step08_arrays

Arrays

An **array** in TypeScript is a collection of values of the same type. TypeScript enhances JavaScript's array capabilities by providing strong typing, which helps catch errors during development.

Using the Array Type

```
let numbers: Array<number> = [1, 2, 3, 4, 5];
```

Using Square Brackets

```
let strings: string[] = ["apple", "banana", "cherry"];
```

Defining and Initializing an Array

```
let fruits: string[] = ["apple", "banana", "cherry"];

let scores: Array<number> = [90, 80, 70];
```

Accessing Array Elements

```
console.log(fruits[0]); // Output: "apple"

console.log(scores[1]);  // Output: 80
```

Modifying Array Elements

```
fruits[1] = "orange";

console.log(fruits); // Output: ["apple", "orange", "cherry"]
```

Multidimensional Arrays

```
let matrix: number[][] = [

   [1, 2, 3],

   [4, 5, 6],

   [7, 8, 9]];

console.log(matrix[1][2]); // Output: 6
```

Using Tuples

```
let user: [string, number] = ["Alice", 30];
```

Using Union Types

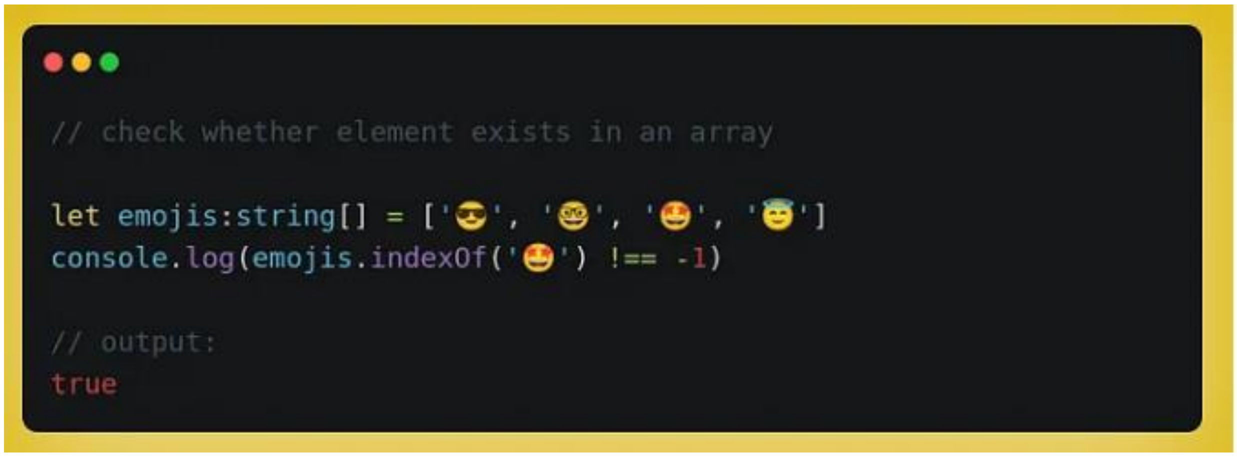
```
let mixedArray: (string | number)[] = ["apple", 42, "banana", 30];
```

indexOf()

Every array element has an index. This method returns the index of an element in an array.

syntax:

array.indexOf(element)



```
// check whether element exists in an array

let emojis:string[] = ['😎', '🤓', '🤩', '😇']
console.log(emojis.indexOf('🤩') !== -1)

// output:
true
```

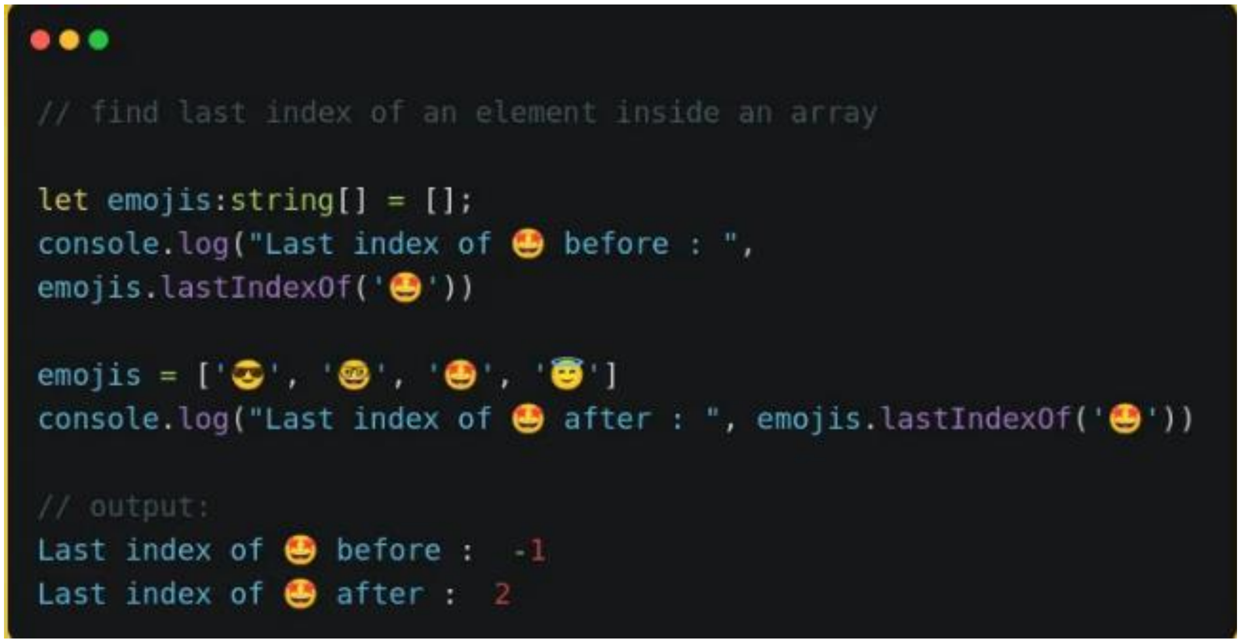
2. lastIndexOf()

This method returns an array’s last element’s index.

a. If an array is empty then, it returns -1 as of the indexOf() function.
   b. If an array has one more same element, then it returns the maximum index of duplicate items.

syntax:

