



Data Types Methods

Ba Nguyễn

Primitive Methods

JavaScript cho phép gọi các phương thức của object trên giá trị nguyên thủy (primitive) giống như một object thông thường. Ngoại trừ **null** và **undefined** là 2 kiểu đặc biệt, không có bất kỳ phương thức nào

```
"abc".toUpperCase(); // "ABC" → new String("abc").toUpperCase()  
(1).toString(); // "1" → new Number(1).toString()  
true.valueOf(); // true → new Boolean(true).valueOf()  
"Ba đẹp trai 😄".split(" "); // 🤔 → new String("Ba đẹp trai 😄").split(" ")
```



Math & Number Methods

Một số phương thức với **Math** và **Number** thường dùng

`toString(radix);` // trả về giá trị chuỗi, hoặc chuyển đổi hệ cơ số, ...

`isFinite(number);` // kiểm tra một giá trị khi chuyển về kiểu number có phải số hữu hạn hay không

`isNaN(number);` // kiểm tra một giá trị khi chuyển về kiểu number có phải NaN hay không

`parseInt(string [, radix]);` // parse một giá trị và trả về số nguyên

`parseFloat(string [, radix]);` // parse một giá trị và trả về số thực

`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, trả về số nguyên

`Math.pow(x, y);` // tính số mũ

`Math.random();` // trả về một số ngẫu nhiên từ 0 → 1

`Math.min(... values); Math.max(... values);` // trả về số lớn/nhỏ nhất trong một tập hợp số

Exercise



1. Viết hàm tạo một số tự nhiên ngẫu nhiên từ 0 đến n. VD `rand(10)` => 8
2. Viết hàm chuyển đổi một số từ hệ cơ số này sang hệ cơ số khác.

VD `convert(255, 10, 16)` => Chuyển giá trị 255 từ cơ số 10 sang 16 => "ff"

3. Viết hàm tạo một số nguyên ngẫu nhiên trong khoảng từ a -> b. VD `rand(10, 100)`; => 34
4. Viết hàm tạo mã màu HEX ngẫu nhiên. VD `hex()` => "fea34f"

String Methods

Chuỗi có một số tính chất đặc biệt, giá trị của chuỗi là bất biến (immutable), các ký tự trong chuỗi được đánh số chỉ mục (bắt đầu từ 0) và cho phép truy cập ký tự thông qua chỉ mục đó. Thuộc tính **length** trả về độ dài (số ký tự) của chuỗi.

```
str[0]; // "A"  
str[6]; // "e"  
str[2]; // "e"  
str[4]; // "o"  
str.length; // 7
```

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

String Methods

Một số phương thức với **String** thường dùng

`toUpperCase(); toLowerCase();` // trả về chuỗi in hoa / thường
`startsWith(); endsWith();` // kiểm tra chuỗi có bắt đầu/kết thúc bằng một chuỗi con hay không
`substring();` // trả về một chuỗi con trong chuỗi
`slice();` // tương tự `substring()`
`includes();` // kiểm tra chuỗi có chứa một chuỗi con hay không
`split();` // tách chuỗi thành một mảng
`repeat();` // lặp chuỗi
`trim();` // cắt bỏ khoảng trắng
`charCodeAt();` // lấy mã unicode của một ký tự
`indexOf();` // lấy chỉ mục của một chuỗi con
`replace();` // thay thế các ký tự trong chuỗi

Exercise

1. Viết hàm cắt chuỗi từ vị trí đầu tiên, tới vị trí chỉ định. VD: “abcd”, 2 => “abc”
2. Viết hàm chuyển đổi một tên thành tên viết tắt. VD: “Ba Nguyễn” => “Ba N.”
3. Viết hàm ẩn địa chỉ email. VD: “banguyen@techmaster.vn” => “ba...@techmaster.vn”

