



Data Types

Ba Nguyễn

Math & Number

Kiểu số là một giá trị số nguyên thủy (số nguyên, số thực, ...) hoặc một object **Number**. **Number** cung cấp một số phương thức xử lý với các giá trị số (các giá trị số nguyên thủy cũng có thể gọi các phương thức của object **Number**) và một số hằng số đặc biệt

```
let n = 10; // let n = new Number(10);

// Trả về giá trị chuỗi/chuyển đổi hệ cơ số
// number.toString([radix]) → string
n.toString(2); // "1010"

// Kiểm tra giá trị có phải SỐ HỮU HẠN hay không
// isFinite(number) || Number.isFinite(number) → boolean
isFinite(n); // true

// Kiểm tra giá trị có phải NaN hay không
// isNaN(number) || Number.isNaN(number) → boolean
isNaN(n); // false
```

Math & Number

Kiểu số là một giá trị số nguyên thủy (số nguyên, số thực, ...) hoặc một object **Number**. **Number** cung cấp một số phương thức xử lý với các giá trị số (các giá trị số nguyên thủy cũng có thể gọi các phương thức của object **Number**) và một số hằng số đặc biệt

```
// Parse một giá trị và trả về số nguyên  
// parseInt(string [, radix]) → number || NaN  
// Number.parseInt(string [, radix]) → number || NaN  
parseInt("123abc", 10); // 123
```

```
// Parse một giá trị và trả về số thực  
// parseFloat(string [, radix]) → number || NaN  
// Number.parseFloat(string [, radix]) → number || NaN  
parseFloat("123.123abc"); // 123.123
```

Math & Number

Ngoài object **Number**, JavaScript còn cung cấp module **Math** chứa nhiều phương thức xử lý số

```
Math.floor(number); // Làm tròn xuống
Math.ceil(number); // Làm tròn lên
Math.round(number); // Làm tròn về số gần nhất
Math.trunc(number); // Cắt bỏ phần số thực
Math.pow(x, y); // Tính x mũ y
Math.random(); // Trả về một số ngẫu nhiên từ 0 → 1
Math.min(v1, v2, v3, ... vN); // Trả về số lớn nhất
Math.max(v1, v2, v3, ... vN); // Trả về số nhỏ nhất
```



Tham khảo thêm các hằng số/phương thức xử lý số: [mdn/Numbers](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number), [mdn/Math](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math)

String

Chuỗi là một giá trị chuỗi nguyên thủy (đặt trong cặp dấu `"`, `"`) hoặc một object **String**.

Chuỗi có tính chất đặc biệt, các ký tự được đánh chỉ mục (*index*) và có thể truy cập thông qua chỉ mục đó (index bắt đầu từ **0**). Thuộc tính đặc biệt **length** trả về độ dài (số ký tự) của chuỗi. Đồng thời **String** cung cấp nhiều phương thức xử lý chuỗi (các chuỗi nguyên thủy cũng có thể gọi phương thức của **String**)

	str = "Awesome"						length = 7
value	A	w	e	s	o	m	e
index	0	1	2	3	4	5	6

```
// Truy cập chỉ mục chuỗi  
str[0]; // "A"  
str[6]; // "e"  
str.length; // 7
```

String

```
s.toUpperCase(); // Trả về chuỗi in hoa  
s.toLowerCase(); // Trả về chuỗi in thường  
s.startsWith(subString); // Kiểm tra ký tự bắt đầu  
s.endsWith(subString); // Kiểm tra ký tự kết thúc  
s.substring(start, end); // Trả về một chuỗi con  
s.slice(start, end); // tương tự substring()  
s.includes(subString); // Kiểm tra có chứa chuỗi con hay không  
s.split(saperator, limit); // Tách chuỗi thành một mảng  
s.repeat(count); // Lặp chuỗi  
s.trim(); // Cắt bỏ khoảng trắng 2 đầu  
s.charCodeAt(index); // Lấy mã unicode của một ký tự  
s.indexOf(subString); // Lấy chỉ mục của một chuỗi con  
s.replace(subString, replaceString); // Thay thế ký tự
```



Tham khảo thêm các phương thức xử lý chuỗi: [mdn/Strings](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)

JSON

JSON (JavaScript Object Notation) là một định dạng chung mô tả các đối tượng, các ứng dụng web hiện đại thường trao đổi dữ liệu dưới dạng **JSON**.

