**Arrays**

Quando trabalhamos com arrays, chamamos de elemento ou item cada um dos valores da lista. A posição do item é chamada de índice.

O array pode ser declarado como uma *const* e mesmo assim ser alterado atráves dos próprios métodos do JS. O que não pode acontecer com uma *const* é receber uma nova atribuição.

**Métodos para arrays**

*.concat() ->* junta dois arrays, sendo um deles passado por parâmetro. Não altera os arrays, por isso precisa ser atribuído à um novo array

*.filter() ->* retorna uma lista contendo todos os elementos que passaram em um teste de uma função feita por nós. Também não altera o array

*.find() ->* parece com o filter porém retorna apenas o primeiro valor verdadeiro, podendo ser string ou número

*.findIndex() ->* retorna o índice ao invés do elemento

*.lastIndexOf() ->* é igual ao findIndex, mas começa do último elemento.

*.forEach() ->* executa uma função em cada elemento do array de forma individual. Não altera o array original, nem retorna um valor

.*pop() ->* remove o último elemento

.*shift() ->* retira o primeiro elemento do array. Altera o array original

.*push() ->* adiciona elemento na última posição. Altera o array

*.unshift() ->* parece o push, porém adiciona na primeira posição. Altera o array.

.*reduce() ->* utiliza uma função definida pelo usuário em cada um dos elementos, guardado o resultado em uma variável que pode ser acessada dentro da função que foi definida, retornando um único valor no final, reduzindo o array para um único valor.

.*reduceRight() ->* começa do final do array e segue até o início.

.*reverse() ->* inverte a ordem dos elementos

.*slice() ->* copia parte do array para outro array

.*sort() ->* organiza de acordo com a classificação Unicode. Não funciona corretamente para números, uma função tem que ser definifa para auxiliar o comando

.*splice() ->* remove um ou mais elementos consecutivos caso o segundo parâmetro tenha um valor maior que 0, e consegue incluir um ou mais elementos a partir de um índice escolhido.

**->** É possível fazer um array vazio, adicionar espaços a esses arrays (simplesmente adicionando virgulas) e o JS vai reconhecer esses espaços e reconhecer que não há elementos nele. Ainda assim podemos adicionar, com o método push, elementos no final do array. O array terá espaços vazios e o elemento no final, formando um **array esparso**

**-**> Usando o método *slice* o segundo parâmetro é da posição final que será cortado a lista, porém esse elemento não entra na contagem, ou seja, se quisermos cortar da primeira posição até a quinta, fica ***(0,5)***, sendo que o elemento do indicie [5] não será incluído. Outra peculiaridade é que o último parâmetro é opcional, se o objetivo for ir até o final da lista

-> O método *splice* pode ser usado para excluir itens. Os parâmetros passados são: primeiro a posição do elemento a ser excluído, e depois quantos elementos após ele. Também é possível passar um terceiro parâmetro que vai ser o elemento que irá entrar no lugar. O *splice* altera a lista original.

-> É possível criar lista com mais de uma dimensão, ou seja, uma lista composta de mais de uma lista. E para acessar um elemento dentro da lista secundária é feito como se fosse uma matriz, primeiro acessa a lista que está guardando as outras listas e depois o índice do elemento que queremos na lista secundária.

-> quando usamos o método *.concat()* e passamos um array como parâmetro, os elementos desse array que será concatenado, não o array em si. E se concatenarmos em um array elemento passados como parâmetro, esses elementos irão fazer parte do array principal.

-> Usando o método ***.includes()*** é possível ver se algum item consta em um array, o item em questão é passado por parâmetro.

-> o método ***.indexOf()*** permite descobrir o índice de um item, passado por parâmetro

Cada execução do bloco for é chamada de iteração.

**Ctrl+C** no terminal encerra o programa

O **for of** é muito bom para percorrer um array

O ***forEach()*** é um método que recebe como parâmetro uma função anônima que é uma função call-back (nome que uma função que é parâmetro de outra recebe). Essa função pode receber como parâmetro um nome genérico que vai ser o elemento do array o qual o método está associado. (É uma lógica interna do js). Outro parâmetro que essa função anônima pode fornecer é o índice dos elementos.

-> Ao usarmos funções call-back o mais normal é usar *arrow functions*

Diferente do **for of** que só fornece o valor dos elementos, o **forEach** fornece o valor e o índice

No **forEach**, as funções call-backs podem ser *arrow function*, ou até mesmo uma função criada anteriormente. Quando passamos uma função que já foi criada anteriormente passamos apenas a referência dessa função, quando fazemos isso, não colocamos ‘()’ depois do nome da função.

Ao contrário do **for** e **for...of**, o **forEach()** é um método do objeto array. (Método = função que é propriedade de um objeto). Sendo assim esse método trabalha apenas com esse tipo de dado, array

-> um iterável representa uma sequência de elementos que pode ser percorrida (ou seja, iterada) utilizando ferramentas próprias para isso. Nem todo método que funciona em um array vai funcionar em outros iteráveis.

Iteráveis = arrays, strings, sets e maps

O **for...of** pode ser executado utilizando qualquer tipo de iterável (arrays, strings, sets e maps) são considerados objetos iteráveis

-> O método **map()** é muito útil quando queremos modificar um array, entretanto ele não altera o array original então tem que ser atrelado à um novo array.

-> O método **.filter()** não modifica o array original e precisamos retornar *verdadeiro* ou *falso* para incluir um elemento no novo array

-> O Js tem um operador chamado **spread**, que é espalhamento, e que pode ser representado por reticências ‘...’. Quando usamos isso com um array para criar um novo, significa que estamos “espalhando” os elementos desse array, que na verdade é que estamos copiando um array para outro, mas apenas os elementos. É um operador muito útil para clonar arrays sem ter o problema de cópia da referência de memória.

Com tipos primitivos, string, números e booleanos, o js não se comporta dessa maneira, isto é, somos capazes de copiar um valor primitivos usando o sinal de atribuição ‘=’. O js entende que é uma cópia e cria uma nova variável com seu próprio espaço na memória, isso é chamado de **atribuição por valor**. Os arrays não são tipos primitivos e o js otimiza a “cópia” usando o mesmo espaço de memória para os dois arrays, como se eles de fato fossem iguais, a atribuição feita é chamada de **atribuição por referência**.

\**quando uma arrow function vai apenas retornar um valor, podemos omitir o return e tirar as chaves {}*

*\*quando podemos usar dois parâmetros em uma função, mas não queremos usar o primeiro, ainda assim temos que declarar ele para poder acessar o segundo. É uma convenção da programação declarar como ‘\_’ o parâmetro que não iremos usar.*

-> **new Set()** o Set é uma classe do js e com ele vem algumas regras diferentes e uma delas é que não pode ter elementos repetidos, ele tira automaticamente.