Data Types

Ba Nguyễn

Math & Number

Kiểu **number** là một giá trị số nguyên thủy (số nguyên, số thực, ...) hoặc một đối tượng **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);

// Một số phương thức của Number
n.toString(); // Trả về chuỗi
n.toString(16); // Chuyển đổi hệ cơ số
n.toLocaleString("vi-VN"); // Định dạng số in ra
n.toFixed(2); // Cắt bớt số thập phân
```

Math & Number

```
// Kiểm tra n có phải hữu hạn hay không
isFinite(n); // true

// Kiểm tra n có phải NaN hay không
isNaN(n);

// Phân tích chuỗi và trả về số nguyên
parseInt("123abc"); // 123

// Phân tích chuỗi và trả về số thực
parseFloat("123.456.789abc"); // 123.456
```

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.max(number, number, ...numbers); // Tìm số lớn nhất
Math.min(number, number, ...numbers); // Tìm số nhỏ nhất
Math.random(); // Trả về số ngẫu nhiên 0 - 1
// Lấy số ngẫu nhiên 0 -> number
Math.floor(Math.random() * number);
// Lấy số ngẫu nhiên a -> b
Math.floor(Math.random() * (b - a)) + a;
```

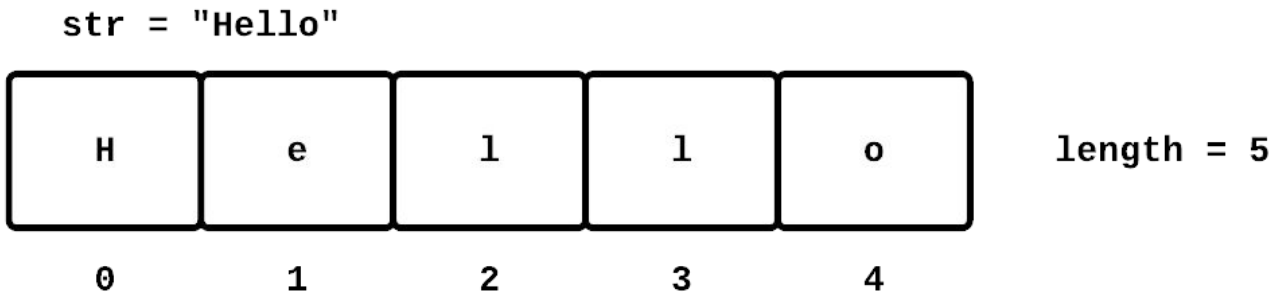
💡 Tham khảo thêm các hằng số/phương thức xử lý số: mdn/Numbers, mdn/Math

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).

Chuỗi có 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**)



String

Một số phương thức của **String**

```
str.toLowerCase(); // Chuỗi in thường
str.toUpperCase(); // Chuỗi in hoa
str.slide(start, end); // Lấy chuỗi con
str.includes(substr); // Có chứa chuỗi substr hay không
str.indexOf(substr); // Vị trí chuỗi con (trái -> phải)
str.lastIndexOf(substr); // Vị trí chuỗi con (phải -> trái)
str.replace(substr, value); // Thay thế chuỗi substr đầu tiên
str.replaceAll(substr, value); // Thay thế toàn bộ chuỗi substr
str.trim(); // Cắt bỏ khoảng trắng
str.split(saperator); // Cắt chuỗi thành mảng
```

💡 Tham khảo thêm các phương thức xử lý chuỗi: mdn/Strings

Datetime

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

Sử dụng phương thức **Date()** để khởi tạo đối tượng date mới

```
new Date(); // Thời gian hiện tại
new Date("YYYY-MM-DD HH:MM:SS"); // Theo chuỗi thời gian
new Date(millisecond); // Tính từ 1970-01-01 00:00:00:0000
new Date(year, month, day, hour, minute, second, ms);
Date.now(); // Số ms tính từ 1970-01-01 00:00:00:0000
```

