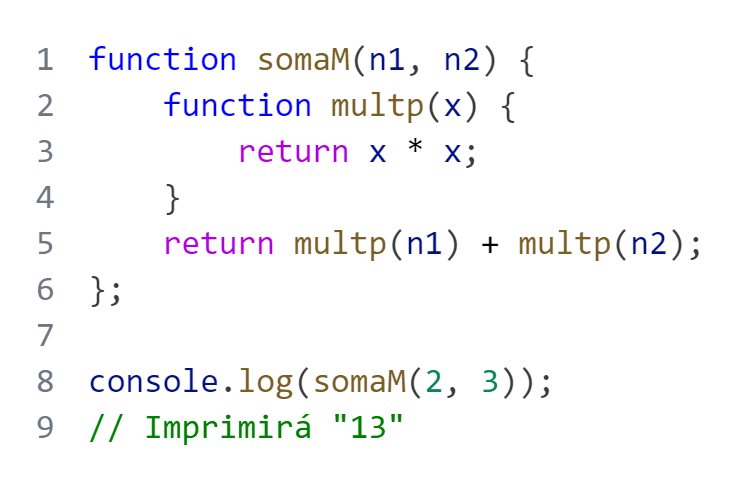


Caso não seja retornado nenhum valor, irá retornar **‘undefined’**.

Quando algum valor é retornado, a função é encerrada instantaneamente.

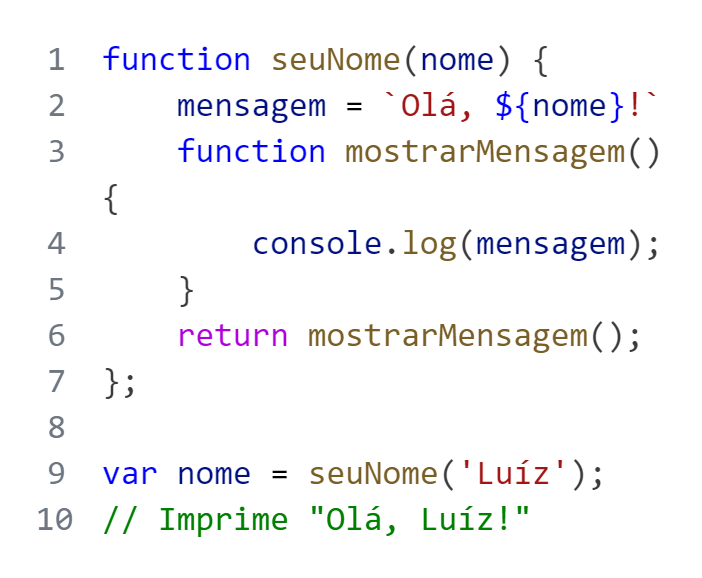
## Funções Aninhadas

É possível aninhar funções, colocando-as uma dentro das outras.

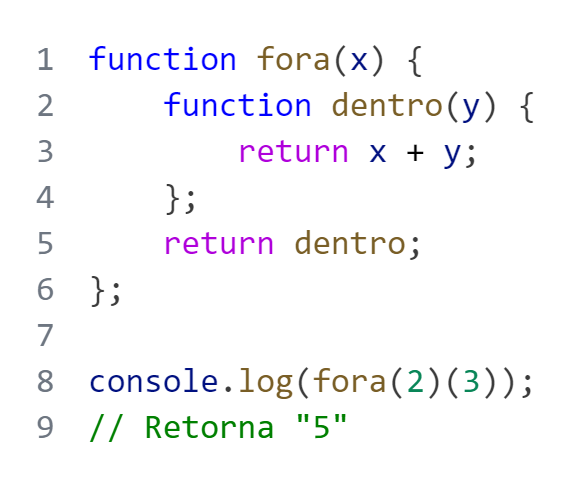


### *Closure*

Uma closure permite usar as variáveis que estão no escopo da função externa mesmo quando a mesma já foi fechada.