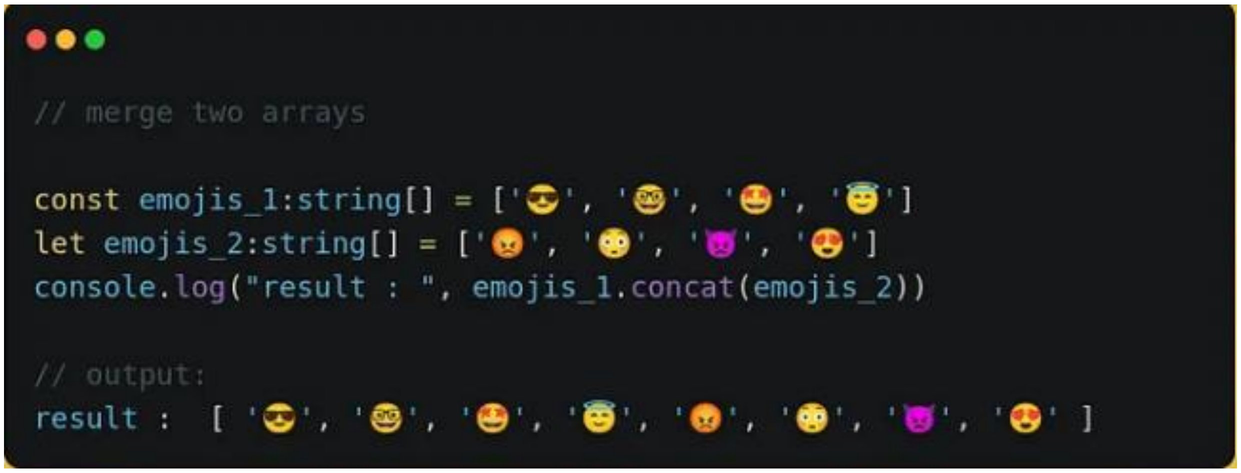
array.lastIndexOf(element)



```
// find last index of an element inside an array

let emojis:string[] = [];
console.log("Last index of 🤩 before : ",
emojis.lastIndexOf('🤩'))

emojis = ['😎', '🤓', '🤩', '😇']
console.log("Last index of 🤩 after : ", emojis.lastIndexOf('🤩'))

// output:
Last index of 🤩 before :  -1
Last index of 🤩 after :  2
```

3. concat()
As the name suggests, this method simply merges two arrays and returns a combined result.

syntax:
array1.concat(array2)

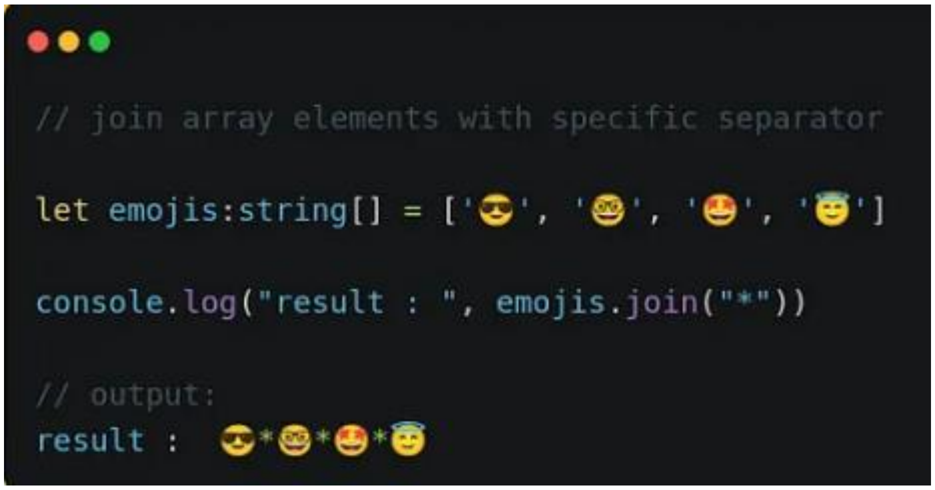


```
// merge two arrays

const emojis_1:string[] = ['😎', '🤓', '🤩', '😇']
let emojis_2:string[] = ['😡', '😳', '👿', '😍']
console.log("result : ", emojis_1.concat(emojis_2))

// output:
result :  [ '😎', '🤓', '🤩', '😇', '😡', '😳', '👿', '😍' ]
```

4. join()
According to the name, this method joins all elements of the array into a string with a given operator.

a. If an operator is not given, it joins elements with a comma(,).

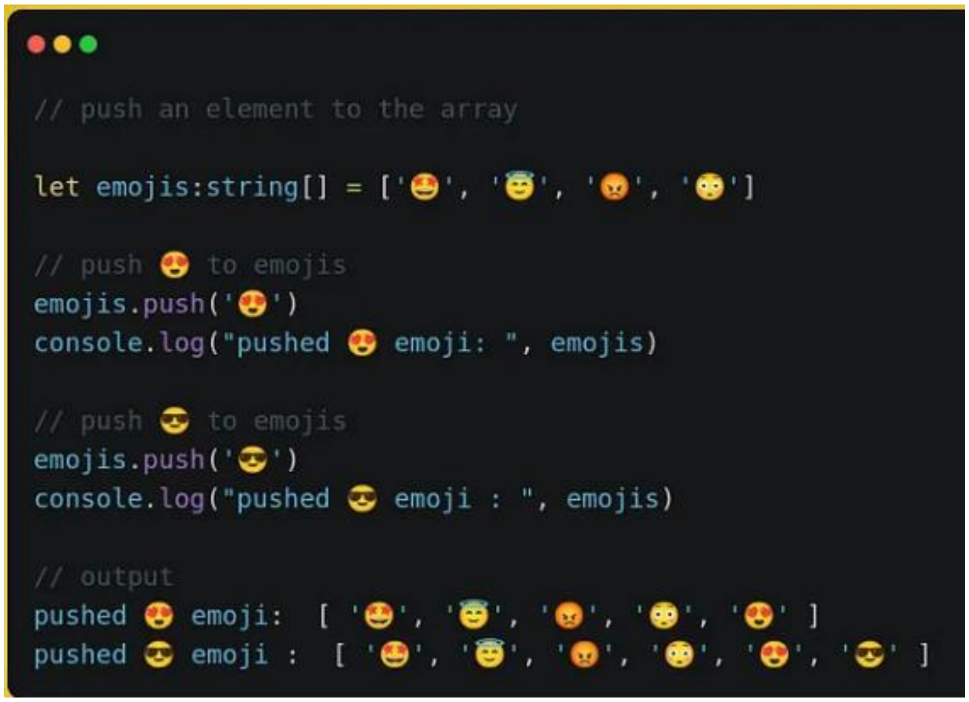
syntax:
array.join(operator)



```
// join array elements with specific separator

let emojis:string[] = ['😎', '🤓', '🤩', '😇']

console.log("result : ", emojis.join("*"))

// output:
result :  😎*🤓*🤩*😇
```

