ForEach;
Map;
Filter;
Reduce;

ForEach

- Is an array method. Can also be used on a NodeList.
- Executes a function once for each array element.
- Can be used as an alternative to loops in some situations.
- Can be used with regular anonymous functions or arrow functions.
- Doesn’t return a new array.
- The forEach method can’t be stopped normally when it’s running: it’s meant to use in situations where you need to perform a function on all array elements.

```
const arr1 = ["a", "b", "c"];
arr1.forEach((element) => console.log(element));
```

Using forEach

```
const array = ["chair", "sofa", "table"];

array.forEach(function(element, index) {
    console.log(`Element at index ${index}: ${element}`);
});
```

```
Element at index 0: chair
Element at index 1: sofa
Element at index 2: table
```

Example of a NodeList

- A NodeList is an array-like iterable. It isn't an array.
- A NodeList can be converted into an array.

```
<ul>
    <li class="listItem">1</li>
    <li class="listItem">2</li>
    <li class="listItem">3</li>
</ul>
```

```
let elements = document.querySelectorAll(".listItem");
console.log(elements);
```

```
forEach.js:11
NodeList(3) [li.listItem, li.listItem, li.listItem]
  i
  ▶ 0: li.listItem
  ▶ 1: li.listItem
  ▶ 2: li.listItem
    length: 3
  ▶ [[Prototype]]: NodeList
```

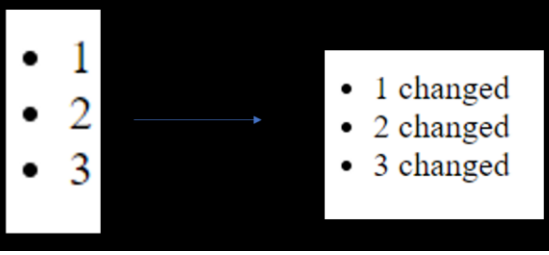
Converting a NodeList into an array, copying

- You can use Array.from() which creates a new shallow-copied array instance from an iterable or array-like object.
- A shallow copy is a copy of an object whose properties share the same references (point to the same actual values) as the source object from which the copy was made. It's still linked to the original object. If you modify the shallow copy, the original object gets changed too.
- A deep copy is a real, separate copy that doesn't have the same references to the original values in the object it's made from. If you change the deep copy, the original copy doesn't get changed.

Using forEach on a NodeList

```
let elements = document.querySelectorAll(".listItem");

elements.forEach((element) => {
    console.log(element);
    element.innerHTML=element.innerHTML + " changed";
})
```



Map

- Creates a new array by calling a function on all the elements of an array.
- Map doesn't execute the function on empty elements.
- It's non-mutating, meaning that it doesn't change the original array.
- Map can be used for adjusting array data values. It can also be used for converting data formats (like changing string values into numbers).
- Map is important in React.
- Filter, map and reduce are part of ES5, they were added in 2011.
- Filter, map and reduce are useful for reducing complexity.

Map example

```
let numbersArr = [1,2,3,4,5];

let squaredNumbers = numbersArr.map(number => number * number);

console.log(squaredNumbers);
```

```
[ 1, 4, 9, 16, 25 ]
```

Converting array values using map

```
let stringNumbers = ["1", "2", "3"];

let numbersFromStringValues = stringNumbers.map(string => Number(string));

console.log(numbersFromStringValues);
console.log(typeof numbersFromStringValues[0]);
```

```
[ 1, 2, 3 ]
number
```

Filter

- Is an essential method for array manipulation.
- Allows creating a new array from array elements that pass a test provided by a function.
- Is non-mutating, returns a new array.
- Accepts only elements that return truthy from the test function into the new array.
- Always produces the same result, it isn't random.

Basic filter use case

```
const numbers = [1,2,3,4,5,6];

const evenNumbers = numbers.filter(number => number % 2 === 0);
console.log(evenNumbers);
```

```
[ 2, 4, 6 ]
```

Array Reduce

- Executes a reducer function on each array element, resulting in a single output value.
- You can think of it as turning many values into one value.
- The single value can be of any type, including Array.
- Reduce executes a reducer function you provide on each element of the array, in order, passing the return value from the calculation on the next element. The final result is a single value.
- Can be used for summing numbers, transforming data structures, contatenating strings…
- Execution order is from left to right.

Summing array elements using reduce

```
let anotherArray = [5,4,3,2,1];

let sum = anotherArray.reduce((total, item) => total + item);
console.log(sum);
```

```
15
```

Multidimensional array

- Is essentially an array of arrays. Arrays inside an array.
- Is a grid-like structure.
- Is created by using 2 or more one-dimensional arrays.

```
let salaries = [ // multi-dimesional array
    ["ABC", 25, 15000],
    ["DEF", 32, 16000],
    ["GHJ", 55, 22000],
];
```

```
console.log(salaries[0][1]);
```

```
25
```

Flattening an array

- Flattening an array means reducing a multi-dimensional array into a single-dimenstional array.

```
const arrays = [[1,2,3], [4,5], [6]];
const flattenedArrays = arrays.reduce((accumulator, currentValue) => accumulator.concat(currentValue), []);
console.log(flattenedArrays);
```

```
[ 1, 2, 3, 4, 5, 6 ]
```

Keywords

- Map
- Filter
- Reduce
- Multidimensional array
- Flatten
- NodeList
- Array.from()
- forEach
- Mutation, mutable, immutable
- Shallow copy
- Deep copy