JavaScript

Buổi 5: Phương thức của mảng (Nâng cao).

Nội dung buổi 5

- Phương thức cơ bản của mảng (Tiếp theo).
- forEach, some, every
- find, findIndex
- map, filter, reduce
- Chaining methods.
- flat, flatMap.
- Sorting.

Phương thức cơ bản của mảng (Tiếp theo)

Phương thức cơ bản của mảng (Tiếp theo)

slice: Tạo ra một mảng mới. Nhận 2 tham số đại diện cho giá trị index đầu tiên và giá trị index cuối cùng của mảng mới theo mảng gọi phương thức trước đó. Không có tham số thứ 2, mặc định sẽ lấy đến index cuối cùng.

```
[1, 2, 3, 4].slice(0, 2); // [1, 2]
[1, 2, 3, 4].slice(2, 3); // [3]
[1, 2, 3, 4].slice(1); // [2, 3, 4]
```

splice: Loại bỏ các phần tử của mảng. Nhận 2 tham số, tham số thứ nhất là vị trí index bắt đầu, tham số thứ 2 là số phần tử sẽ loại bỏ. Bản thân phương thức sau khi được gọi sẽ trả lại mảng có các phần tử bị loại bỏ đó.

```
const array = [1, 2, 3, 4];
array.splice(0, 2); // => [1, 2]
array; // [3, 4]
```

Phương thức cơ bản của mảng (Tiếp theo)

reverse: Làm đảo ngược vị trí các phần tử của mảng hiện tại.

```
[1, 2, 3, 4].reverse();
// => [4, 3, 2, 1]
```

concat: Nhận tham số là một hoặc các mảng, trả về mảng mới với các tham số đã được ghép mảng vào mảng gọi phương thức này.

```
[1].concat([2, 3]); // => [1, 2, 3]
[1].concat([2], [3]); // => [1, 2, 3]
```

Bài thực hành số 1

Bài thực hành số 1

Bài 1: Viết một hàm nhận tham số là một mảng, hàm trả về một mảng gồm có 3 phần tử cuối cùng của mảng tham số. Nếu mảng tham số có ít hơn 3 phần tử thì trả về toàn bộ mảng đó.

Bài 2: Viết một hàm nhận 2 tham số, tham số thứ nhất là một mảng, tham số thứ 2 là một giá trị số. Hàm loại bỏ phần tử có vị trí index bằng giá trị của tham số thứ 2 trong mảng tham số. Nếu tham số thứ 2 không hợp lệ trả về undefined. Không sử dụng vòng lặp để giải quyết bài toán này.

Bài 3: Viết một hàm nhận 3 tham số là 3 mảng gồm các chữ số. Hàm kiểm tra tính hợp lệ của các mảng, các mảng hợp lệ là mảng có phần tử là số 1. Hàm trả về kết quả là một mảng gồm các phần tử của các mảng hợp lệ. Không sử dụng vòng lặp.

Ví dụ:

joinValidArrays([1, 3, 5], [2, 4], [1, 6]) => [1, 3, 5, 1, 6]

joinValidArrays([3, 5], [1, 2, 4], [6]) => [1, 2, 4]

joinValidArrays([3, 5], [2, 4], [6]) => [ ]

Phương thức lặp của mảng

Phương thức lặp của mảng

- JavaScript cung cấp sẵn cho kiểu dữ liệu mảng các phương thức lặp để phục vụ nhiều mục đích khác nhau. Ví dụ như lọc, kiểm tra điều kiện phần tử, tìm phần tử, tạo ra mảng mới dựa trên dữ liệu của mảng cũ.
- Cách thức hoạt động:
    - Phương thức nhận tham số là một hàm (đây gọi là khái niệm call back).
    - Phương thức tiến hành lặp từng phần tử của mảng. Mỗi lần lặp, nó sẽ gọi hàm nhận được từ tham số và truyền phần tử ở lần lặp đó vào trong hàm.
    - Với kết quả trả về từ mỗi lần gọi hàm, tùy theo phương thức sẽ trả lại kết quả khác nhau.

Phương thức lặp của mảng

Ví dụ: In ra cửa sổ console giá trị của lần lượt từng phần tử trong mảng.