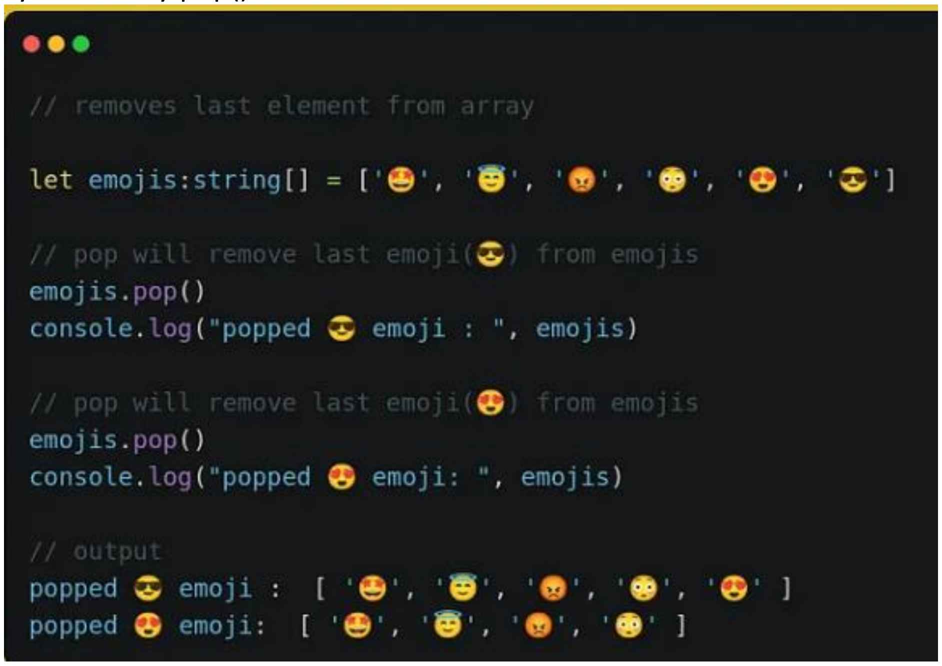
5. push()
This method pushes/adds one or more elements to the array at the last of an array.

syntax: array.push(element)



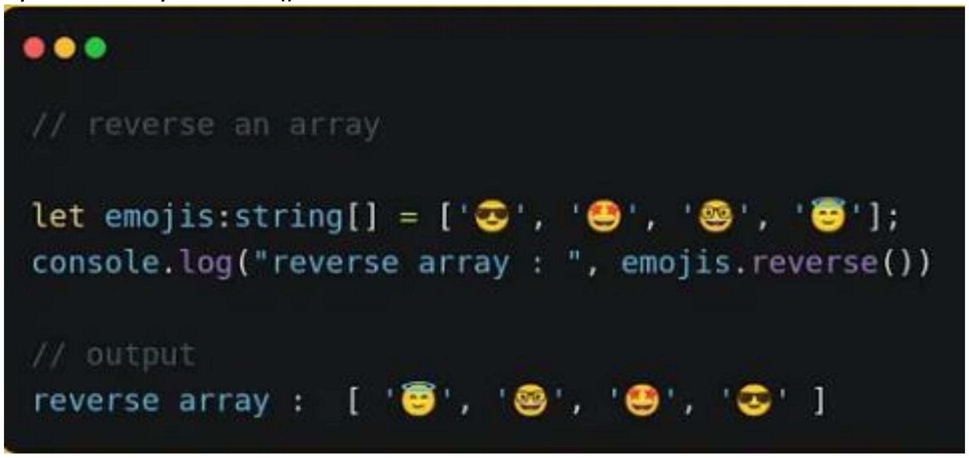
```
// push an element to the array

let emojis:string[] = ['🤩', '😇', '😡', '😳']

// push 😍 to emojis
emojis.push('😍')
console.log("pushed 😍 emoji: ", emojis)

// push 😎 to emojis
emojis.push('😎')
console.log("pushed 😎 emoji : ", emojis)

// output
pushed 😍 emoji:  [ '🤩', '😇', '😡', '😳', '😍' ]
pushed 😎 emoji :  [ '🤩', '😇', '😡', '😳', '😍', '😎' ]
```

6. pop()
This method pops/removes the last element from an array.
syntax: array.pop()



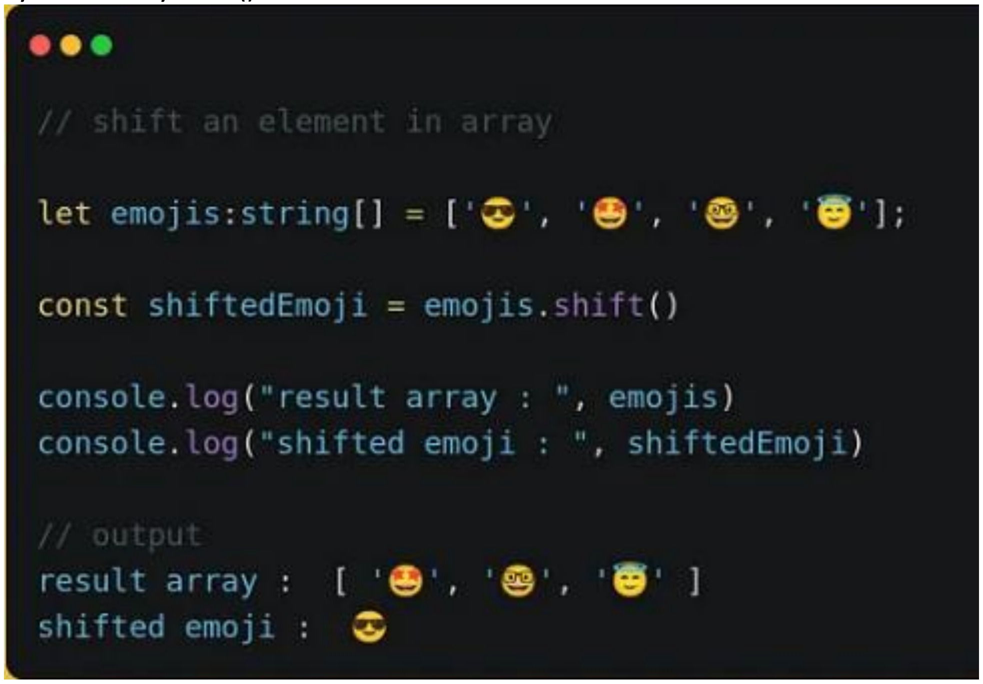
```
// removes last element from array

let emojis:string[] = ['🤩', '😇', '😡', '😳', '😍', '😎']

// pop will remove last emoji(😎) from emojis
emojis.pop()
console.log("popped 😎 emoji : ", emojis)

// pop will remove last emoji(😍) from emojis
emojis.pop()
console.log("popped 😍 emoji: ", emojis)

// output
popped 😎 emoji :  [ '🤩', '😇', '😡', '😳', '😍' ]
popped 😍 emoji:  [ '🤩', '😇', '😡', '😳' ]
```

