Array (Mảng)

1. Tổng quan về array
2. Array object - Props & Methods
3. Array destructuring
4. Clone mảng
5. Duyệt mảng
6. Thêm xoá phần tử
7. Kiểm tra tồn tại của phần tử
8. Tìm kiếm một phần tử
9. Transform mảng
10. Lọc phần tử theo điều kiện
11. Sắp xếp mảng
12. reduce
13. Bài tập thực hành

1. Tổng quan về array

Khai báo mảng

- Đặt tên nên dùng suffix là List. Eg: dùng numberList thay vì numbers
- Mỗi phần tử có thể có một kiểu dữ liệu khác nhau.

```
// khai báo mảng rỗng
const numberList = [];
const numberList = [1, 2, 3, 4]; // a list of numbers
const wordList = ['Long', 'Do']; // a list of strings
const flagList = [true, false, false]; // a list of boolean
// a list of students
const studentList = [
 { id: 1, name: 'Hoang' },
 { id: 2, name: 'Minh' },
 { id: 3, name: 'Khanh' },
 { id: 3, name: 'Thanh' },
]
```

```
// a list of list
const board = [
 [1, 2],
 ['a', 'b', 'c'],
 [true, false, false, false]
]
// a list of mixed data type
const mixedList = [
 1,
 2,
```

```
  'word',
  true,
  null,
  undefined,
  { id: 1, name: 'Long' },
  [1, 2, 3]
];
```

Truy xuất phần tử của mảng

- Dùng square brackets để truy xuất phần tử tại vị trí index.
- Index của mảng bắt đầu từ 0
- Nếu length của array bằng 3, thì index lớn nhất là length - 1 = 2

```
const numberList = [3, 5, 7]; // recommended
numberList[0]; // 3
numberList[1]; // 5
numberList[2]; // 7
numberList.length; // 3
numberList[numberList.length - 1]; // 7 (the last element)
```

Mảng 2 chiều

Trong JavaScript, bạn có thể tạo mảng 2 chiều bằng cách sử dụng mảng lồng nhau, tức là mỗi phần tử trong mảng chính lại là một mảng con.

```
// Khởi tạo mảng 2 chiều 3x3
var array2D = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

// Truy cập phần tử trong mảng 2 chiều
console.log(array2D[0][0]); // Output: 1
console.log(array2D[1][2]); // Output: 6
console.log(array2D[2][1]); // Output: 8

// Cập nhật giá trị của phần tử
array2D[1][1] = 10;
console.log(array2D[1][1]); // Output: 10
```

2. Giới thiệu Array object

Static methods

#	Name	Desc

#	Name	Desc
1	Array.isArray(arr)	Kiểm tra arr có phải là mảng không?
2	Array.from()	Tạo mảng mới từ các dữ liệu khác như Set, Iterable, ...

Instance props

```
const numberList = [1, 2, 3];
numberList.length; // 3
```

Instance methods

Nhóm hàm kiểm tra phần tử có tồn tại không?

#	Name	Desc
1	every(callbackFn)	Kiểm tra tất cả phần tử thoả điều kiện
2	some(callbackFn)	Kiểm tra có một phần tử thoả điều kiện
3	indexOf(searchElement)	Tìm vị trí đầu tiên của phần tử searchElement
4	lastIndexOf(searchElement)	Tìm vị trí cuối cùng của phần tử searchElement
5	includes(searchElement)	Kiểm tra có chứa phần tử searchElement không
6	find(callbackFn)	Tìm phần tử đầu tiên thoả điều kiện
7	findIndex(callbackFn)	Tìm vị trí của phần tử đầu tiên thoả điều kiện

Nhóm hàm thêm xoá phần tử

#	Name	Desc
1	push(element0, ..., elementN)	Thêm cuối mảng
2	pop()	Xoá cuối mảng
3	shift()	Xoá đầu mảng
4	unshift(element0, ..., elementN)	Thêm đầu mảng
5	splice(start, deleteCount, item1, ..., itemN)	Xoá/thêm giữa mảng