Datetime

Một số phương thức của **Date**

```
date.getFullYear(); // 2020
date.getMonth(); // 0 -> 11
date.getDate(); // 0 -> 31
date.getDay(); // T2 -> CN (0 - 6)
date.getTime(); // millisecond
date.getHours(); // 0 -> 23
date.getMinutes(); // 0 -> 59
date.getSeconds(); // 0 -> 59
date.getMilliseconds(); // 0 -> 999
```

Datetime

```
date.setFullYear(year); // Chỉnh sửa năm
date.setMonth(month); // Chỉnh sửa tháng (0 -> 11)
date.setDate(date); // Chỉnh sửa ngày (1 -> 31)
date.setHours(hour); // Chỉnh sửa giờ
date.setMinutes(minute); // Chỉnh sửa phút
date.setSeconds(second); // Chỉnh sửa giây
date.setMilliseconds(ms); // Chỉnh sửa ms
date.setTime(ms); // Chỉnh sửa theo số ms (từ 1970)
```

💡 Các giá trị vượt quá giới hạn tự động điều chỉnh cho phù hợp

💡 Tham khảo thêm các phương thức datetime: mdn/Date

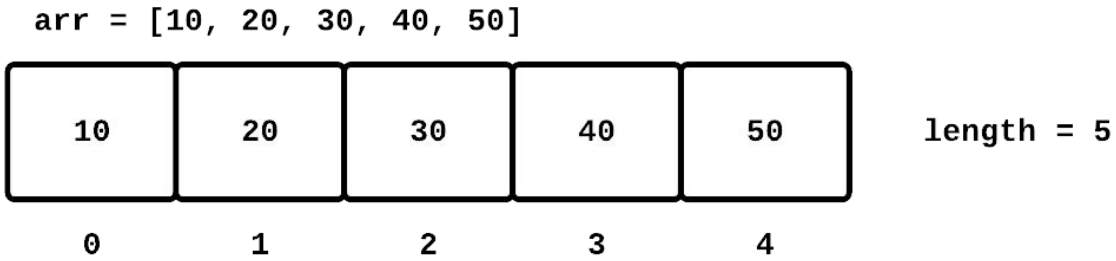
Arrays

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.

Thuộc tính **length** trả về độ dài (số lượng phần tử) của mảng



💡 Thực tế, mảng cũng là object (**typeof (array) === “object”**)

💡 Thuộc tính **length** bằng **index** lớn nhất **+1**, và có thể chỉnh sửa trực tiếp (xóa, mở rộng mảng)

Array

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

```
let arr = [];
let arr = new Array();
let arr = new Array(10); // 10 phần tử
let arr = [10, 20, 30, 40, 50];
let arr = [10, "Hai mươi", true, null];
let arr = [[1, 0, -1], { name: "Ba", age: 29 }];
```

Array

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

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

for (let i = 0; i < arr.length; i++) {
    console.log(arr[i]); // 10 20 30 40 50
}

for (let i of arr) {
    console.log(i); // 10 20 30 40 50
}
```

💡 Mảng mặc định khi chuyển về kiểu ***string*** sẽ có dạng: ***“10,20,30,40,50”***

Array Methods

```
Array.isArray(arr); // Kiểm tra giá trị có phải mảng
arr.reverse(); // Đảo ngược mảng
arr.includes(value); // Mảng có chứa value hay không
arr.indexOf(value); // Chỉ mục của value (trái -> phải)
arr.lastIndexOf(value); // Chỉ mục của value (phải -> trái)
arr.pop(); // Xóa phần tử cuối cùng - trả về giá trị
arr.push(value); // Thêm giá trị vào cuối mảng
arr.slice(from, to); // Sao chép các giá trị trong mảng
arr.splice(from, to); // Xóa hoặc chèn phần tử
arr.join(saperator); // Nối mảng thành chuỗi
arr.concat(other); // Nối 2 (hoặc nhiều) thành 1 mảng mới
```

💡 Tham khảo thêm các phương thức của mảng: mdn/Array

Callback

Callback là một hàm được truyền vào hàm khác dưới dạng đối số, và được gọi trong hàm đó.

Lưu ý: Hàm **callback** được truyền dưới dạng **giá trị hàm - reference**, không phải cuộc gọi hàm

```
function func(data) { alert(data); }

function demo(callback) {
    let data = 10;
    callback(data);
}

demo(func); // callback = func
```

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 các tham số đặc biệt

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

function func(value, index) { console.log(value); }

arr.forEach(func); // 10 20 30 40 50

arr.forEach(function (value, index) {
    console.log(value); // 10 20 30 40 50
});
```

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

Array Methods

Hàm **find()** nhận vào 1 tham số là hàm **callback**, nó lặp qua mảng và với mỗi phần tử, nó gọi hàm **callback** với các tham số đặc biệt.