7. reverse()
As per the name, this method reverts the order of an array.
syntax: array.reverse()



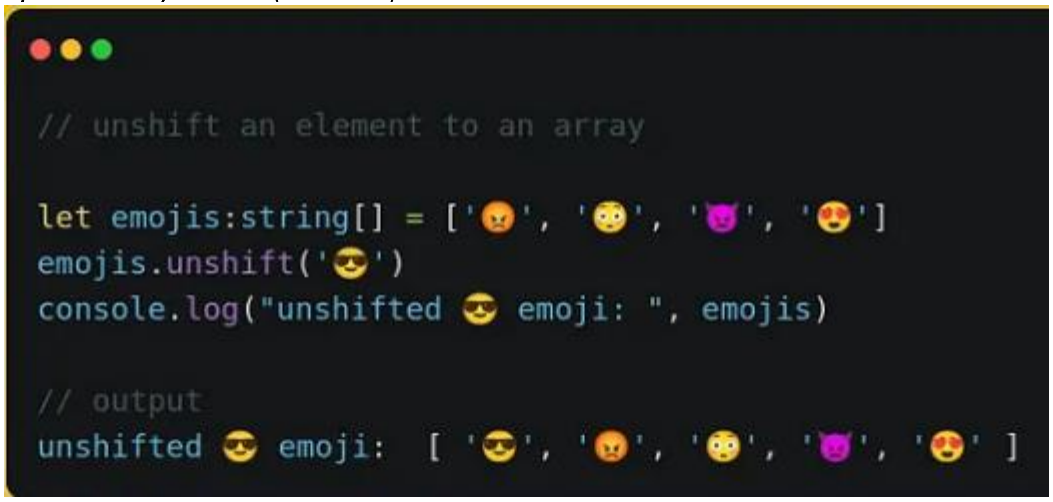
```
// reverse an array

let emojis:string[] = ['😎', '🤩', '🤓', '😇'];
console.log("reverse array : ", emojis.reverse())

// output
reverse array :  [ '😇', '🤓', '🤩', '😎' ]
```

8. shift()
This method removes starting(first) element from an array and returns the removed element. We can say that it’s the exact opposite of pop() method, which removes the last element and returns the result.
syntax: array.shift()



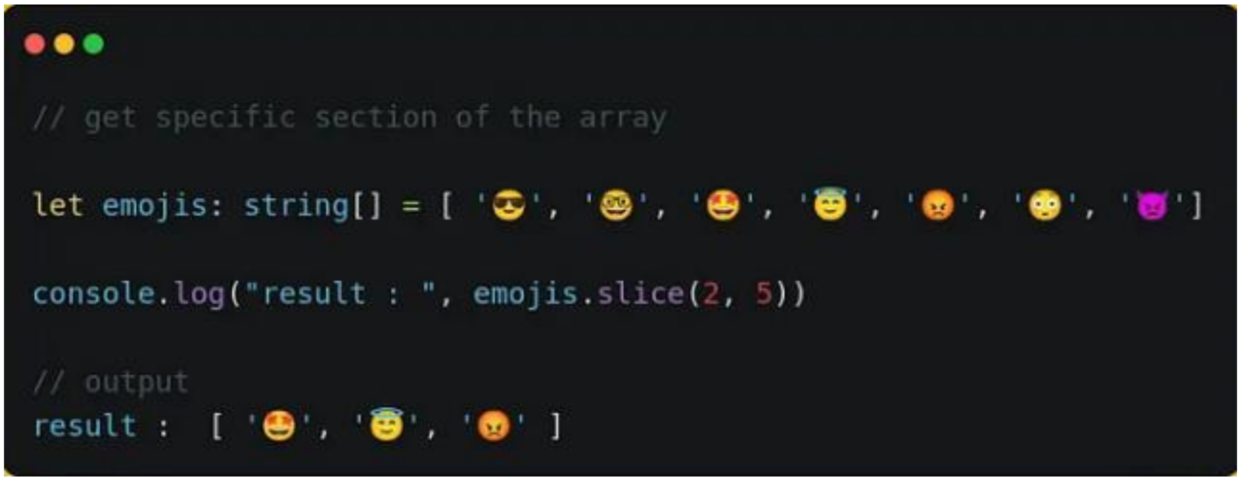
```
// shift an element in array

let emojis:string[] = ['😎', '🤩', '🤓', '😇'];

const shiftedEmoji = emojis.shift()

console.log("result array : ", emojis)
console.log("shifted emoji : ", shiftedEmoji)

// output
result array :  [ '🤩', '🤓', '😇' ]
shifted emoji :  😎
```

9. unshift()
It has the exact opposite behavior to the shift() method. It adds an element at starting of an array and returns a new array.
syntax: array.unshift(element)