Nhóm hàm hay sử dụng

#	Name	Desc
1	forEach(callbackFn)	Duyệt mảng
2	map(callbackFn)	Biến đổi mảng
3	filter(callbackFn)	Lọc mảng theo điều kiện

#	Name	Desc
4	slice(start, end)	Lấy mảng con
5	reduce()	Duyệt mảng và tính toán cho ra kết quả cuối cùng

Và một số hàm khác

#	Name	Desc
1	fill(value, start = 0, end = arr.length)	Fill value từ start tới end
2	join()	Biến đổi mảng thành chuỗi
3	concat()	Nối mảng
4	reverse()	Đảo ngược mảng
5	sort()	Sắp xếp mảng

```
Array.isArray(123); // false
Array.isArray('Long Do'); // false
Array.isArray(true); // false
Array.isArray([]); // true
Array.isArray([1, 2, 3]); // true
[null, undefined].fill(false); // [false, false]
Array(5).fill(1); // [1, 1, 1, 1, 1]
['long', 'do'].join('-'); // 'long-do'
[1, 2, 3].reserve(); // [3, 2, 1]
```

> Tham khảo: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

3. Array Destructuring

```
// object destructuring
const student = {
 id: 1,
 name: 'Long Do',
}
const { id, name } = student;
console.log(id); // 1
console.log(name); // 'Long Do'
```

```
const numberList = [3, 5, 7];
// old way
const first = numberList[0]; // 3
const second = numberList[1]; // 5
const third = numberList[2]; // 7
```

```
// similar but new way
const [first, second, third] = numberList;
// or even direct array
const [first, second, third] = [3, 5, 7, 9, 11];
// rest operator (...)
const [first, second, third, ...rest] = [3, 5, 7, 9, 11];
console.log(rest); // [9, 11]
```

4. Clone array

Issue: Array cũng là tham chiếu, nên cẩn thận khi dùng phép gán.

```
const numberList = [1, 2, 3];
const anotherList = numberList;
anotherList[0] = 4; // anotherList = [4, 2, 3]
console.log(numberList); // [4, 2, 3]
```

Solution: Clone array trước khi thực hiện thay đổi.

```
const numberList = [1, 2, 3];
const anotherList = [...numberList];
anotherList[0] = 4; // anotherList = [4, 2, 3]
console.log(numberList); // [1, 2, 3]
```

5. Duyệt các phần tử của array

- Before ES5: for...i
- ES5: forEach
- ES6: for...of

```
// before ES5
const numberList = [2, 4, 6];
for (let i = 0; i < numberList.length; i++) {
 const number = numberList[i];
 console.log(number); // 2, 4, 6
}
```

```
// ES5 forEach
const numberList = [2, 4, 6];
numberList.forEach(x => console.log(x)); // 2, 4, 6
```

```
// ES6 for...of
const numberList = [2, 4, 6];
for (const number of numberList) {
 console.log(number); // 2, 4, 6
}
```