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

array.forEach((item) => {
  console.log(item);
});
```

```
// Lặp phần tử 1 thực thi khối code
{
  console.log(1);
}

// Lặp phần tử 2 thực thi khối code
{
  console.log(2);
}

// Lặp phần tử 3 thực thi khối code
{
  console.log(3);
}

// Lặp phần tử 3 thực thi khối code
{
  console.log(4);
}
```

Phương thức “forEach” có tác dụng gọi hàm trong tham số với mỗi lần lặp, phương thức này không cần trả kết quả.

- Lần lặp 1: Phương thức gọi hàm bên trong tham số và truyền phần tử đầu tiên của mảng là số 1. Hàm in ra cửa sổ console giá trị là 1.
- Tương tự với các lần lặp tiếp theo cho đến phần tử cuối cùng trong mảng.

Phương thức lặp của mảng

some: Sử dụng để kiểm tra một phần tử bất kỳ trong mảng có thỏa mãn một điều kiện nào đó. Sử dụng hàm call back để kiểm tra điều kiện, hàm trả về kết quả Truthy nếu thỏa mãn, Falsy với trường hợp ngược lại. Phương thức sẽ trả về kết quả là true nếu có tối thiểu một call back trả kết quả về giá trị Truthy.

Bài toán: Kiểm tra xem trong mảng có phần tử nào lớn hơn 5 hay không.

```
const array = [2, 3, 5, 7, 9];

const hasItemLargerThanFive = array.some((item) => {
  return item > 5;
});

console.log(hasItemLargerThanFive); // true
```

Phương thức lặp của mảng

every: Sử dụng để kiểm tra tất cả phần tử bất kỳ trong mảng có thỏa mãn một điều kiện nào đó. Sử dụng hàm call back để kiểm tra điều kiện, hàm trả về kết quả Truthy nếu thỏa mãn, Falsy với trường hợp ngược lại. Phương thức sẽ trả về kết quả là true nếu tất cả call back trả kết quả về giá trị Truthy.

Bài toán: Kiểm tra xem tất cả phần tử trong một mảng có phải là giá trị số (Truthy) không.

```
const array = [2, 3, 5, 7, 9];

const numbersAreValid = array.every((item) => {
  return typeof item === "number" && !Number.isNaN(item);
});

console.log(numbersAreValid); // true
```

Phương thức lặp của mảng

find: Sử dụng để tìm phần tử thỏa mãn một điều kiện nào đó. Sử dụng hàm call back để kiểm tra điều kiện, hàm trả về kết quả Truthy nếu thỏa mãn, Falsy với trường hợp ngược lại. Phương thức sẽ trả về phần tử đầu tiên thỏa mãn điều kiện trong hàm call back.

Bài toán: Tìm phần tử trong mảng có chữ cái bắt đầu là chữ "J". Trả về chuỗi tìm được.

```
const names = ["Tom", "Frank", "Jane", "Henry", "Jimmy"];

const nameStartsWithJ = names.find((name) => {
  return name.slice(0, 1) === "J";
});

console.log(nameStartsWithJ); // "Jane"
```

Phương thức lặp của mảng

findIndex: Tương tự **find** nhưng kết quả trả về vị trí index của phần tử tìm được trong mảng.

Bài toán: Tìm phần tử trong mảng có chữ cái bắt đầu là chữ "J". Trả về chuỗi tìm được.

```
const names = ["Tom", "Frank", "Jane", "Henry", "Jimmy"];

const nameStartsWithJ = names.findIndex((name) => {
  return name.slice(0, 1) === "J";
});

console.log(nameStartsWithJ); // 2
```

Bài thực hành số 2

Bài thực hành số 2

Không sử dụng vòng lặp for, while.

Bài 1: Viết một hàm nhận tham số là một mảng gồm các giá trị chuỗi là các chữ cái. Hàm trả về một chuỗi gồm các phần tử của mảng tham số theo dạng “A, B, C”.

Bài 2: Viết một hàm nhận tham số là một mảng gồm các giá trị bất kỳ. Hàm kiểm tra xem trong mảng trên có giá trị nào thuộc kiểu Falsy không. Kết quả trả về true, false.

Bài 3: Viết một biến là một mảng gồm các object với 2 key là “firstName” và “lastName”. Giả sử mảng đó là các bộ hồ sơ lưu trữ theo tên người trong một tủ hồ sơ được chia theo chữ cái đầu trong tên theo “lastName”. Hàm kiểm tra xem tất cả các hồ sơ đã được đặt đúng tủ theo tên chưa. Kết quả trả về true, false.

Bài 4: Viết một hàm nhận tham số là một chuỗi gồm họ tên của một người. Hàm kiểm tra xem trong mảng của bài 3 có object nào có họ tên trùng với tham số hay không. Nếu có trả về object đó, nếu không trả về undefined.

Phương thức lặp của mảng

filter: Sử dụng để tìm các phần tử thỏa mãn một điều kiện nào đó. Sử dụng hàm call back để kiểm tra điều kiện, hàm trả về kết quả Truthy nếu thỏa mãn, Falsy với trường hợp ngược lại. Phương thức sẽ trả về một mảng mới chứa các phần tử thỏa mãn điều kiện trong hàm call back.

Bài toán: Tạo một mảng mới gồm các phần tử là số dương ở trong một mảng cho trước.