Homework

1. Viết hàm chuyển đổi một chuỗi thành dạng capitalize. VD: “hello world” => “Hello World”
2. Viết hàm chuyển đổi một chuỗi thành dạng paramaterize. VD “Hello World” => “hello-world”
3. Viết hàm loại bỏ khoảng trắng ở đầu, cuối, và rút gọn khoảng trắng ở giữa các từ trong chuỗi. VD: “Hello world ! ” => “Hello world !”
4. Viết hàm đảo ngược chữ thường thành chữ hoa, và ngược lại. VD “aBcD” => “AbCd”
5. Viết hàm lặp chuỗi n lần. VD “Hehe”, 3 => “HeheHeheHehe”
6. Viết hàm chèn một chuỗi con vào chuỗi tại index chỉ định. VD “ac”, “b”, 1 => “abc”
7. Viết hàm rút gọn chuỗi nếu chuỗi dài hơn giá trị chỉ định. VD “abcdef”, 2 => “ab...”;
8. Viết hàm đếm số lần xuất hiện của chuỗi con trong chuỗi. VD “abca”, “a” => 2
9. Viết hàm cắt chuỗi theo số từ chỉ định. VD “My name is Ba”, 2 => “My name”
10. Viết hàm tạo chuỗi GUID ngẫu nhiên với độ dài 32 ký tự. VD
“sfjh2ih4(U#%(kljo423oiir*(#%UIOJ”
11. Viết hàm thay thế các ký tự trong chuỗi thành ký tự liền sau trong bảng mã Unicode. VD:
“ad” => “be”

Array

Array - mảng là một cấu trúc dữ liệu, cho phép lưu trữ bộ sưu tập dữ liệu có thứ tự, thực tế, mảng cũng là **object**, nó cung cấp các phương thức để xử lý dữ liệu trong mảng (**typeof array == "object"**). Mỗi mục trong mảng được gọi là một phần tử, được “đánh số” chỉ mục (**index**) theo thứ tự bắt đầu từ 0

Cú pháp khai báo mảng

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

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

	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

Array

Ví dụ:

```
let students = ["ba", "Béo"]; // length = 2
// truy cập phần tử mảng
students[0]; // "ba"
students[1]; // "Béo"
students[-1]; // undefined
// thay đổi giá trị
students[0] = "Ba";
students[1] = "Đẹp";
// thêm phần tử
students[2] = "Trai";
students[3] = "😄"; // students = ["Ba", "Đẹp", "Trai", "😄"]
```

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[2].name; // "Béo Ú"
arr[5][0]; // 1
arr.length; // 5
arr[5].length; // 2
typeof arr[0]; // "string"
typeof arr[2]; // "object"
typeof arr[4]; // "null"
typeof arr[5]; // "object"
```



Array

length là thuộc tính xác định “độ dài” 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

```
let arr = [1, 2, 3, 4, 5, 6, 7];  
  
for (let i = 0; i < arr.length; i++) {  
    arr[i] = arr[i] * 2;  
}
```

```
alert(arr); // "2,4,6,8,10,12,14"
```

💡 Mảng mặc định khi chuyển về kiểu **string** sẽ có dạng

💡 Vòng lặp **for in** cũng có thể duyệt qua mảng, tuy nhiên nên sử dụng **for** thông thường

Array Methods

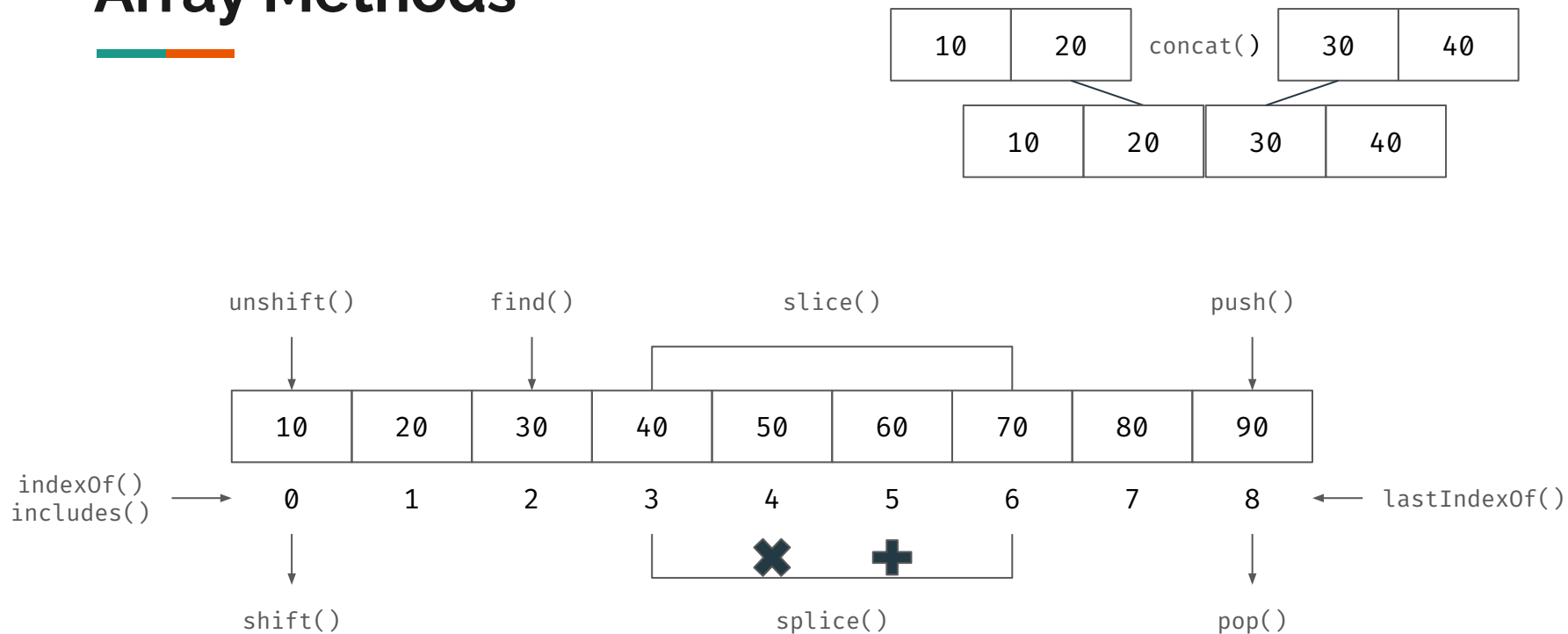
Một số phương thức mảng thường dùng