6. Thêm xoá phần tử

Nhóm hàm thêm xoá phần tử

#	Name	Desc
1	push(element0, ..., elementN)	Thêm cuối mảng
2	pop()	Xoá cuối mảng
3	shift()	Xoá đầu mảng
4	unshift(element0, ..., elementN)	Thêm đầu mảng
5	splice(start, deleteCount, item1, ..., itemN)	Xoá/thêm giữa mảng

```
// Add new items at the end
const numberList = [1, 2, 3];
numberList.push(4, 5);
console.log(numberList); // [1, 2, 3, 4, 5]
```

```
// Remove items at the end
const numberList = [1, 2, 3];
const lastNumber = numberList.pop();
console.log(numberList, lastNumber); // [1, 2], 3
```

```
// Add new items at the beginning
const numberList = [1, 2, 3];
numberList.unshift(0);
console.log(numberList); // [0, 1, 2, 3]
```

```
// Add new items at the beginning
const numberList = [1, 2, 3];
const firstNumber = numberList.shift();
console.log(numberList, firstNumber); // [2, 3], 1
```

```
// Add/remove items at the middle of an array
const numberList = [3, 5, 7];
numberList.splice(0, 0, 2, 4);
console.log(numberList);
```

```
const numberList = [1, 3, 5, 7]; // [1, 3, 2, 4, 7]
numberList.splice(2, 1, 2, 4);
console.log(numberList);

const numberList = [1, 3, 5, 7]; // [1, 3, 2, 4, 7]
numberList.splice(2, 0, 4, 4);
console.log(numberList);

const numberList = [1, 3, 5, 7];
numberList.pop();
numberList.pop();
console.log(numberList);

// [1, 2, 3]
// [1, 2, 3, empty]
// [0, 1, 2, 3]
```

> Tham khảo https://javascript.info/array#methods-pop-push-shift-unshift

7. Kiểm tra tồn tại của phần tử

Nhóm hàm kiểm tra phần tử có tồn tại không?

#	Name	Desc
1	every(callbackFn)	Kiểm tra tất cả phần tử thoả điều kiện
2	some(callbackFn)	Kiểm tra có một phần tử thoả điều kiện
3	indexOf(searchElement)	Tìm vị trí đầu tiên của phần tử searchElement
4	lastIndexOf(searchElement)	Tìm vị trí cuối cùng của phần tử searchElement
5	includes(searchElement)	Kiểm tra có chứa phần tử searchElement không

```
// check if all numbers is even
[1, 2, 4].every((x) => x % 2 === 0); // false
[2, 4, 6].every((x) => x % 2 === 0); // true
```

```
// check if exist one number is even
[1, 2, 4].some((x) => x % 2 === 0); // true
[1, 3, 5].some((x) => x % 2 === 0); // false
```

```
[1, 1, 1].indexOf(1); // 0
[1, 1, 1].lastIndexOf(1); // 2
['long', 'do', 'long'].indexOf('long'); // 0
['long', 'do', 'long'].lastIndexOf('long'); // 2
['long', 'do', 'long'].includes('long'); // true
['do'].includes('long'); // false
```

```
// every v1
function checkIfAllEven(numberList) {
 if (!Array.isArray(numberList)) return false;
 let isValid = true;
 for (let i = 0; i < numberList.length; i++) {
    const number = numberList[i];
    if (number % 2 !== 0) {
        isValid = false;
        break;
    }
 }
 return isValid;
}
console.log(checkIfAllEven([2, 1, 3]));
console.log(checkIfAllEven([2, 4, 6]));
```

```
// every v2
function checkIfAllEven(numberList) {
 if (!Array.isArray(numberList)) return false;
 for (let i = 0; i < numberList.length; i++) {
    if (numberList[i] % 2 !== 0) return false;
 }
 return true;
}
console.log(checkIfAllEven([2, 1, 3]));
console.log(checkIfAllEven([2, 4, 6]));
```

8. Tìm kiếm một phần tử với hàm find()

Nhóm hàm kiểm tra phần tử có tồn tại không?

#	Name	Desc
1	find(callbackFn)	Tìm phần tử đầu tiên thoả điều kiện
2	findIndex(callbackFn)	Tìm vị trí của phần tử đầu tiên thoả điều kiện

```
[2, 1, 3].find(x => x % 2 === 0); // 2
[2, 1, 3].findIndex(x => x % 2 === 0); // 0
['do', 'long'].find(x => x.length > 3); // 'long'
['do', 'long'].findIndex(x => x.length > 3); 1
```

> Tham khảo: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find

Callback là gì?