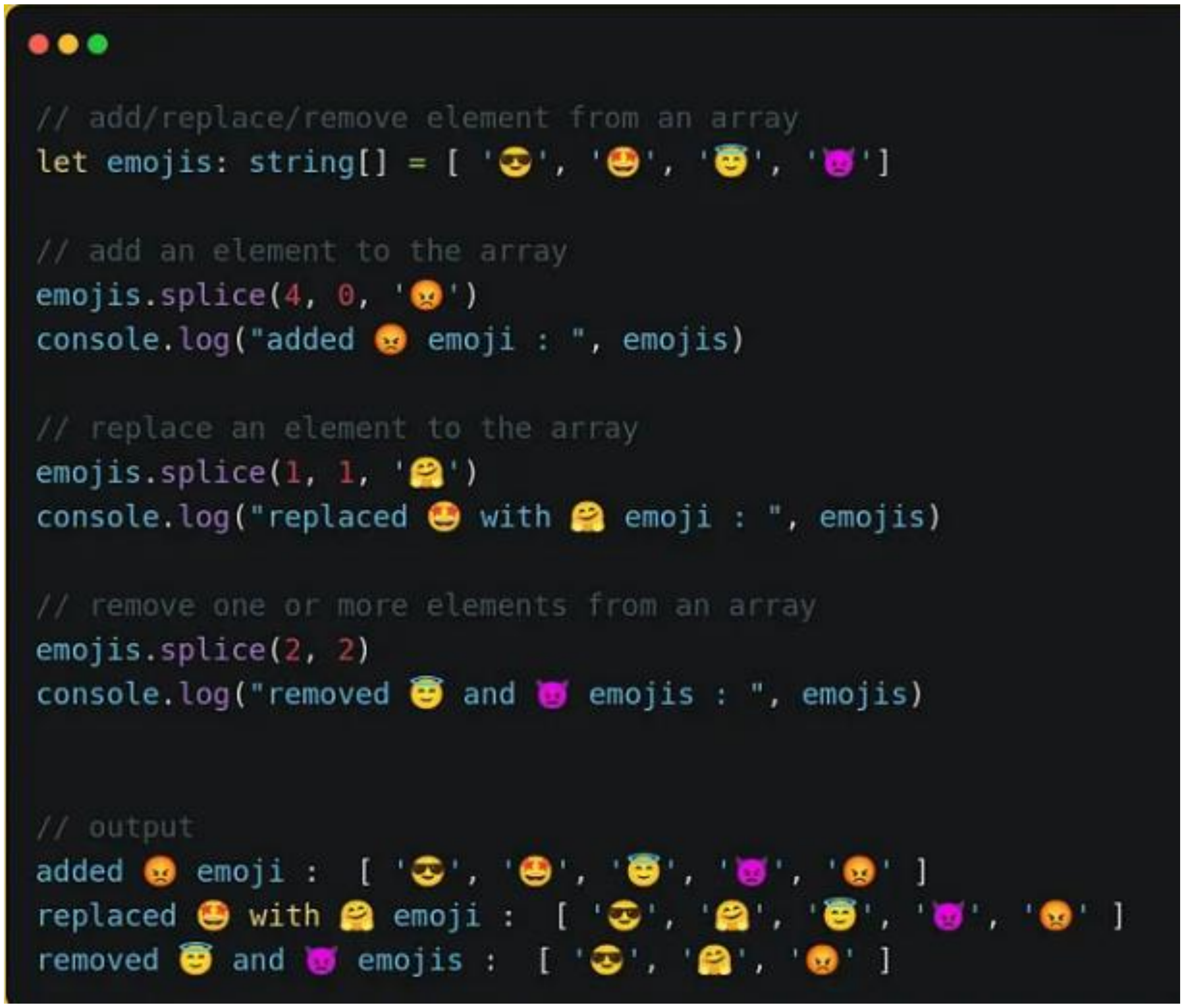
```
// unshift an element to an array

let emojis:string[] = ['😡', '😳', '😈', '😍']
emojis.unshift('😎')
console.log("unshifted 😎 emoji: ", emojis)

// output
unshifted 😎 emoji:  [ '😎', '😡', '😳', '😈', '😍' ]
```

10. slice()
This method cuts an array, in whichever manner we want and returns the trimmed array.
a. It excludes the last index from an argument.
syntax: array.slice(start_index, end_index)



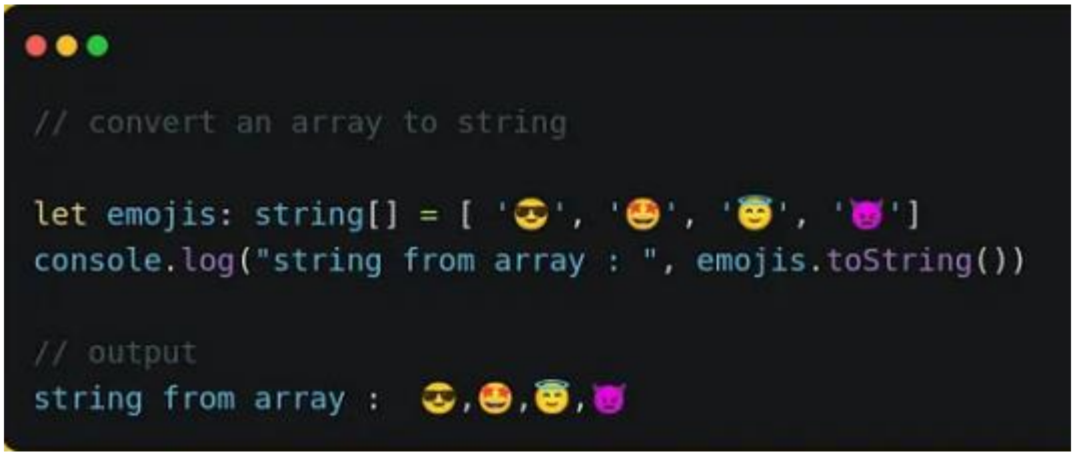
```
// get specific section of the array

let emojis: string[] = [ '😎', '🤓', '🤩', '😇', '😡', '😳', '😈']

console.log("result : ", emojis.slice(2, 5))

// output
result :  [ '🤩', '😇', '😡' ]
```

11. splice()
This method can be used for multiple purposes. For,
1. Add an element to an array
2. Replace specific elements within an array
3. Remove specific elements from an array
syntax: array.splice(index, number of elements to be removed, element1,..,elementN)



```
// add/replace/remove element from an array
let emojis: string[] = [ '😎', '🤩', '😇', '😈']

// add an element to the array
emojis.splice(4, 0, '😡')
console.log("added 😡 emoji : ", emojis)

// replace an element to the array
emojis.splice(1, 1, '🤗')
console.log("replaced 🤩 with 🤗 emoji : ", emojis)

// remove one or more elements from an array
emojis.splice(2, 2)
console.log("removed 😇 and 😈 emojis : ", emojis)


// output
added 😡 emoji :  [ '😎', '🤩', '😇', '😈', '😡' ]
replaced 🤩 with 🤗 emoji :  [ '😎', '🤗', '😇', '😈', '😡' ]
removed 😇 and 😈 emojis :  [ '😎', '🤗', '😡' ]
```