Hàm **callback** đánh giá các phần tử, kết quả trả về phần tử đầu tiên thỏa mãn điều kiện hoặc **undefined** nếu không có phần tử nào thỏa mãn

```
arr.find(function (value, index) {
    return value % 3 == 0
            && value % 5 == 0;
}); // 30
```

Array Methods

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

Mặc định, các phần tử trong mảng sẽ được chuyển về kiểu **string** và so sánh, nếu mảng chứa các giá trị số, sắp xếp bằng hàm **sort()** đơn thuần có thể làm sai lệch kết quả

```
let arr = ["a", "d", "c", "b"];
arr.sort(); // ["a", "b", "c", "d"]

let arr = [1, 3, 2, 11, 21];
arr.sort(); // [1, 11, 2, 21, 3]
```

Array Methods

Để sắp xếp mảng số, hoặc các đối tượng phức tạp, hàm **sort()** có thể nhận một tham số là hàm **callback** để so sánh 2 phần tử của mảng

```
let arr = [1, 3, 2, 11, 21];

arr.sort(function (a, b) { return a - b; });
// [1, 2, 3, 11, 21]

arr.sort(function (a, b) { return b - a; });
// [21, 11, 3, 2, 1]
```

Array Methods

Để sắp xếp mảng số, hoặc các đối tượng phức tạp, hàm **sort()** có thể nhận một tham số là hàm **callback** để so sánh 2 phần tử của mảng

```
let arr = [{ x: 1 }, { x: 3 }, { x: 2 }, { x: 0 }];

arr.sort(function (a, b) {
    return a.x - b.x;
});
// [{ x: 0 }, { x: 1 }, { x: 2 }, { x: 3 }]
```

Array Methods

Hàm **filter()** sử dụng để lọc ra các phần tử trong mảng khớp với một điều kiện nào đó. Nó nhận vào một hàm **callback** để so sánh giá trị, hàm **callback** phải trả về giá trị **true** hoặc **false**

```
let arr = [10, "Hai mươi", true, null];

arr.filter(function (value, index) {
    return typeof value === "number";
}); // [ 10 ]
```

Array Methods

Hàm **map()** sử dụng để biến đổi một mảng, nó nhận tham số là một hàm **callback**. Với mỗi phần tử của mảng, nó gọi hàm **callback** và ghi giá trị trả về từ hàm **callback** vào một mảng mới.

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

arr.map(function (value, index) {
    return (value % 10) * 2;
}); //[ 2, 4, 6, 8, 10]
```

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)

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

arr.reduce(function (sum, value, index) {
    sum += value;
    return sum; // Trả về giá trị để tích lũy
}, 0); // Giá trị khởi tạo sum = 0

// Kết quả: 150
```

Exercises

1. Viết hàm **sumAvg(arr)** tính trung bình cộng một mảng số
2. Viết hàm **findMax(arr)** tìm giá trị lớn nhất trong một mảng
3. Viết hàm **fibonacci(n)** trả về một mảng chứa **n** số Fibonacci
4. Viết hàm **removeFalsy(arr)** trả về một mảng mới chỉ chứa các giá trị **đúng** trong mảng
5. Viết hàm **sortByColumn(arr, col)** sắp xếp một mảng 2 chiều theo giá trị cột **col** tăng dần
6. Viết hàm **double(arr)** trả về một mảng mới với giá trị các phần tử bằng bình phương chính nó
7. Viết hàm **capitalize(str)** chuyển đổi một chuỗi thành dạng capitalize.

VD: capitalize(“hello world!”); // “Hello World!”

8. Viết hàm **randItem(arr)** trả về một phần tử ngẫu nhiên trong mảng