> A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

- Tạm dịch nôm na, callback là tham số của hàm có kiểu dữ liệu là hàm

```
function main(callbackFn) {
 // processing ...
 // do another stuff
 callbackFn()
}
function callback() {
 console.log('call me when needed')
}
main(callback)
```

```
function main(onFinish) {
 let sum = 0;
 for (let i = 0; i < 10; i++) {
    sum += i;
 }
 onFinish(sum)
}
function handleOnFinish(sum) {
 console.log('Sum is:', sum);
}
main(handleOnFinish);
```

Cài đặt hàm **find()** với for...i

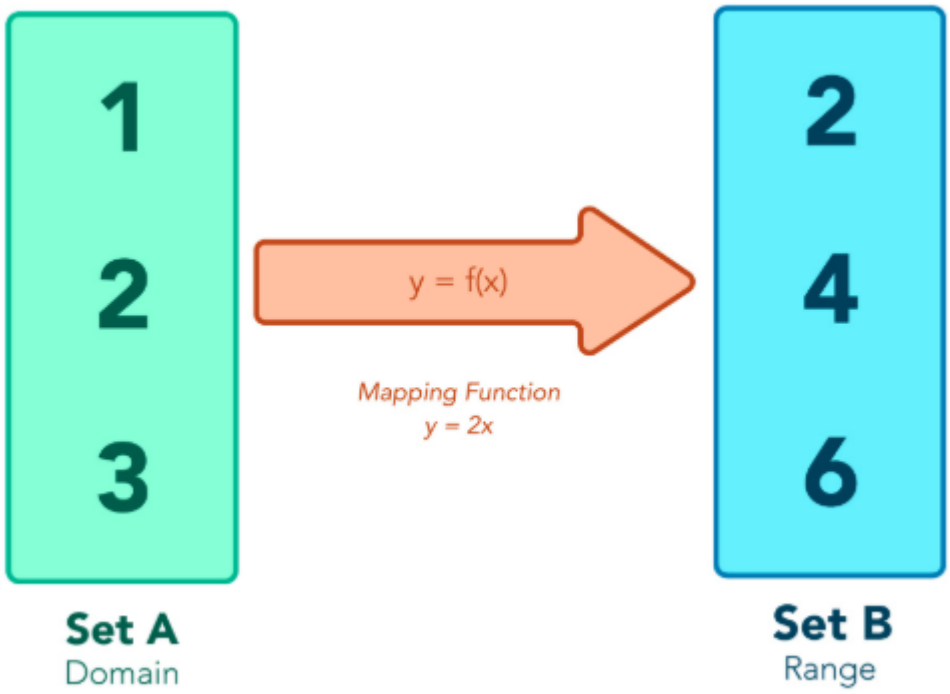
```
function find(arr, target) {
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] === target) {
            return i; // Trả về chỉ số của phần tử nếu tìm thấy
        }
    }
    return -1; // Trả về -1 nếu không tìm thấy
```

```
}

// Sử dụng hàm find
let myArray = [1, 2, 3, 4, 5];
let index = find(myArray, 3);
console.log(index); // Kết quả: 2
```

9. Transform array với hàm map()

- Dùng hàm map(transformFn) để biến đổi các phần tử này sang phần tử khác.
- Lưu ý số lượng phần tử không thay đổi.
- Điều thay đổi ở đây là mỗi phần tử sẽ bị biến đổi theo một công thức giống nhau.
- Kết quả trả về là **một mảng mới**.




```
const numberList = [1, 3, 5, 7];
numberList.map(x => x + 1); // [2, 4, 6, 8]
numberList.map(x => x * 2); // [ 2, 6, 10, 14]
```

```
const wordList = ['do', 'long'];
wordList.map(x => x.length); // [2, 4]
wordList.map(x => `name-${x}`); // ['name-Do', 'name-long']
```

> Tham khảo https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

Cài đặt hàm **map()** với for...i

```
function map(arr, callback) {
    let mappedArray = [];
    for (let i = 0; i < arr.length; i++) {
        mappedArray.push(callback(arr[i], i, arr));
    }
    return mappedArray;
}

// Sử dụng hàm map
let numbers = [1, 2, 3, 4, 5];
let squaredNumbers = map(numbers, function(num) {
    return num * num;
});
console.log(squaredNumbers); // Kết quả: [1, 4, 9, 16, 25]
```