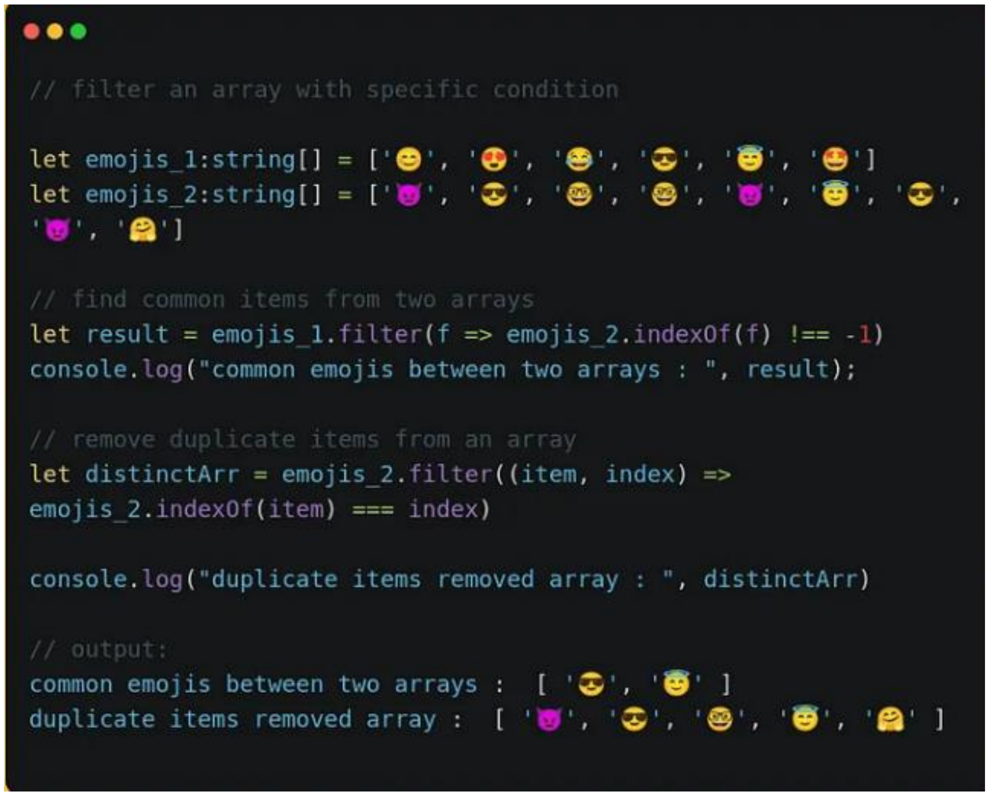
12. toString()
This method converts an array to a comma-separated string.
syntax: array.toString()



```
// convert an array to string

let emojis: string[] = [ '😎', '🤩', '😇', '😈']
console.log("string from array : ", emojis.toString())

// output
string from array :  😎,🤩,😇,😈
```

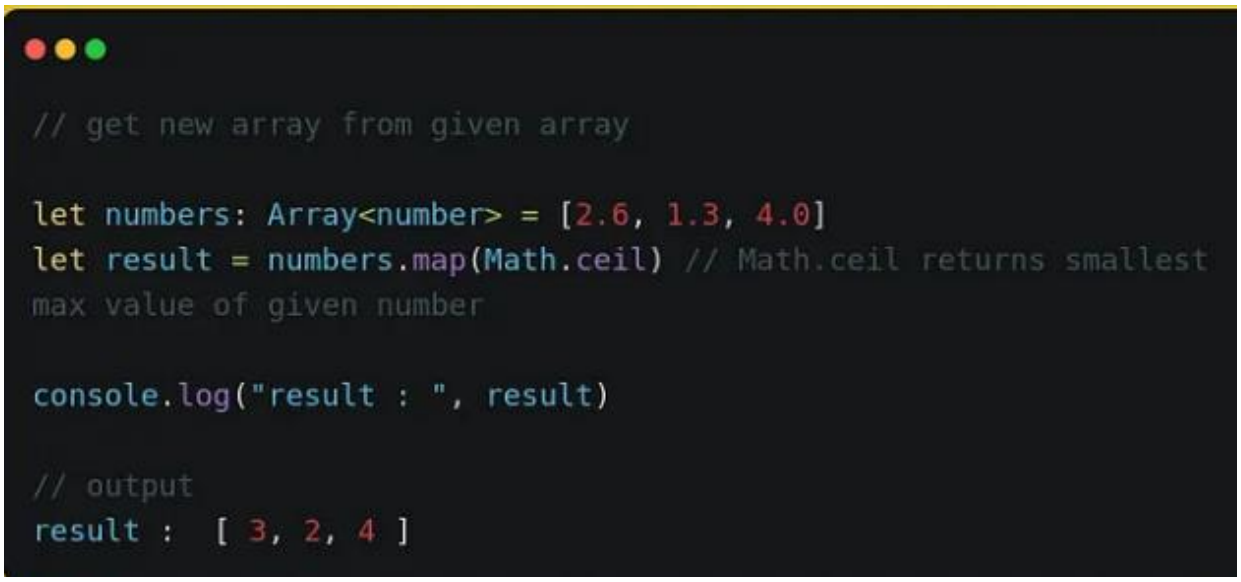
13. filter()
This method can also be used in multiple use cases. Like, such as finding even numbers from an array, finding common items from two arrays, or getting a distinct array.

Basically, it checks the conditions which are provided and returns a filtered array.
syntax: array.filter(callback)



```
// filter an array with specific condition

let emojis_1:string[] = ['😊', '😍', '😂', '😎', '😇', '🤩']
let emojis_2:string[] = ['😈', '😎', '🤓', '🤓', '😈', '😇', '😎',
'😈', '🤗']

// find common items from two arrays
let result = emojis_1.filter(f => emojis_2.indexOf(f) !== -1)
console.log("common emojis between two arrays : ", result);

// remove duplicate items from an array
let distinctArr = emojis_2.filter((item, index) =>
emojis_2.indexOf(item) === index)

console.log("duplicate items removed array : ", distinctArr)

// output:
common emojis between two arrays :  [ '😎', '😇' ]
duplicate items removed array :  [ '😈', '😎', '🤓', '😇', '🤗' ]
```

14. map()
This method creates a new array with the results of calling a provided function on every element in this array.
In the example, we've invoked map() with Math.ceil which returns the lowest maximum number.
syntax: array.map(callback)

