Array important methods and properties to remember -:

1. **.push**() - Used to insert a new element at the end of the array
2. **.pop**() - Used to remove the last ek
3. **.length** - Used to calc the number of elements of the array.
4. **.unshift**() - Used to remove the first element of the array.
5. **.slice**(**start_index, end_index**) - Used to create a slice (or copy) of the range of array elements and return it. The original array is untouched by this method.

The end_index in the slice method is not included. For example :

```
const arr = [1,2,3,4,5,6,7]
const new_arr = arr.slice(2,5); // This gives [3,4,5].
```

This means if you wish to create a slice of the array from the 2nd to the 5th index, you slice range should be arr.slice(2,6);

6. **.splice**() - Used to delete and add new items in the original array. The splice method mutates the original array and does not return a new array. The syntax for splice() is as follows -:

.splice(**start_index, no_of_elements_to_be_removed, el1,el2,….**)

Here, start_index is the index where the splicing should begin. No_of_elements_to_be_removed is the no of elements you want to remove. El1, el2, el3 are optional elements which can be added in place of removed elements.

7. **.reverse**() - Used to reverse an array

8. **.map**(**callback_fn**) - Used to run the callback fn for each element of the array once and return a new array. The map method can be used to create a mutated copy of the original array based on the conditions provided in the callback function.

```
const arr = [1,2,3,4,5,6]
const newArr = arr.map((element) => {
    return element * 2
});
console.log(newArr); // This will give [2,4,6,8,10,12];
```

Note - The original array is left untouched

9. **.forEach**(**callback_fn**) - Used to run a callback function once for every element of the array. Unlike the map method, forEach does not return a new array.

10. **.filter**(**callback_fn**) - Used to run a callback function and return a new array which contains filtered elements from the original array based on the truthiness of the return expression. The original array is left untouched in this method.

```
const arr = [1,2,3,4,5];
const filteredArr = arr.filter((el) => {
    return el % 2 === 0;
})
console.log(filteredArr); // [2,4]
```

11. **.sort**() - Used to sort an array in either ascending or descending order. The sort method in array is unique and used in place sorting to sort the elements based on the return value of the callback function. The original array is modified.

If the return value is > 0, it sorts in ascending order
If the return value is < 0, it sorts in descending order.

```
const arr = [45,67,3,2,1]
arr.sort((a,b) => a-b); // This will sort the array as [1,2,3,45,67]
arr.sort((b,a) => b-a); // This will sort the array as [67,45,3,2,1]
```

Basically, the comparator function (the callback function of the sort) finds the difference between two consecutive elements of the array and returns it. Depending on the difference (+ve, -ve or 0), the sorting takes place.

12. **.reduce**() - Used to reduce the array into a single value and return that new value. The original array is left untouched.

Reduce takes in a callback function which has two arguments inside of it.

.reduce((acc,val) => {}, initial acc val)

Here, acc stands for accumulator. It is the single value which compounds/changes on every iteration of the array elements.

Val is the current array element. Depending on the logic passed inside the callback function, the accumulator accumulates the value with every value of Val, starting its default value from the initial acc val.

```
const arr = [1,2,3,4]
const sum = arr.reduce((acc,val) => acc + val, 0);
console.log(sum); // 1+2+3+4 = 10
```

13. **.find**(**callback_fn**) - Used to find and return the element from a list of array elements. The criteria for finding the element in the array is defined in the callback function.

14. **findIndex**(**callback_fn**) - Used to return the first index of the found element from the list of array elements. If there are multiple instances found for a particular element, it only returns the first instance it is able to find. If not found, it returns a negative number

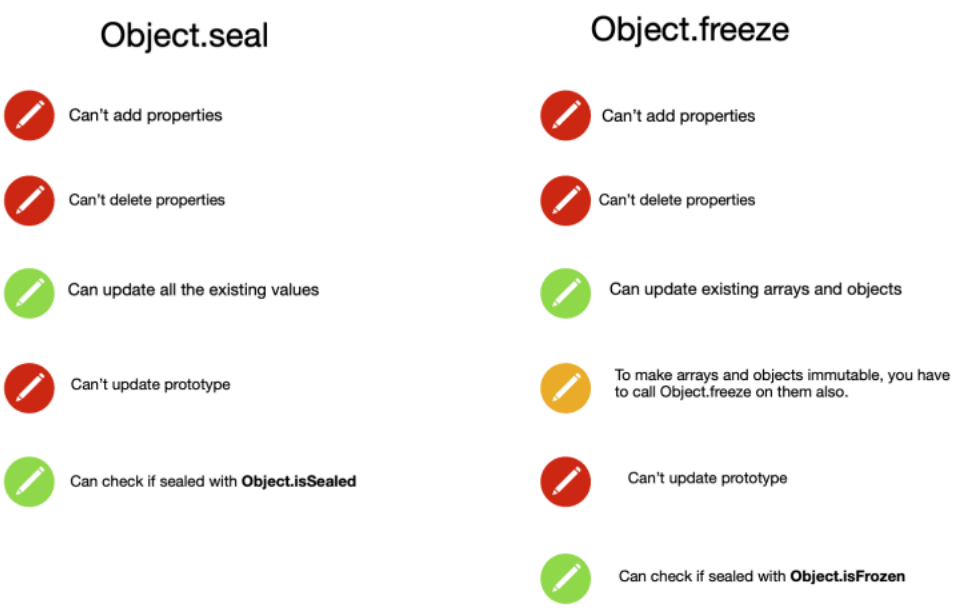
15. **indexOf**(**element**) - Used to find the index of the element from the array. if not found, it returns -1.

16. **includes**(**element**) - Used to return a Boolean whether the element is present inside the array or not.

Some other not so important array methods which might be useful

1. Array.is(element) - Used to check whether element is of type array or not.
2. Array.from(element) - Used to convert the element into an array.
3. Array.flat(level_of_flattening) - Used to convert a nested array into a singly nested array.

Check out the article : https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/flat



4. .join(token) - Used to convert an array of elements back into a string depending on the token passed as argument. Often used with the String function .split(token).

Important object methods and properties

1. **.keys**() - Returns an array of all keys as strings of an object

2. **.values**() - Returns an array of all values as strings of an object
3. **.entries**() - Returns an array of arrays, where each array holds a key-value pair.
4. **Object.create**(**some_existing_object**) - Used to create a new object using the existing object's prototype.
5. **Object.assign**(**target, source**) - Used to copy properties from one source to the target object.

There are essentially 2 ways to access values of a certain property :

1. Dot notation (obj.keyname)
2. Square bracket notation (obj[keyname])

Dot notations can be used when we know the name of the key we are accessing. If the name of the key is dynamic, then dot notation fails

For accessing a value of an object based off of a dynamic key, we use the bracket notation.

```
const keyName = "name"
const obj = {age: 20, name: 'Kishan'}
obj[keyName]; // Kishan
obj.keyName; // Reference Error : Key by the name keyName not found.
```

For deleting a key value pair from the object, we can use the **delete** keyword.

Once thing to remember for both arrays and objects is that, they are stored in the memory as references and can be accessed by the virtue of call by reference.