10. Lọc phần tử theo điều kiện với hàm filter()

- Dùng hàm filter(callbackFn) để lọc mảng theo điều kiện cho trước.
- Kết quả trả về là một mảng con mới.

```
const numberList = [1, 3, 5, 2, 7];
numberList.filter(x => x % 2 === 0); // [2]
numberList.filter(x => x > 2); // [3, 5, 7]
numberList.filter(x => x <= 10 || x % 5 === 0); // [5]
```

```
const wordList = ['long', 'do', 'developer'];
wordList.filter(x => x.length < 5); // ['long']
wordList.filter(x => x.startsWith('de')); // ['long', 'developer']
```

> Tham khảo: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

Cài đặt hàm **filter()** với for...i

```
function filter(arr, callback) {
    let filteredArray = [];
    for (let i = 0; i < arr.length; i++) {
        if (callback(arr[i], i, arr)) {
            filteredArray.push(arr[i]);
        }
    }
    return filteredArray;
}
```

```
// Sử dụng hàm filter
let numbers = [1, 2, 3, 4, 5];
let evenNumbers = filter(numbers, function(num) {
    return num % 2 === 0;
});
console.log(evenNumbers); // Kết quả: [2, 4]
```

11. Sắp xếp mảng với hàm sort()

- Trong Javascript, có hỗ trợ sẵn hàm sort(compareFn) để sắp xếp mảng theo điều kiện mong muốn.
- Nếu là null / undefined thì auto được cho xuống cuối mảng, null đứng trước, tới undefined.
- Nếu hàm compareFn không được cung cấp, các phần tử sẽ được chuyển về strings để thực hiện so sánh.
- Nếu compareFn(a, b) được cung cấp thì dựa vào kết quả của hàm để xác định:
    - KQ trả về < 0, a sẽ đứng trước b
    - KQ trả về = 0, a và b như nhau
    - KQ trả về > 0, a sẽ đứng sau b
- Hàm sort() sẽ trả về mảng sau khi sort (nhưng đây là mảng hiện tại, không phải mảng mới)

```
const numberList = [2, 5, 3, 1];
numberList.sort(); // [1, 2, 3, 5]
[null, 2, 1, 5, 3, undefined, null].sort(); // [1, 2, 3, 5, null, null, undefined]
```

```
// v1
function compareFn(a, b) {
    if (a > b) return 1;
    if (a === b) return 0;
    return -1;
}
[2, 1, 3].sort(compareFn); // 1, 2, 3
```

```
// v2
function compareFn(a, b) {
    return a - b;
}
[2, 1, 3].sort(compareFn); // 1, 2, 3
```

```
// v3
function compareFn(a, b) {
    return a - b;
}
[2, 1, 3].sort((a, b) => a - b); // 1, 2, 3
```

Có nhiều thuật toán sắp xếp

1. Bubble sort
2. Selection sort
3. Merge sort
4. Quick sort
5. ...

> Tham khảo https://visualgo.net/vi/sorting

Tham khảo:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort
- https://www.tutorialspoint.com/which-algorithm-does-the-javascript-arrayhashsort-function-use
- https://stackoverflow.com/questions/234683/javascript-array-sort-implementation

12. Duyệt mảng và tính toán với hàm reduce()

Trong JavaScript, reduce() là một phương thức của mảng được sử dụng để thu gọn (reduce) một mảng thành một giá trị duy nhất. Phương thức này lặp qua từng phần tử của mảng, thực hiện một hàm gọi là reducer và tích lũy kết quả.