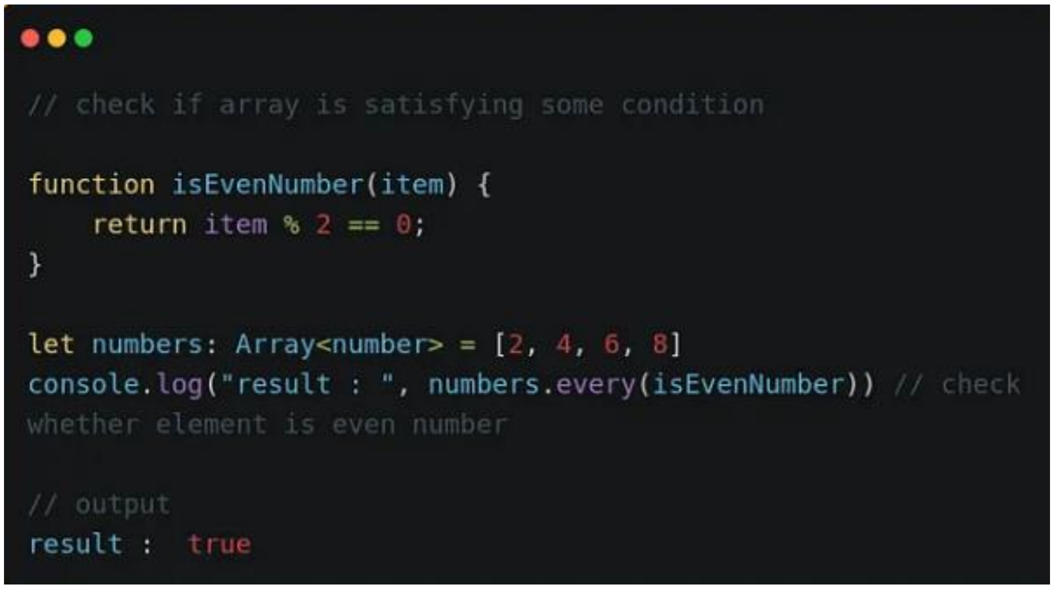


```
// get new array from given array

let numbers: Array<number> = [2.6, 1.3, 4.0]
let result = numbers.map(Math.ceil) // Math.ceil returns smallest
max value of given number

console.log("result : ", result)

// output
result :  [ 3, 2, 4 ]
```

15. every()
This method tests whether all the elements in an array pass the test implemented by the provided function.

In the example, we have checked for even numbers.

syntax: array.every(callback)

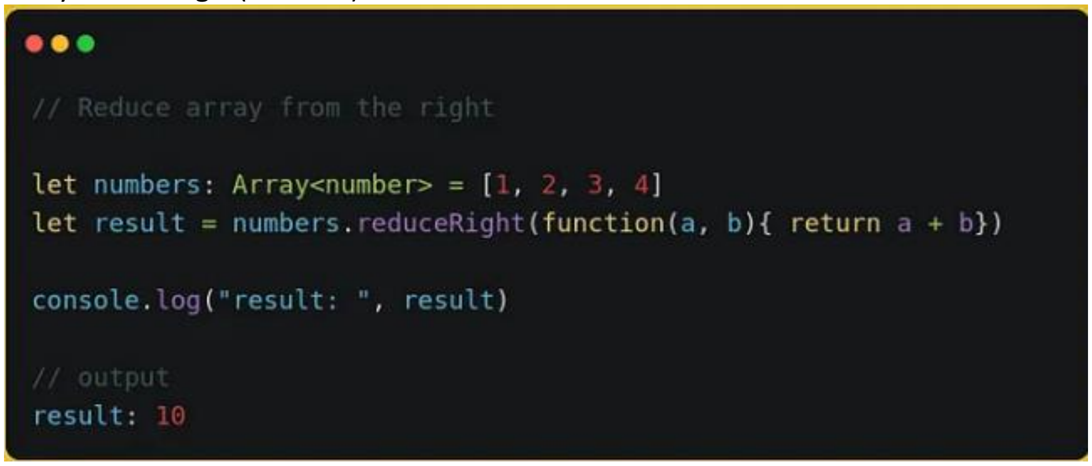


```
// check if array is satisfying some condition

function isEvenNumber(item) {
    return item % 2 == 0;
}

let numbers: Array<number> = [2, 4, 6, 8]
console.log("result : ", numbers.every(isEvenNumber)) // check
whether element is even number

// output
result :  true
```

16. reduceRight()
This method applies a function simultaneously against two values of the array (from right to left) to reduce it to a single value.

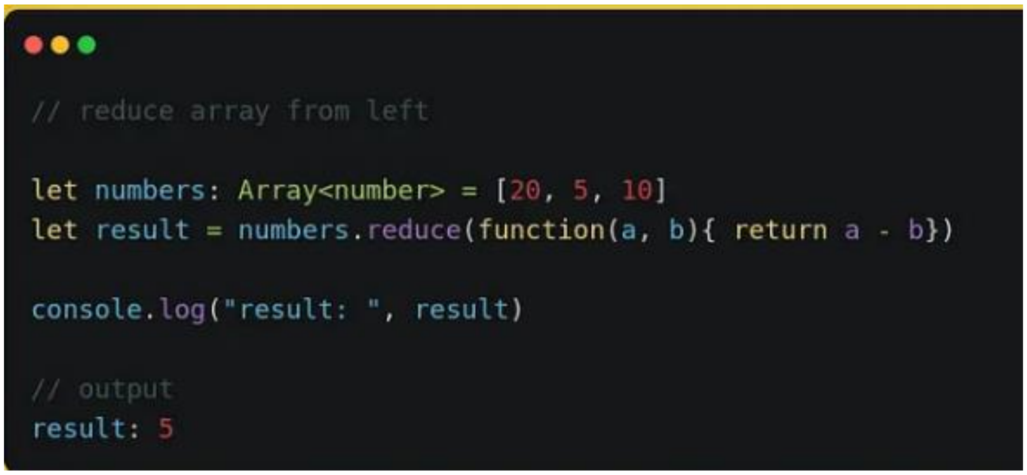
In the example, the array is reduced with the addition of an element to the previous one(right to left).

syntax:
array.reduceRight(callback)



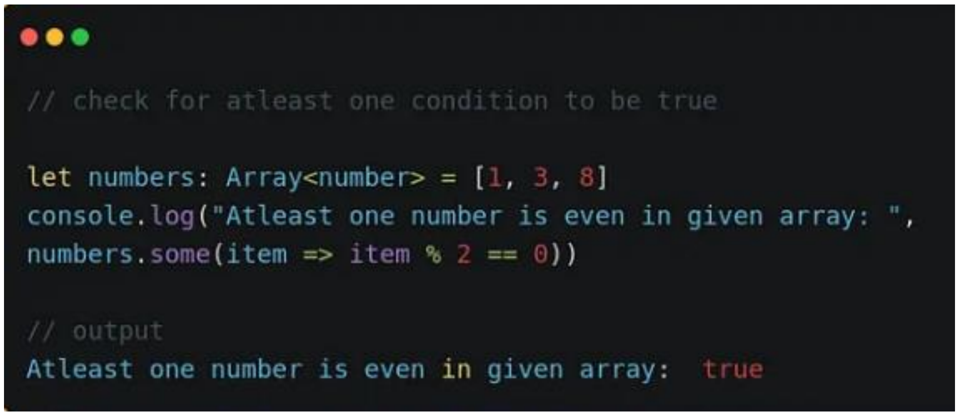
```
// Reduce array from the right

let numbers: Array<number> = [1, 2, 3, 4]
let result = numbers.reduceRight(function(a, b){ return a + b})

console.log("result: ", result)

// output
result: 10
```