```
arr.reverse(); // đảo ngược mảng  
arr.slice();  
arr.join(); // gộp mảng thành chuỗi  
arr.push();  
arr.concat();  
arr.pop();  
arr.shift();  
arr.unshift();  
Array.isArray(); // kiểm tra có phải mảng hay không  
Array.from(); // chuyển đổi một collection thành mảng
```

```
arr.sort();  
arr.splice();  
arr.indexOf();  
arr.lastIndexOf();  
arr.includes();  
arr.map();  
arr.find();
```



Array Methods



Exercise

1. Viết hàm **arr._concat(arr2)** gộp các phần tử của mảng **arr2** vào **arr1**
2. Viết hàm **arr._push(value)** thêm giá trị vào cuối mảng
3. Viết hàm **arr._pop()** xóa phần tử cuối mảng, đồng thời trả về giá trị của phần tử bị xóa
4. Viết hàm **arr._indexOf(value)** tìm và trả về index của phần tử, nếu không có trả về -1
5. Viết hàm **arr._reverse()** đảo ngược giá trị mảng

Array Methods

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

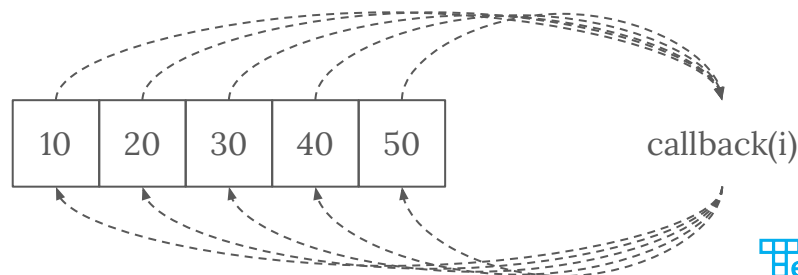
Hàm `sort()` có thể nhận vào một tham số là hàm **callback** để so sánh 2 phần tử của mảng. Mặc định `sort()` so sánh phần tử theo kiểu dữ liệu **string**

```
[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]
```

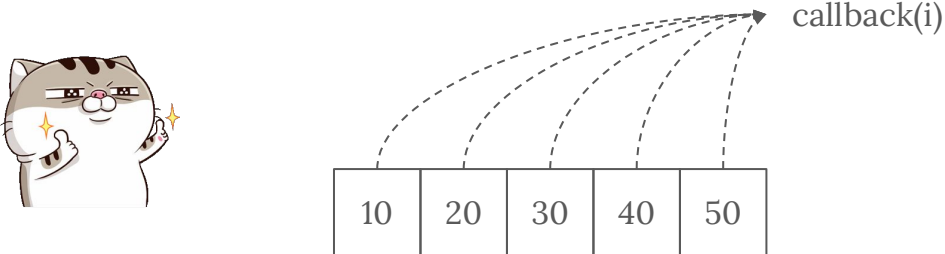
```
[15, 1, 2].sort((a, b) ⇒ b - a);  
// [15, 2, 1]
```



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}]`)
);
```