JSON.stringify(obj) nhận vào một đối tượng và chuyển đổi nó thành chuỗi được gọi là chuỗi mã hóa JSON. **JSON.parse(str)** nhận vào một chuỗi mã hóa JSON và chuyển đổi nó thành đối tượng JavaScript thông thường

```
let user = { name: "Ba", age: 28 };

JSON.stringify(user);
// '{"name": "Ba", "age": 28}'
JSON.parse('{"name": "Ba", "age": 28}')
// user { name: "Ba", age: 28 }
```

Datetime

Object **Date** trong JavaScript cung cấp các phương thức xử lý dữ liệu về thời gian

```
// Khởi tạo một object Date  
new Date(); // current date time  
new Date(milliseconds); // from 1970-01-01 00:00:00:000  
new Date(string); // parse string to date "YYYY-MM-DD"  
new Date(year, month, date, hours, minutes, seconds, ms);  
Date.now(); // current timestamp
```


Datetime

```
d.getFullYear(); // 2020
d.getMonth(); // 0 → 11
d.getDate(); // 1 → 31
d.getDay(); // 0 (sunday) → 6 (saturday)
d.getTime(); // timestamp
d.getTimezoneOffset(); // UTC+7 ⇒ -420
d.getHours();
d.getMinutes();
d.getSeconds();
d.getMilliseconds();
```

Datetime



```
setFullYear(year, month, date);  
setMonth(month, date);  
setDate(date);  
setHours(hour , min, sec, ms);  
setMinutes(min, sec, ms);  
setSeconds(sec, ms);  
setMilliseconds(ms);  
setTime(ms);
```

Datetime

```
toLocaleString(locale); // "vi-VN"  
toLocaleDateString(locale); // "vi-VN"  
toLocaleTimeString(locale); // "vi-VN"  
  
// 💡 Lấy chênh lệch thời gian  
date1.getTime() - date2.getTime();
```

💡 Tham khảo thêm các phương thức datetime: [mdn/Date](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date)



Arrays

Ba Nguyễn

Array

Array - mảng là một cấu trúc dữ liệu, cho phép lưu trữ bộ sưu tập dữ liệu có thứ tự, mỗi mục trong mảng được gọi là một phần tử, được *đánh chỉ mục (index)* theo thứ tự bắt đầu từ **0**, và cung cấp các phương thức để xử lý dữ liệu trong mảng.

```
// Cú pháp khai báo mảng
```

```
let arr = [];
```

```
let arr = new Array();
```

```
let arr = [10, 20, 30, 40, 50, 60, 70];
```

	arr = [10, 20, 30, 40, 50, 60, 70]						length = 7
value	10	20	30	40	50	60	70
index	0	1	2	3	4	5	6

💡 Thực tế, mảng cũng là object (`typeof array == "object"`)

Array

```
let arr = [10, 20, 30, 40, 50, 60, 70];  
arr.length; // 7  
// Truy cập phần tử mảng  
arr[0]; // 10  
arr[1]; // 20  
// Thay đổi giá trị  
arr[0] = "Mười";  
arr[1] = "Hai mươi";  
// Thêm phần tử  
arr[7] = 80;  
// ["Mười", "Hai mươi", 30, 40, 50, 60, 70, 80]
```

Array

Mảng cho phép lưu dữ liệu bất kỳ, các phần tử không nhất thiết phải có cùng kiểu dữ liệu

```
let arr = ["Ba", 0, { name: "Béo Ú", age: 28 },  
           |   |   |           true, null, [1, 2]];  
arr.length; // 5  
arr[2].name; // "Béo Ú"  
arr[5][0]; // 1  
arr[5].length; // 2
```

Array

`length` là thuộc tính xác định *độ dài* (số lượng phần tử) của mảng, được cập nhật tự động khi mảng được thêm/xóa phần tử.