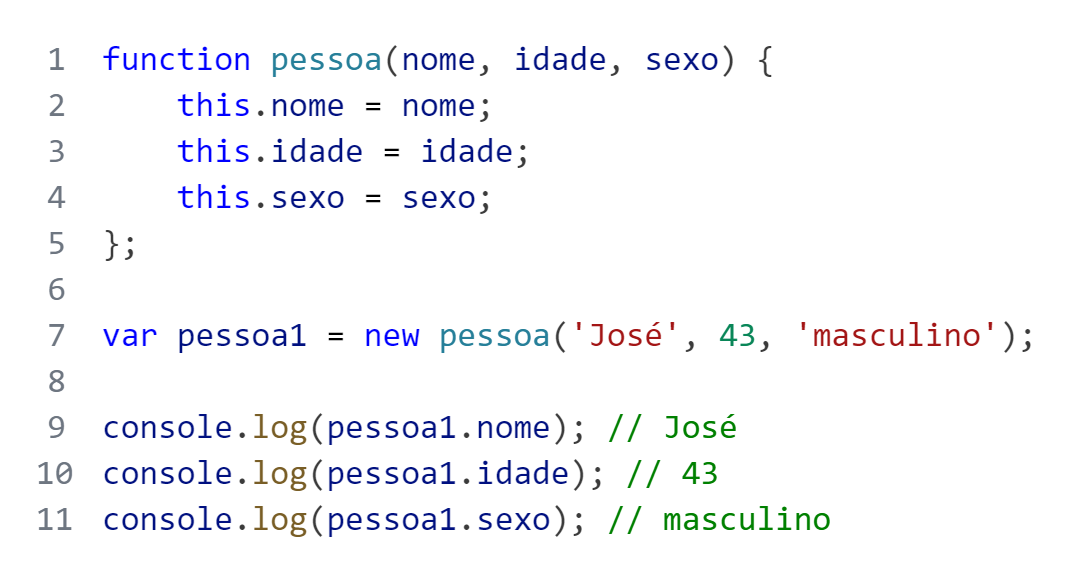


Além disso, é possível utilizar parâmetros também nas funções internas, e defini-los junto aos parâmetros da função externa.



## Operador ‘new’

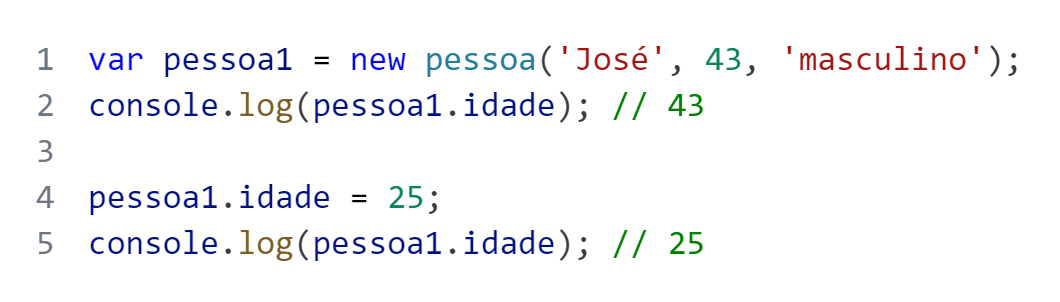
Cria uma instância de um tipo de objeto que possui uma função construtora.



Retornará **‘undefined’** caso o argumento não exista.

### Alterar argumentos

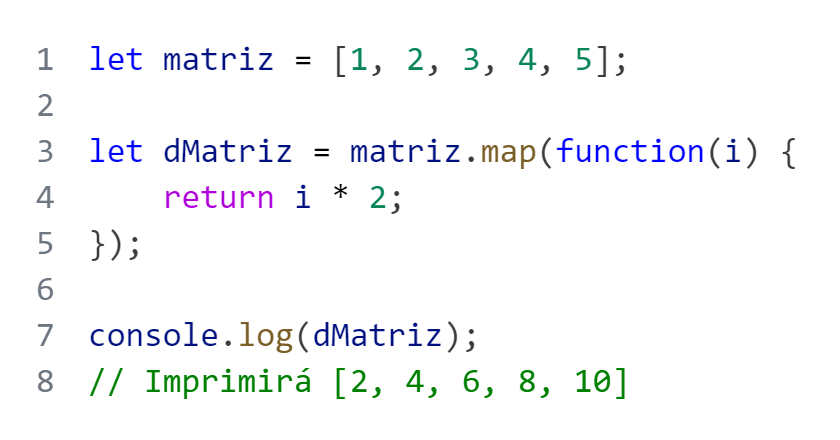
Para alterar a idade de José, por exemplo, podemos apenas declarar um novo valor para o argumento **‘idade’**.



