**Collections**

**Arrays**

**Arrays are list-like objects** whose prototype has methods to perform traversal and mutation operations. **Neither the length of a JavaScript array nor the types of its elements are fixed**. Since an array's length can change at any time, and data can be stored at non-contiguous locations in the array but if these features are not desirable for your particular use, you might consider using typed arrays

Arrays cannot use strings as element indexes (as in an [associative array](https://en.wikipedia.org/wiki/Associative_array) or dictionary) but must use integers

**Basic Array operations**

**Create, access, find index, copy, loop, add and remove items by index (or first or last)**

**for-of and Arrays – ES6**

The for...of statement **creates a loop iterating over iterable objects**, including: built-in String, Array, array-like objects (e.g., arguments or NodeList), TypedArray, Map, Set, and user-defined iterables

**Multidimensional Arrays**

JavaScript **does not provide the multidimensional array natively**. However, you can create a multidimensional array by defining an array of elements, where each element is also another array.

For this reason, we can say that a JavaScript multidimensional array is an array of arrays

**It has same functions as normal arrays**

**Adding and removing elements**

Unshift, push, shift, pop, splice

**.find(), .map(), .filter(), etc. .flatMap().sort() others?**

**Find()**

The find() method **returns the value of the first element** in the provided array **that satisfies the provided testing function**. If no values satisfy the testing function, undefined is returned.

**Map()**

The map() method **creates a new array populated with the results of calling a provided functio**n on every element in the calling array

**Filter()**

The filter() method **creates a new array** with all elements **that pass the test** implemented by the provided function.

**Every()**

The every() method **tests whether** **all elements in the array pass the test** implemented by the provided function (compare with find)

**FlatMap()**

The flatMap() method **returns a new array formed by applying a given callback function to each element of the array, and then flattening the result by one level**. It is identical to **a map() followed by a flat()** of depth 1, but slightly more efficient than calling those two methods separately

**Sort()**

The sort() method sorts the elements of an array in place and **returns the sorted array**. The default sort order is ascending

More methods in [Array - JavaScript | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)

**Typed Arrays**

A ***TypedArray*** object describes an array-like view of an underlying [binary data buffer](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer). **There is no global property named TypedArray, nor** is there a directly visible TypedArray **constructor**. Instead, there are a number of different global properties, **whose values are typed array constructors for specific element types**

[TypedArray - JavaScript | MDN (mozilla.org)](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/TypedArray)

**Maps**

The Map object **holds key-value pairs** and remembers the original insertion order of the keys. **Any value** (both objects and primitive values) **may be used as either a key or a value**

**WeakMap**

**What is?**

The WeakMap object is a **collection of key/value pairs in which the keys are weakly referenced.** The keys must be objects and the values can be arbitrary values.

Keys of WeakMaps are of the type Object only. **Primitive data types as keys are not allowed**

**Set**

The Set object lets you **store unique values of any type**, whether primitive values or object references.

Set objects are **collections of values**. You can iterate through the elements of a set in insertion order. **A value in the Set may only occur once**; it is unique in the Set's collection.

**WeakSets**

The WeakSet object lets you store weakly held objects in a collection.

WeakSet objects are **collections of objects**. Just as with Sets, **each object in a WeakSet may occur only once**; all objects in a WeakSet's collection are unique

**Synchronous generators**

We are used to **waiting for the completion of a subroutine before being able to continue with the program**. In other words, **each function will run until the end of its body and no other code is able to interfere, running in between**

**Generators break this rule, allowing us to pause their execution** and enabling a straightforward messages system to pass information in both directions. Functions marked with the async keyword, thanks to the await superpower, are able to not follow the rule too because they are a direct consequence of generators

A generator declaration is pretty much the same as a function declaration plus the addition of a star between the function keyword and the function name:

function \* generator(...args) {

yield;

}

The big syntax difference is obviously the possibility to use a new keyword inside the body of the generator: **the yield keyword. It returns the control to the caller suspending the execution of the generator**. The generator will literally stop on each encountered yield waiting a signal to resume.

Like with the return keyword, you are able to return a value each time you use the yield keyword.

[JavaScript Iterators and Generators: Synchronous Generators · andrea simone costa](https://andreasimonecosta.dev/posts/javascript-iterators-and-generators-synchronous-generators/)