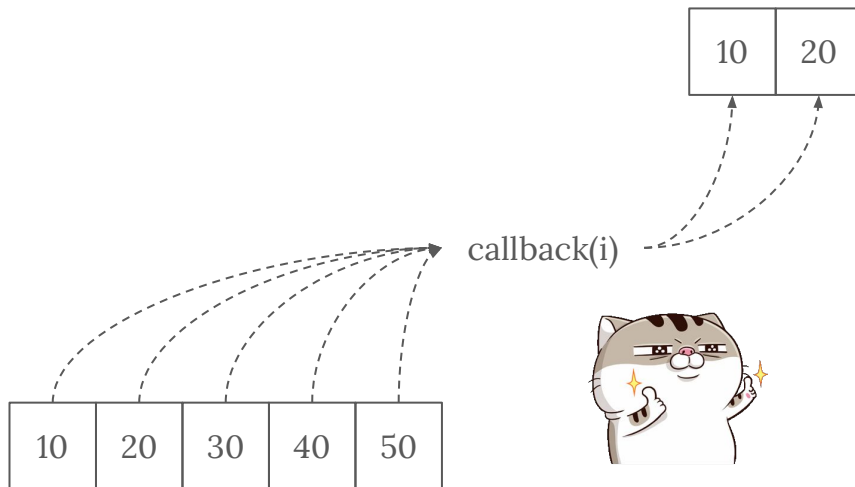


💡 Không thể ngắt `forEach()` với **break** hoặc **continue** và câu lệnh **return** bị bỏ qua

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**

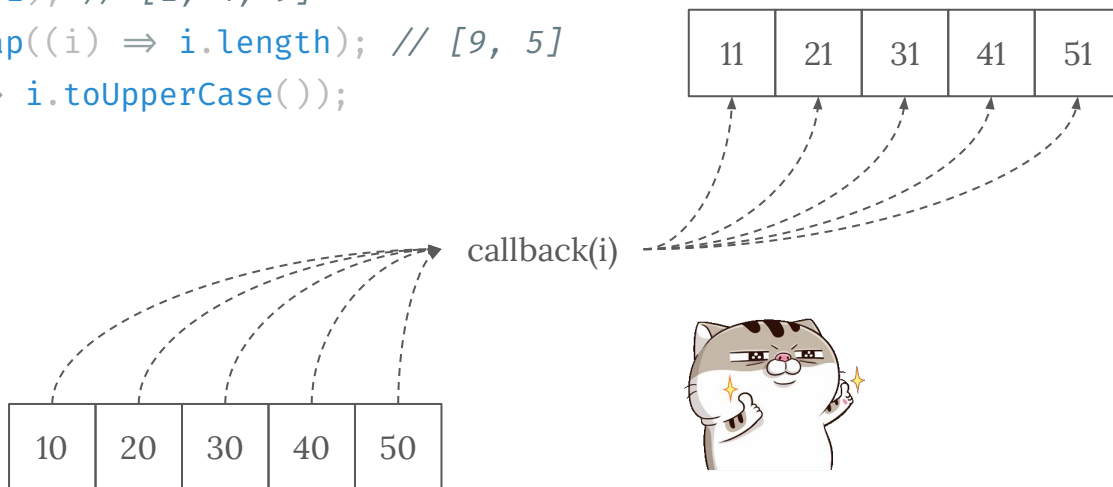
```
let arr = [  
  { name: "Ba", age: 29 },  
  { name: "Bon", age: 3 },  
];  
  
arr.filter((i) => i.age > 20);  
// [ {name: "Ba", age: 29} ]
```