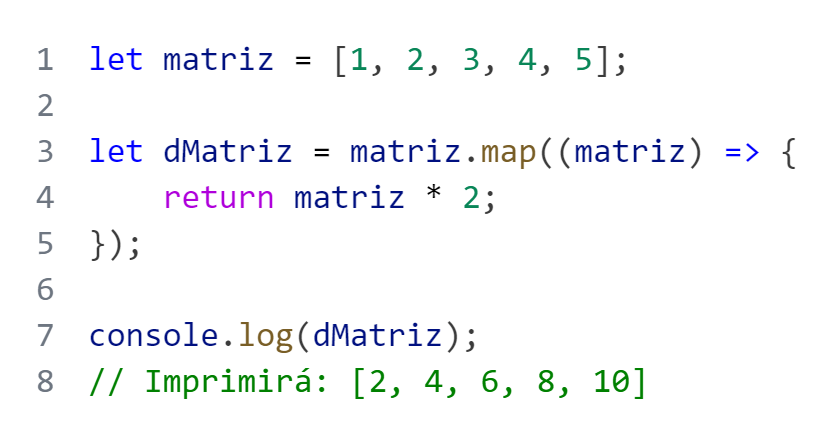
## *Arrows Function*

As funções de seta permitem escrever uma função de forma mais simples por meio dos caracteres **‘=>’**, que simbolizam uma seta.

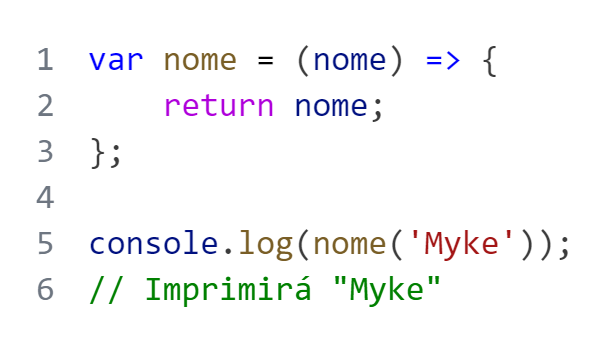
Primeiro um exemplo de função que dobra o valor de cada item da matriz, e depois um outro exemplo com a mesma função, porém utilizando a ***Arrow Function***.



Agora utilizando ***Arrow Function***:



Outro exemplo de função de seta.

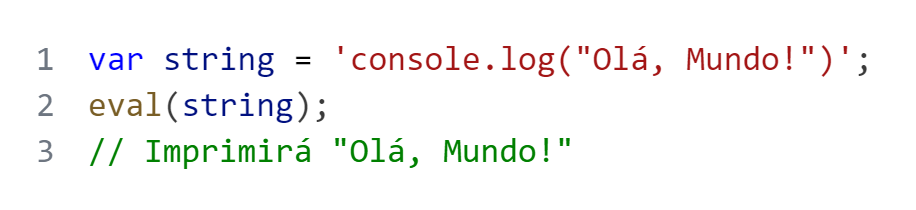


### Uso do operador ‘new’

As funções de seta não podem ser usadas como construtores e lançarão um erro quando usadas com o **‘new’**.

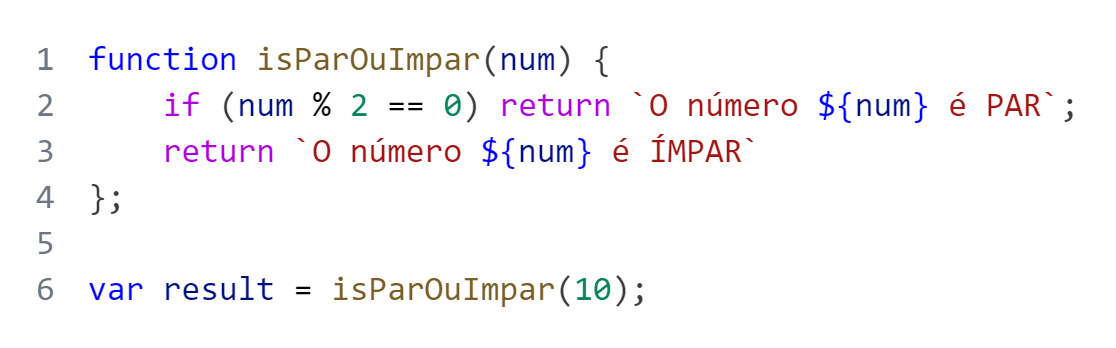
# Executar String

A função **‘eval()’** representa uma string como código JavaScript, podendo executá-lo.