- Dùng hàm reduce() khi có nhu cầu duyệt qua từng phần tử và tính toán ra một kết quả cuối cùng.

```
// Arrow function
reduce((accumulator, currentValue) => {...})
reduce((accumulator, currentValue, currentIndex) => {...})
reduce((accumulator, currentValue, currentIndex, array) => {...})
reduce((accumulator, currentValue, currentIndex, array) => {...}, initialValue)

// Callback function
reduce(callbackFn)
reduce(callbackFn, initialValue) // initialValue (optional)

// Inline callback function
reduce(function callbackFn(accumulator, currentValue) {...})
reduce(function callbackFn(accumulator, currentValue, currentIndex) {...})
reduce(function callbackFn(accumulator, currentValue, currentIndex, array) {...})
reduce(function callbackFn(accumulator, currentValue, currentIndex, array) {...},
initialValue)
```

Trong đó:

- **callbackFn**: Một hàm được gọi cho mỗi phần tử trong mảng, nhận các tham số sau:
    - **accumulator**: Giá trị tích lũy từ các lần gọi trước đó của callback hoặc giá trị khởi tạo nếu có, nếu không thì giá trị của nó là **array[0]**.

    - **currentValue**: Giá trị của phần tử hiện tại. Trong lệnh gọi đầu tiên, giá trị của nó là **array[0]** nếu giá trị ban đầu được chỉ định; mặt khác giá trị của nó là **array[1]**.
    - **currentIndex** (tùy chọn): Chỉ số của phần tử hiện tại (**currentValue**) trong mảng. Trong lệnh gọi đầu tiên, giá trị của nó là **0** nếu **initialValue** được chỉ định, nếu không thì 1.
    - **array (tùy chọn)**: Mảng đang được lặp qua.
- **initialValue (tùy chọn)**: Giá trị khởi tạo cho **accumulator**. Nếu không có **initialValue**, phần tử đầu tiên của mảng sẽ được sử dụng và callback sẽ được gọi từ phần tử thứ hai trở đi. Còn nếu có **initialValue** thì callback sẽ được gọi từ phần tử đầu tiên của mảng.

Ví dụ:

```
const array = [1, 2, 3, 4];

// 0 + 1 + 2 + 3 + 4
const initialValue = 0;
const sumWithInitial = array.reduce((accumulator, currentValue) => accumulator +
currentValue, initialValue);

console.log(sumWithInitial);
// Expected output: 10
```

```
const numberList = [2, 4, 6];
let sum = 0;
for (let i = 0; i < numberList.length; i++) {
 sum += numberList[i];
}
console.log(sum); // 12

// the same with above but using reduce()
numberList.reduce((sum, number) => sum + number); // 12
```

Cài đặt hàm reduce với for...i

Cách 1:

```
// reduce(arr, callbackFn, initialValue)
// rules:
// - arr should be an array and callbackFn should be a function
// - arr.length = 0 and initialValue === undefined --> throw error
// - arr.length = 0 and initialValue !== undefined --> return initialValue
function reduce(arr, callbackFn, initialValue) {
    if (!Array.isArray(arr) || typeof callbackFn !== 'function') {
        throw new Error('Invalid parameters');
    }
```

```
    // arr is an array
    if (arr.length === 0) {
        if (initialValue === undefined) {
            throw new Error('Should have initialValue when arr is empty');
        }
            return initialValue;
    }
    const hasInitialValue = initialValue !== undefined;
    const startIndex = hasInitialValue ? 0 : 1;
    let accumulator = hasInitialValue ? initialValue : arr[0];
    for (let i = startIndex; i < arr.length; i++) {
        accumulator = callbackFn(accumulator, arr[i], i);
    }
    return accumulator;
}
```

Cách 2 (rút gọn):

```
// setup
function reduce(array, callbackFn, initialValue) {
  let accumulator = initialValue !== undefined ? initialValue : array[0];

  for (let i = initialValue !== undefined ? 0 : 1; i < array.length; i++) {
    accumulator = callbackFn(accumulator, array[i], i, array);
  }

  return accumulator;
}

// sử dụng
const array = [1, 2, 3, 4, 5];

const sum = reduce(array, (accumulator, currentValue) => accumulator +
currentValue, 0);
console.log(sum); // Output: 15
```