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"]
```



Exercise

1. Viết hàm **arr._sort(arr, callback)** thực thi code giống như hàm **sort()**
2. Viết hàm **arr._forEach(arr, callback)** thực thi code giống như hàm **forEach()**
3. Viết hàm **arr._filter(arr, callback)** thực thi code giống như hàm **filter()**
4. Viết hàm **arr._map(arr, callback)** thực thi code giống như hàm **map()**
5. Viết hàm **arr._reduce(arr, callback)** thực thi code giống như hàm **reduce()**



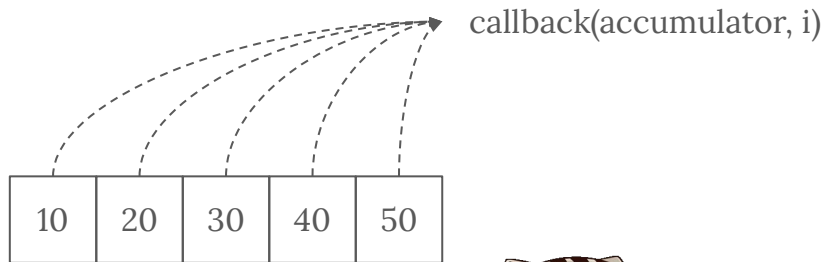
Array Methods

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 ] );
```



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



Homework

1. Cho một mảng số, viết hàm tính trung bình cộng tất cả phần tử trong mảng
2. Cho một mảng số, viết hàm tìm index của một số trong mảng
3. Viết hàm sao chép một mảng số
4. Cho một mảng số, viết hàm tìm giá trị lớn nhất trong mảng
5. Viết hàm đổi chỗ vị trí 2 phần tử trong mảng
6. Cho một mảng số đã được sắp xếp tăng dần, viết hàm tìm số lớn thứ 2 trong mảng
7. Viết hàm chuyển đổi một chuỗi thành dạng capitalize. VD "hello world" => "Hello World"
8. Viết hàm tìm số lần xuất hiện lớn nhất của một phần tử trong mảng
9. Viết hàm cắt chuỗi thành một mảng có độ dài chỉ định. VD "Hello", 2 => ["He", "ll", "o"]
10. Viết hàm tách chuỗi thành một mảng các chuỗi con. VD "dog" => ["d", "do", "dog", "og", "g"]
11. Cho một mảng số, viết hàm loại bỏ số trùng lặp trong mảng. VD [1,2,2,3] => [1,2,3]
12. Viết hàm trả về một mảng lưu dãy số Fibonacci từ 0 -> n. VD 8 => [0, 1, 1, 2, 3, 5, 8, 13]
13. Viết hàm trả về một mảng các số trùng nhau trong 2 mảng. VD [1,2,3], [2,3,4] => [2,3]
14. Viết hàm trả về một mảng các số không trùng nhau trong 2 mảng. VD [1,2,3], [2,3,4] => [1,4]
15. Viết hàm loại bỏ các giá trị "false" khỏi mảng. VD [null, 1, 0, NaN, ""] => [1]

Homework

1. Viết hàm sắp xếp một mảng số nguyên
2. Viết hàm sắp xếp một mảng "string"
3. Cho một mảng object user [{name: "Ba", age: 28}, {name: "Bon", age: 3}, ...] Viết hàm sắp xếp mảng user tăng dần theo age
4. Tương tự, viết hàm sắp xếp mảng user theo name.length
5. Viết hàm sắp xếp mảng user theo name
6. Cho một mảng số, và một số n, tìm trong mảng vị trí 2 phần tử có tổng bằng n, kết quả trả về là một mảng lưu vị trí 2 phần tử, hoặc mảng rỗng nếu không tìm thấy
7. Viết hàm lấy một phần tử ngẫu nhiên trong mảng
8. Viết hàm sắp xếp mảng với vị trí ngẫu nhiên (xáo trộn mảng)
9. Viết hàm biến một mảng 2 chiều thành mảng 1 chiều.

VD $[[1,2,3],[3,4,5]] \Rightarrow [1,2,3,3,4,5]$

10. Viết hàm biến một mảng nhiều chiều (3 hoặc nhiều hơn) thành mảng một chiều

Homework

1. Viết hàm biến đổi các phần tử của mảng số nguyên thành bình phương của chính nó
2. Viết hàm biến đổi các phần tử của mảng chuỗi thành dạng uppercase()
3. Viết hàm lọc ra các phần tử có kiểu “number” trong một mảng hỗn hợp
4. Tạo một mảng object với các thông tin name, age, ...
5. Viết hàm lọc ra các object với age > 20
6. Viết hàm chuyển đổi name của object thành dạng capitalize
7. Viết hàm chuyển đổi name của object thành dạng viết tắt. VD “Ba Nguyen” => “Ba N.”
8. Viết hàm để chuyển mảng object thành một mảng chỉ chứa name.

VD [{name: “Ba”, age: 28}, {name: “Béo Ú”, age: 82}] => [“Ba”, “Béo Ú”]