Tuy nhiên, `length` không phải “độ dài” thực sự (số lượng phần tử) của mảng, mà là **chỉ mục lớn nhất trong mảng + 1**. Thuộc tính `length` có thể thay đổi được, khi `length` giảm, phần tử mảng sẽ bị cắt bớt.

```
let a = [1, 2, 3];  
a[999] = 0;  
a.length; // 1000;  
a[998]; // undefined  
a.length = 0; // clear array
```


Array

Lặp qua tất cả phần tử trong mảng sử dụng vòng lặp `for`, hoặc `for of`

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

for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]); // 1, 2, 3, 4, 5
}

for (let i of arr) {
  console.log(i); // 1, 2, 3, 4, 5
}
```

💡 Mảng mặc định khi chuyển về kiểu **string** sẽ có dạng: `alert(arr); // "1,2,3,4,5"`

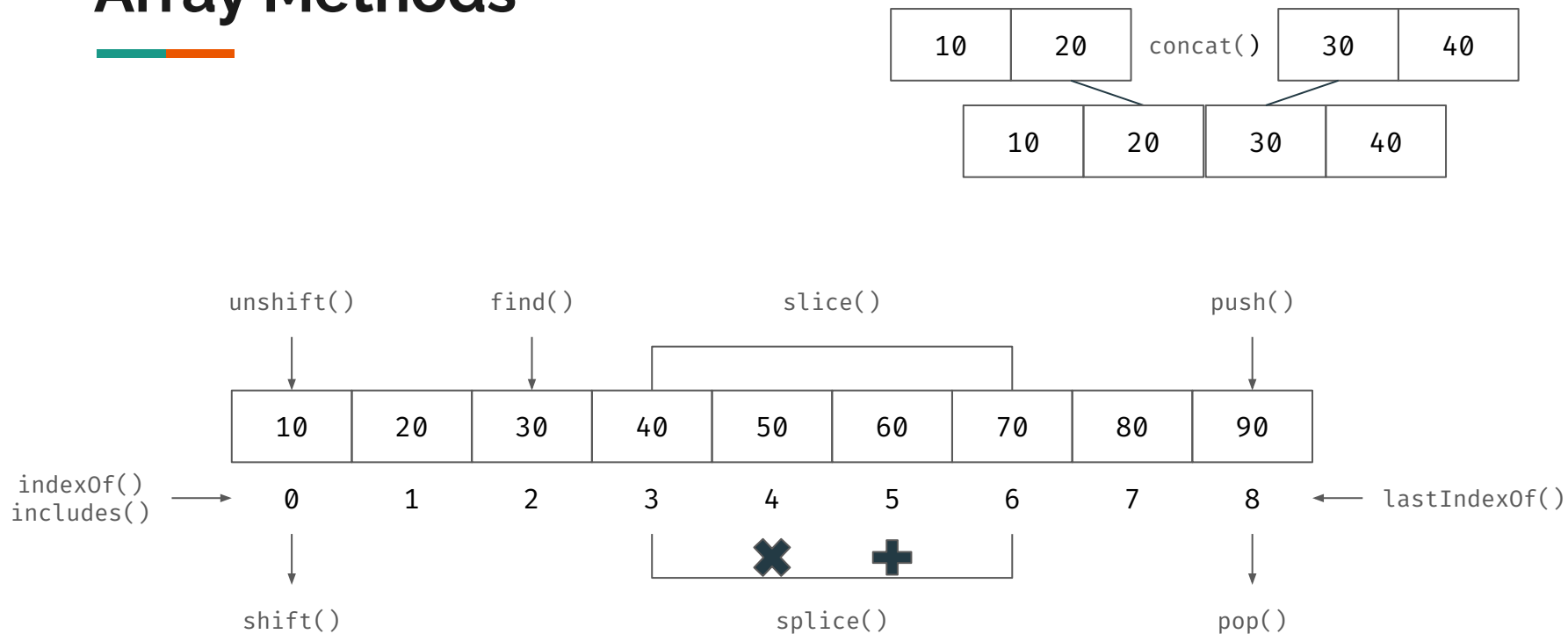
Array Methods

// Một số phương thức array thường dùng

```
arr.reverse();  
arr.join();  
arr.includes();  
arr.lastIndexOf();  
arr.slice();  
arr.pop();  
Array.isArray();  
arr.map();  
arr.filter();
```