> Tham khảo: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce

13. Bài tập thực hành

Bài tập mảng - Print numbers:

1. Print numbers: 1 -> 10
2. Print even numbers [2, 4, 6, 8, 10]
3. Print even numbers but less than n

Giải bài tập:

```
// 1. Print numbers: 1 -> 10
function printNumbers() {
 for (let i = 1; i <= 10; i++) {
 console.log(i);
 }
}
printNumbers();
```

```
// 2. Print even numbers [2, 4, 6, 8, 10]
function printEvenNumbers() {
 for (let i = 2; i <= 10; i++) {
 if (i % 2 === 0) console.log(i);
 }
}
printEvenNumbers();
function printEvenNumbers() {
 for (let i = 2; i <= 10; i += 2) {
 console.log(i);
 }
}
printEvenNumbers();
```

```
// 3. Print even numbers but less than n
function printEvenNumbersN(n) {
 if (n < 2) return;
 for (let i = 2; i < n; i += 2) {
 console.log(i);
 }
}
printEvenNumbersN(99);
```

Bài tập mảng - Tìm max:

- Cho 1 mảng arr = [5, 1, 7, 9, 3, 0]
- Tìm số lớn nhất trong mảng đã cho

```
function findMaxI(numberList) {
 if (!Array.isArray(numberList) || numberList.length === 0) return undefined;
 let max = numberList[0];
 for (let i = 0; i < numberList.length; i++) {
    if (numberList[i] > max) {
        max = numberList[i];
    }
 }
 return max;
```

```
}
console.log(findMaxI([2, 3, 5, 11]));
```

```
// forEach
function findMaxEach(numberList) {
    if (!Array.isArray(numberList) || numberList.length === 0) return undefined;
    let max = numberList[0];
    numberList.forEach((number) => {
        if (number > max) {
            max = number;
        }
    });
    return max;
}
console.log(findMaxEach([2, 3, 5, 11]));
```

```
// reduce
function findMaxReduce(numberList) {
    if (!Array.isArray(numberList) || numberList.length === 0) return undefined;
    let max = numberList[0];
    numberList.forEach((number) => {
        if (number > max) {
            max = number;
        }
    });

    // return numberList.reduce((max, number) => {
        // if (number > max) return number;
        // return max;
    // });

    // return numberList.reduce((max, number) => {
        // return number > max ? number : max;
    // });

    return numberList.reduce((max, number) => (number > max ? number : max));
}
console.log(findMaxReduce([2, 3, 5, 11]));
```

Bài tập mảng - Tìm từ dài nhất

```
function findLongestWordI(wordList) {
 if (!Array.isArray(wordList) || wordList.length === 0) return undefined;
 let longestWord = wordList[0];
 for (let i = 0; i < wordList.length; i++) {
    const currentWord = wordList[i];
    if (currentWord.length > longestWord.length) {
```

```
            longestWord = currentWord;
        }
     }
     return longestWord;
}
```

```
function findLongestWordEach(wordList) {
    if (!Array.isArray(wordList) || wordList.length === 0) return undefined;
    let longestWord = wordList[0];
    wordList.forEach((currentWord) => {
        if (currentWord.length > longestWord.length) {
            longestWord = currentWord;
        }
    });
    return longestWord;
}
```

```
function findLongestWordReduce(wordList) {
    if (!Array.isArray(wordList) || wordList.length === 0) return undefined;
    return wordList.reduce((longestWord, currentWord) => currentWord.length >
longestWord.length ? currentWord : longestWord);
}
const wordList = ['long', 'do'];
console.log(findLongestWordI(wordList));
console.log(findLongestWordEach(wordList));
console.log(findLongestWordReduce(wordList));
```