17. reduce()
This method behaves the exact opposite of the reduceRight() method.
It applies a function simultaneously against two values of the array (from left to right) to reduce it to a single value. In the example, an array is reduced with the subtraction of an element from the previous one(left to right).
syntax: array.reduce(callback)



```
// reduce array from left

let numbers: Array<number> = [20, 5, 10]
let result = numbers.reduce(function(a, b){ return a - b})

console.log("result: ", result)

// output
result: 5
```

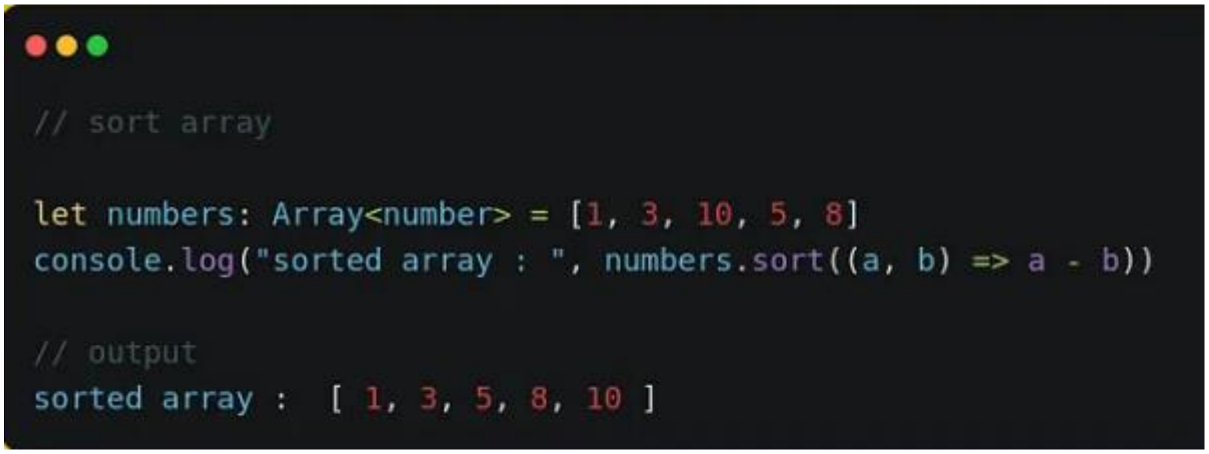
18. some()
This method is generally used for testing purposes.
i.e. To know whether at least a single item from an array is fulfilling a given condition or not.
In the example, again we've checked for at least a single even number present in an array.
syntax: array.some(callback)



```
// check for atleast one condition to be true

let numbers: Array<number> = [1, 3, 8]
console.log("Atleast one number is even in given array: ",
numbers.some(item => item % 2 == 0))

// output
Atleast one number is even in given array:  true
```

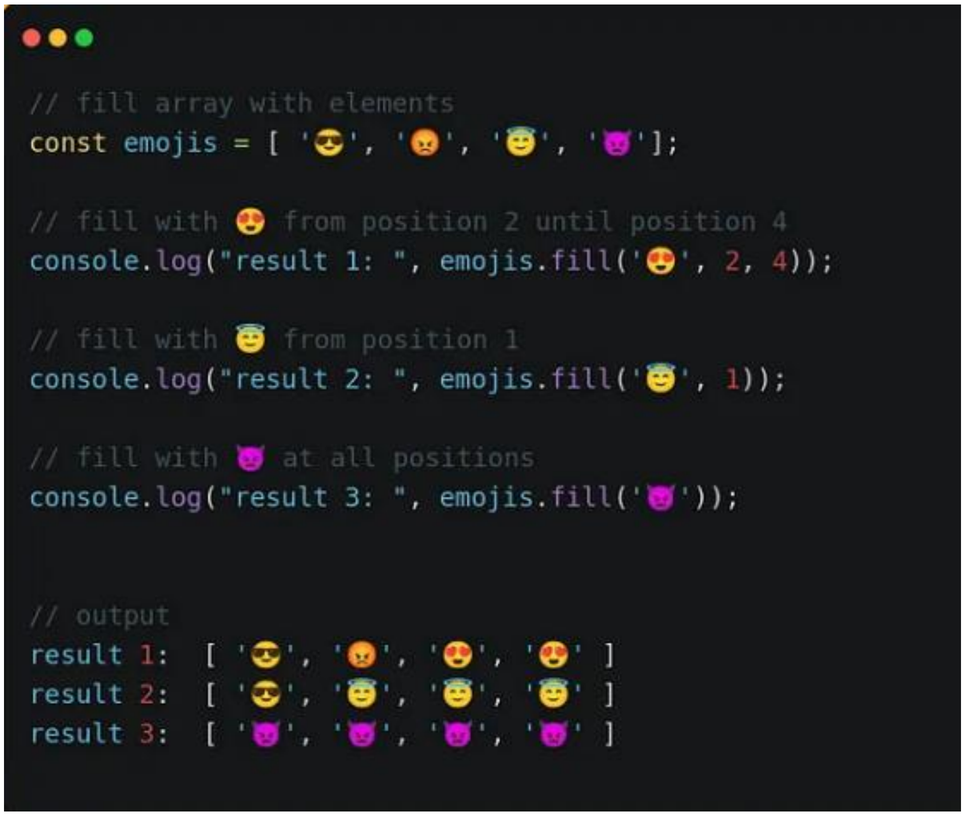
19. sort()
As the name suggests, this method arranges array elements in sorting orders.

In the example, we've sorted an array in ascending order. It will sort in descending order, with the condition b-a instead of a-b . similarly, we do in js.
syntax: array.sort(callback)



```
// sort array

let numbers: Array<number> = [1, 3, 10, 5, 8]
console.log("sorted array : ", numbers.sort((a, b) => a - b))

// output
sorted array :  [ 1, 3, 5, 8, 10 ]
```

20. fill()
This method changes all elements in an array to a static value, from a start index (default 0) to an end index (default array.length) and returns the modified array.
a. It can add new elements to specific(multiple) positions
syntax: array.fill(value, start_index, end_index)



```
// fill array with elements
const emojis = [ '😎', '😡', '😇', '😈'];

// fill with 😍 from position 2 until position 4
console.log("result 1: ", emojis.fill('😍', 2, 4));

// fill with 😇 from position 1
console.log("result 2: ", emojis.fill('😇', 1));

// fill with 😈 at all positions
console.log("result 3: ", emojis.fill('😈'));


// output
result 1:  [ '😎', '😡', '😍', '😍' ]
result 2:  [ '😎', '😇', '😇', '😇' ]
result 3:  [ '😈', '😈', '😈', '😈' ]
```