```
arr.sort();  
arr.indexOf();  
arr.find();  
arr.concat();  
arr.splice();  
arr.push();  
Array.from();  
arr.forEach();  
arr.reduce();
```

Array Methods



Array Methods

Hàm `sort()` được sử dụng để sắp xếp mảng, nó cập nhật trực tiếp giá trị trong mảng

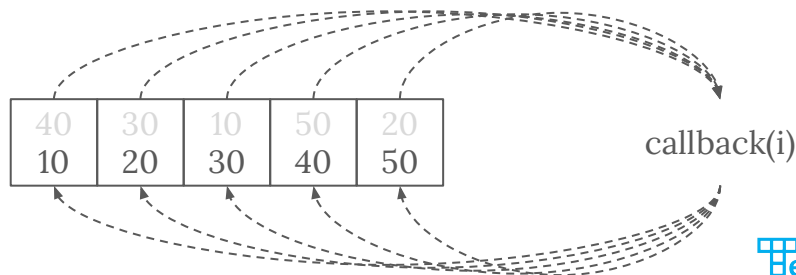
Hàm `sort()` có thể nhận vào một tham số là hàm **callback** để so sánh 2 phần tử của mảng. Mặc định `sort()` so sánh phần tử theo kiểu dữ liệu **string**

```
[1, 2, 15].sort(); // ⇒ "1", "2", "15" ⇒ [1, 15, 2]
```

Để sắp xếp một mảng theo giá trị kiểu number

```
[15, 1, 2].sort((a, b) ⇒ a - b);  
// [1, 2, 15]
```

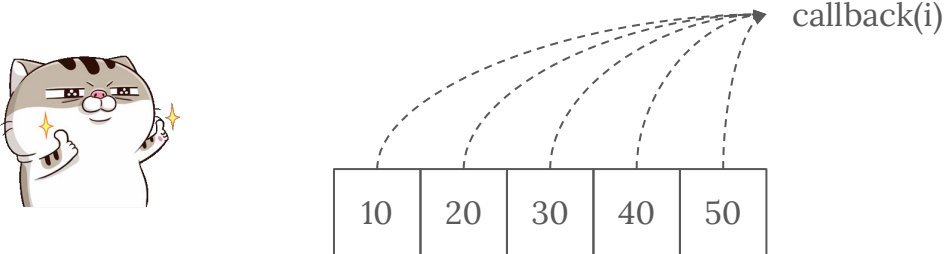
```
[15, 1, 2].sort((a, b) ⇒ b - a);  
// [15, 2, 1]
```



Array Methods

Hàm `forEach()` nhận vào 1 tham số là hàm **callback**, nó lặp qua mảng và với mỗi phần tử, nó gọi hàm **callback** với giá trị của phần tử đó

```
[1, 2, 3].forEach((i) => console.log(i * i)); // 1, 4, 9
[1, 2, 3].forEach((value, index, array) =>
  console.log(`${value} has index ${index} in array [${array}]`)
);
```



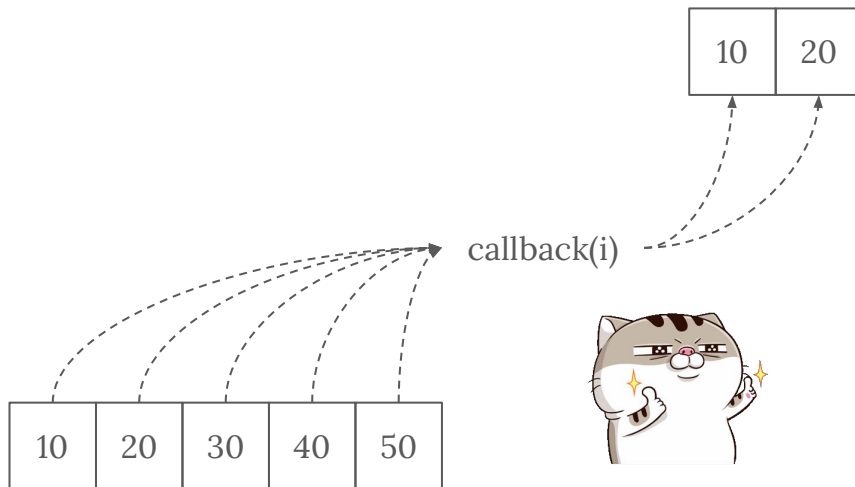
💡 Không thể ngắt `forEach()` với **break** hoặc **continue** và câu lệnh **return** bị bỏ qua

Array Methods

Hàm `filter()` trả về một mảng các phần tử “khớp” với điều kiện chỉ định, nó nhận vào một hàm **callback** để so sánh giá trị, hàm **callback** phải trả về giá trị **true** hoặc **false**

```
let arr = [
  { name: "Ba", age: 29 },
  { name: "Bon", age: 3 },
];

