Radwa Hassan Sadek
Open Source

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

Example:

```
const array1 = [1, 4, 9, 16];
const map1 = array1.map(x => x * 2);
console.log(map1);

// Expected output: Array [2, 8, 18, 32]
```

Return value

A new array with each element being the result of the callback function.

Description

The map() method is an iterative method. It calls a provided callbackFn function once for each element in an array and constructs a new array from the results.

callbackFn is invoked only for array indexes which have assigned values. It is not invoked for empty slots in sparse arrays.

The map() method is a copying method. It does not alter this. However, the function provided as callbackFn can mutate the array. Note, however, that the length of the array is saved *before* the first invocation of callbackFn. Therefore:

- callbackFn will not visit any elements added beyond the array's initial length when the call to map() began.
- Changes to already-visited indexes do not cause callbackFn to be invoked on them again.
- If an existing, yet-unvisited element of the array is changed by callbackFn, its value passed to the callbackFn will be the value at the time that element gets visited. Deleted elements are not visited.

The **forEach()** method executes a provided function once for each array element.

Example:

```
const array1 = ['a', 'b', 'c'];
array1.forEach(element => console.log(element));

 // Expected output: "a"
                     "b"
                     "c"
```

Return value:
undefined.

Description:
The forEach() method is an iterative method. It calls a provided callbackFn function once for each element in an array in ascending-index order. Unlike map(), forEach() always returns undefined and is not chainable. The typical use case is to execute side effects at the end of a chain.

callbackFn is invoked only for array indexes which have assigned values. It is not invoked for empty slots in sparse arrays.

forEach() does not mutate the array on which it is called, but the function provided as callbackFn can. Note, however, that the length of the array is saved *before* the first invocation of callbackFn. Therefore:

- callbackFn will not visit any elements added beyond the array's initial length when the call to forEach() began.
- Changes to already-visited indexes do not cause callbackFn to be invoked on them again.
- If an existing, yet-unvisited element of the array is changed by callbackFn, its value passed to the callbackFn will be the value at the time that element gets visited. Deleted elements are not visited.