Mảng – Array

```
// Mảng có thể chứa nhiều kiểu dữ liệu bên trong
const cars = [{ name: 'BMW', location: 'germany' }, 'Toyota', 24]
console.log(cars[1]) // Toyota
cars[0].name = 'Mercedes'
console.log(cars) // [ { name: 'Mercedes', location: 'germany' }, 'Toyota', 24 ]
```

Một số phương thức và thuộc tính hay dùng

length: return độ dài mảng

```
const num = [1, 2, 3, 4]
num.length // 4
```

Array.isArray() hoặc **instanceof** để nhận biết một biến có phải là mảng hay không

```
const fruits = ['Banana', 'Orange', 'Apple', 'Mango']
Array.isArray(fruits) // true
fruits instanceof Array // true
```

toString() hoặc **join()** để chuyển mảng sang chuỗi

```
const fruits = ['Banana', 'Orange', 'Apple', 'Mango']
const str1 = fruits.toString() // Banana,Orange,Apple,Mango
const str2 = fruits.join('-') // Banana-Orange-Apple-Mango
```

push(): thêm 1 phần từ vào cuối mảng, return lại chiều dài mảng mới

```
const fruits = ['Banana', 'Orange', 'Apple']
const x = fruits.push('Mango') // giá trị của x là 4
```

pop(): xóa phần tử cuối cùng của mảng, return lại phần tử vừa xóa

```
const fruits = ['Banana', 'Orange', 'Apple', 'Mango']
const x = fruits.pop() // giá trị của x là Mango
```

shift(): xóa phần tử đầu tiên của mảng, return lại phần tử vừa xóa

```
const fruits = ['Banana', 'Orange', 'Apple', 'Mango']
const x = fruits.shift() // giá trị của x là Banana
```

unshift(): thêm 1 phần tử vào đầu mảng, return lại chiều dài mảng mới

```
const fruits = ['Orange', 'Apple', 'Mango']
const x = fruits.unshift('Banana') // giá trị của x là 4
```

Lưu ý khi dùng **delete**, phần tử sẽ bị mất khỏi mảng nhưng để lại 1 khoảng trống. Khi truy cập đến khoảng trống này thì giá trị của nó là **undefined**

```
const fruits = ['Banana', 'Orange', 'Apple', 'Mango']
delete fruits[0]
console.log(fruits) // [ <1 empty item>, 'Orange', 'Apple', 'Mango' ]
console.log(fruits.length) // 4
```

splice(vị trí thêm, số lượng cần xóa, ... phần tử thêm): Hàm **splice** dùng để thêm hoặc xóa nhiều phần tử trong 1 mảng. Return về mảng với những phần tử vừa được xóa

```
const fruits = ['Banana', 'Orange', 'Apple', 'Mango']
// Thêm vào vị trí số 2
const x = fruits.splice(2, 0, 'Lemon', 'Kiwi')
console.log(x) // [] vì không xóa phần tử nào mà chỉ thêm
console.log(fruits) // [ 'Banana', 'Orange', 'Lemon', 'Kiwi', 'Apple', 'Mango' ]
```

```
const fruits = ['Banana', 'Orange', 'Apple', 'Mango']
// Xóa 1 phần tử tại vị trí số 0
const x = fruits.splice(0, 1)
console.log(x) // [ 'Banana' ]
console.log(fruits) // [ 'Orange', 'Apple', 'Mango' ]
```

slice(vị trí bắt đầu, vị trí kết thúc): tách ra một mảng mới từ mảng cũ

```
const fruits = ['Banana', 'Orange', 'Apple', 'Mango']
// tách ra 1 mảng mới bắt đầu tại vị trí đầu tiên đến vị trí cuối
const newFruits = fruits.slice(1)
console.log(newFruits) // [ 'Orange', 'Apple', 'Mango' ]
console.log(fruits) // [ 'Banana', 'Orange', 'Apple', 'Mango' ]
// tách ra 1 mảng mới bắt đầu tại vị trí 1 đến 2 (3-1)
const newFruits2 = fruits.slice(1, 3)
console.log(newFruits2) //[ 'Orange', 'Apple' ]
```

concat(): Tạo mới một mảng bằng cách nối các mảng cũ

```
const myGirls = ['Cecilie', 'Lone']
const myBoys = ['Emil', 'Tobias', 'Linus']
const myChildren = myGirls.concat(myBoys) // [ 'Cecilie', 'Lone', 'Emil',
'Tobias', 'Linus' ]
```

spread operator : Phân rã mảng (object) thành từng phần tử nhỏ (tưởng tượng [1,2,3] => 1,2,3)

```
const cars1 = [1, 2, 3]
const cars2 = [3, 4, 5]
// Nối mảng
const newCars = [...cars1, ...cars2] // [ 1, 2, 3, 3, 4, 5 ]
// Tạo thành 1 mảng mới
const cars3 = [...cars1] // [1, 2, 3]
console.log(cars1 !== cars3) // true
```

forEach(): Lặp qua từng phần tử trong mảng

tham số là một **callback function** với 3 đối số:

- giá trị phần tử
- index phần tử
- mảng đang thực hiện

```
const numbers = [1, 2, 3, 4, 5, 6, 7]
const newNumbers = []
numbers.forEach((value, index, array) => {
  newNumbers.push(value)
})
console.log(newNumbers) // [1, 2, 3, 4, 5, 6, 7]
```

map(): Tạo một mảng mới bằng cách thực hiện tính toán trên mỗi phần tử. **map()** không thay đổi mảng cũ

```
const numbers = [1, 2, 3]
const newNumbers = numbers.map((value, index, array) => {
  return value * 2
})
console.log(newNumbers) // [2, 4, 6]
```

filter(): Tạo một mảng mới với những phần tử thỏa điều kiện

```
const numbers = [1, 2, 3]
const newNumbers = numbers.filter((value, index, array) => {
  return value >= 2
})
console.log(newNumbers) // [2, 3]
```

find(): trả về phần tử thỏa điều kiện đầu tiên. Nếu không có sẽ return undefined

```
const numbers = [1, 2, 3]
const result = numbers.find((value, index, array) => {
  return value >= 2
})
console.log(result) // 2
```

findIndex(): trả về index của phần tử thỏa điều kiện đầu tiên. Nếu không có sẽ return **-1**

```
const cars = ['BMW', 'Toyota', 'Hyundai']
const result = cars.findIndex((value, index, array) => {
  return value === 'Toyota'
})
console.log(result) // 1
```

indexOf(): trả về index của phần tử trong mảng. Nếu không có sẽ return -1

```
const cars = ['BMW', 'Toyota', 'Hyundai']
const index = cars.indexOf('Toyota')
console.log(index) // 1
```

every(): Nếu mọi phần tử thỏa điều kiện sẽ return true, còn không sẽ return false

```
const numbers = [1, 2, 3]
const check = numbers.every((value, index, array) => {
  return value > 2
})
console.log(check) // false
```

some(): Nếu có một phần tử thỏa điều kiện sẽ return true, còn không sẽ return false

```
const numbers = [1, 2, 3]
const check = numbers.some((value, index, array) => {
  return value > 2
```

```
})
console.log(check) // true
```

includes(): Kiểm tra một phần tử có trong mảng hay không. return true/false

```
const numbers = [1, 2, 3, 4, 5]
const check = numbers.includes(5) // true
Thường thì các method với tring, array không thay đổi giá trị gốc. Ngoại trừ: pop,
push, shift, unshift, delete
```

reverse(): Đảo ngược thứ tự các item trong array

```
var fruits = ['Banana', 'Orange', 'Apple', 'Mango']
fruits.reverse()
console.log(fruits) // ["Mango", "Apple", "Orange", "Banana"]
```

sort(): Mặc định sẽ sắp xếp các giá trị như là **string** Điều này hoạt động ổn cho string ("Apple" thì đứng trước "Banana"). Tuy nhiên, nếu số cũng bị cho là string, `25` thì bị cho là lớn hơn `100`, bởi vì `2` thì lớn hơn `1`. Bởi vì điều này, phương thức `sort()` sẽ thực thi không chính xác khi sắp xếp number. Bạn có thể fix điều này bằng cách cung cấp một **compare function**. **compare function** sẽ có dạng như thế này.

```
function(a, b){return a - b}
```

Nếu kết quả là âm, `a` sẽ được xếp trước `b`. Nếu kết quả là dương, `a` sẽ được xếp sau `b`. Nếu kết quả là 0, sẽ không có sự thay đổi nào giữa 2 giá trị.

Ví dụ sắp xếp tăng dần:

```
var points = [40, 100, 1, 5, 25, 10]
points.sort(function (a, b) {
  return a - b
})
console.log(points) // [1, 5, 10, 25, 40, 100]
```

Ví dụ sắp xếp giảm dần:

```
var points = [40, 100, 1, 5, 25, 10]
points.sort(function (a, b) {
  return b - a
})
console.log(points) // [100, 40, 25, 10, 5, 1]
```

reduce(): Dùng để "xào nấu" một array, ví dụ tính tổng 1 array hay từ 1 array chuyển thành 1 array, object khác.

```
arr.reduce((total, current, currentIndex) => {
  return something
}, initialValue)
```

`initialValue` là option, có hoặc không vẫn được. Nếu có thì ở vòng lặp đầu tiên, `total = initalValue` và `current = arr[0]`. Tức là currentIndex sẽ chạy từ 0 cho đến hết.

```
var points = [1, 2, [3, 4], 5, 6]
// flat array
const result = points.reduce((total, current) => {
  return total.concat(current)
}, [])
console.log(result) // [1, 2, 3, 4, 5, 6]
```

Còn nếu không dùng `inititalValue` thì đầu tiên `total = arr[0]` và `current = arr[1]`. Tức là currentIndex sẽ chạy từ 1 cho đến hết.

```
var points = [1, 2, 3, 4, 5, 6]
const result = points.reduce((total, current) => {
  return total + current
})
console.log(result) // 21
```