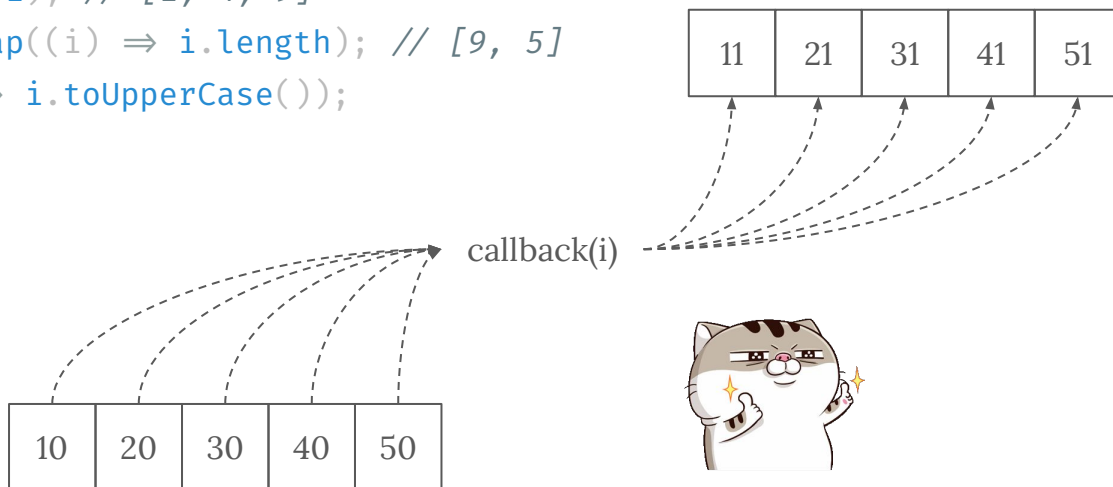
arr.filter((i) => i.age > 20);
// [ {name: "Ba", age: 29} ]
```



Array Methods

Hàm `map()` nhận một tham số là hàm **callback**, với mỗi phần tử, nó gọi hàm **callback** và trả về một mảng mới với giá trị của phần tử là giá trị trả về từ hàm **callback**

```
[1, 2, 3].map((i) => i * i); // [1, 4, 9]
["Ba Nguyen", "Béo Ú"].map((i) => i.length); // [9, 5]
["abc", "def"].map((i) => i.toUpperCase());
// ["ABC", "DEF"]
```



Array Methods

Hàm `reduce()` thực hiện tính toán (tổng hợp) giá trị của mảng, nó nhận vào tham số là một hàm **callback** và một giá trị khởi tạo (tùy chọn)

Cú pháp

Giá trị tích lũy sau mỗi lần lặp, ban đầu nhận giá trị **[initial]** nếu có hoặc giá trị phần tử đầu tiên trong mảng

```
arr.reduce(callback(accumulator, value, index, array) {  
    // code  
    // code  
}, [ initial ] );
```

callback(accumulator, i)



```
[1, 2, 3, 4, 5, 6].reduce((sum, i) => (sum += i));  
// 17
```





Scheduling

Ba Nguyễn

setTimeout()

setTimeout() cho phép “đặt lịch” cho một hành động nào đó, sẽ được thực thi sau một khoảng thời gian nhất định

Cú pháp:

```
let timer = setTimeout(func | code[, delay][, args]);  
clearTimeout(timer);
```

Ví dụ:

```
let timer = setTimeout(() => {  
    alert("Hello babe");  
}, 1000); // thông báo alert() sau 1s
```

```
let timer = setTimeout(function hi() {  
    alert("Hello babe");  
    timer = setTimeout(hi, 1000);  
}, 1000); // lặp thông báo alert() mỗi 1s
```

setInterval()

setInterval() cho phép “đặt lịch” cho một hành động nào đó, sẽ được thực thi lặp đi lặp lại sau mỗi khoảng thời gian nhất định

Cú pháp:

```
let timer = setInterval(func | code[, delay][, args]);  
clearInterval(timer);
```

Ví dụ:

```
let timer = setInterval(() => {  
    console.log("I love you!");  
}, 1000); // mỗi 1s in ra "I love you!"
```

```
let seconds = 1;  
let timer = setInterval(() => {  
    console.log(seconds++);  
}, 1000); // mỗi 1s in ra seconds
```