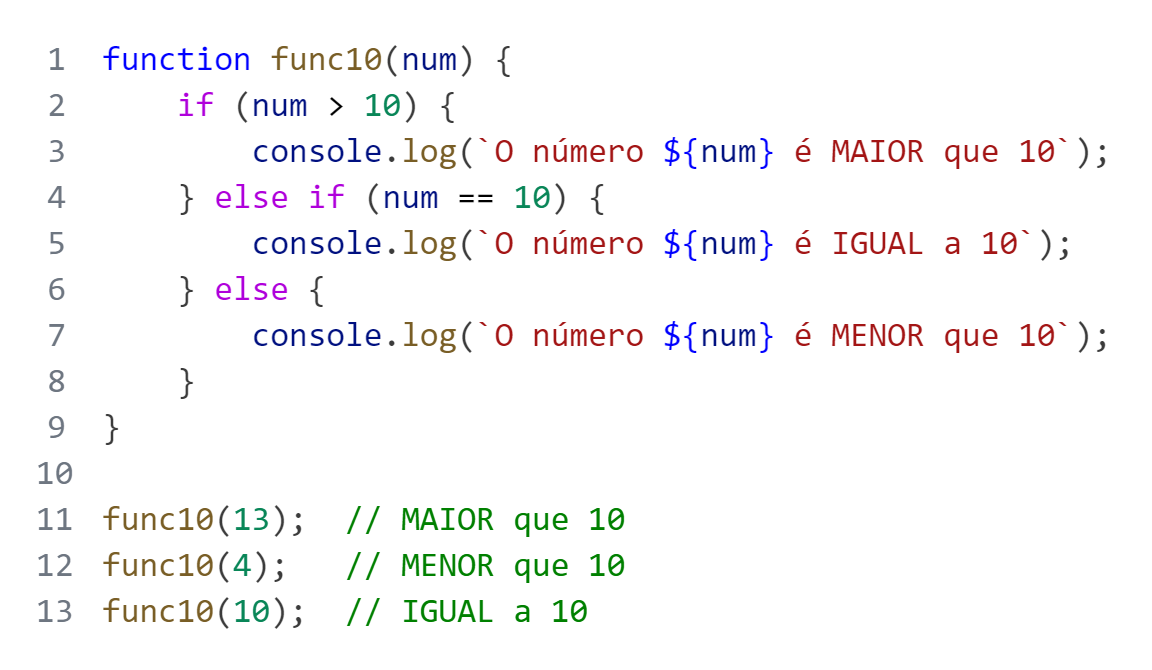


# Condições

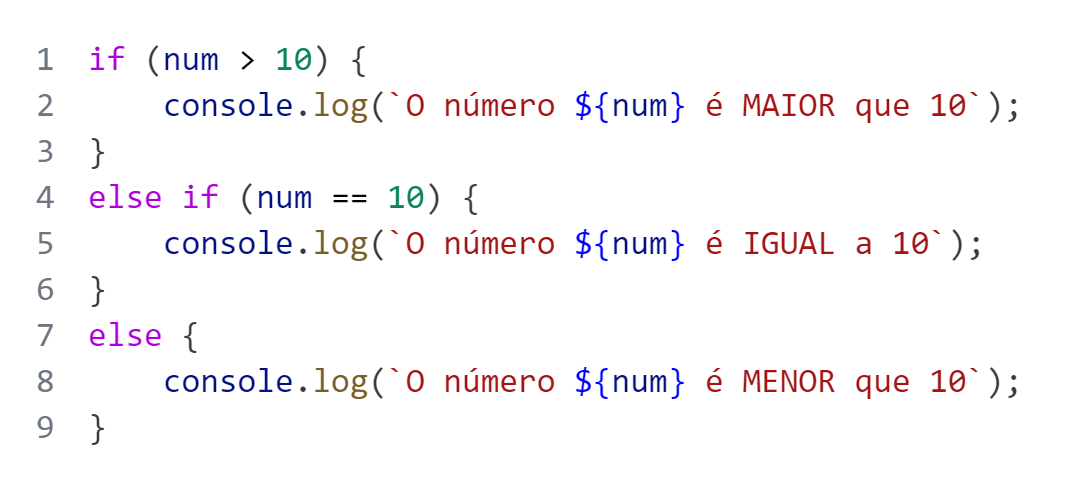
Existem três estruturas condicionais em JavaScript: **‘if’**, **‘else’** e **‘else if’**.



Aqui um outro exemplo, porém usando todas as três estruturas condicionais:



A próxima estrutura condicional também pode ser escrita abaixo:

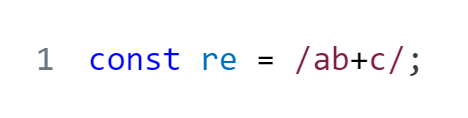


# Expressões Regulares

São padrões usados para selecionar combinações de caracteres em uma string.

Há duas maneiras de construir uma expressão regular:

Uma é construindo o padrão entre barras, por exemplo:



E a outra é chamando o **‘RegExp’**, que é um construtor de objeto.