```
const numbers = [1, -2, 3, -4, 5];

const positiveNumbers = numbers.filter((number) => {
  return number >= 0;
});

console.log(positiveNumbers); // [1, 3, 5]
```

Phương thức lặp của mảng

map: Phương thức này sử dụng để tạo ra một mảng mới với các phần tử dựa trên mảng gọi phương thức. Giá trị trả về của call back là giá trị của các phần tử trong mảng mới được tạo ra.

Bài toán: Tạo ra một mảng có các phần tử có giá trị gấp đôi với phần tử của mảng cho trước.

```
const numbers = [1, 2, 3, 4, 5];

const doubledNumbers = numbers.map((number) => {
  return number * 2;
});

console.log(doubledNumbers); // [2, 4, 6, 8, 10]
```

Bài thực hành số 3

Bài thực hành số 3

Không sử dụng vòng lặp for, while.

Bài 1: Viết một hàm nhận 2 tham số, tham số thứ nhất là một mảng gồm các giá trị chuỗi, tham số thứ 2 là một số. Hàm lọc và trả về một mảng mới có các phần tử là các chuỗi có độ dài nhỏ hơn giá trị của tham số thứ 2.

Bài 2: Viết một biến là một mảng gồm các object gồm 2 key là “name” và “gender” (nhận 3 giá trị “male”, “female”, “other”). Viết một hàm nhận tham số là một chuỗi (là một trong 3 giá trị của “gender”), hàm tiến hành lọc và trả về một mảng mới các các object có giá trị của key “gender” giống với giá trị của tham số.

Bài 3: Từ mảng tạo ra ở bài 2, tạo một mảng mới gồm các object nhưng trong mỗi object có thêm 1 key là “id” với các giá trị không trùng nhau giữa các object (Có thể dùng random hoặc Date.now()).

Bài 4: Từ mảng mới tạo ra ở bài 3, tạo ra một mảng mới khác gồm các phần tử là các giá trị của key “id” trong các object trong mảng đó.

Phương thức lặp của mảng

reduce (Nâng cao): Phương thức này nhận 2 tham số, tham số thứ nhất là một hàm call back, tham số thứ 2 là một giá trị xuất phát. Call back của phương thức này tham số đầu tiên truy cập vào giá trị của giá trị xuất phát, tham số thứ hai là phần tử hiện tại của mỗi lần lặp (tương tự như các phương thức mảng khác). Dựa vào mỗi quan hệ của giá trị xuất phát và giá trị của phần tử mảng mỗi lần lặp để trả về giá trị mong muốn

Bài toán: Tính tổng các giá trị số có trong một mảng.

```
const numbers = [1, 2, 3, 4, 5];

const total = numbers.reduce((increment, currentNumber) => {
  return increment + currentNumber;
}, 0);

console.log(total); // 15
```

Bài toán: Từ dữ liệu dạng Entries trả về thành dạng object.

```
const entries = [
  ["name", "John"],
  ["age", 27],
];

const person = entries.reduce((object, currentItem) => {
  const [key, value] = currentItem;
  return { ...object, [`${key}`]: value };
}, {});

console.log(person); // { name: 'John', age: 27 }
```

Phương thức lặp của mảng

flat: Phương thức này sử dụng để làm phẳng các mảng có phần tử là một mảng (nested array). Phương thức này nhận tham số là một số và tương ứng sẽ là số lần nó “gỡ bỏ” các dấu ngoặc mảng ở trong mảng gọi phương thức này. Phương thức này trả về một mảng mới.

```
const nestedArray = [[1], [[2]], [[[3]]]];

nestedArray.flat(1); // [1, [2], [[3]]]
nestedArray.flat(2); // [1, 2, [3]]
nestedArray.flat(3); // [1, 2, 3]
```

Tham số của hàm Call back.

Hàm call back của một phương thức mảng nhận đầy đủ các tham số sau:

- Tham số 1: Phần tử mảng ở lần lặp hiện tại.
- Tham số 2: Vị trí index của phần tử đó.
- Tham số 3: Mảng đang sử dụng phương thức đó.

```
const students = ["Minh", "Tuấn", "Mai"];

students.forEach((currentItem, index, array) => {
  console.log(array); // ["Minh", "Tuấn", "Mai"]
  console.log(`Bạn số ${index + 1} là ${currentItem}`);
});
```

Chú ý: *Riêng hàm call back của phương thức reduce sẽ có tham số thứ nhất là giá trị xuất phát, 3 tham số tiếp theo tương tự như các phương thức khác.*

Method Chaining

Bài toán: Từ một mảng gồm các số cho trước, tạo một mảng mới gồm các phần tử là giá trị gấp đôi của các phần tử là giá trị dương trong mảng cho trước.

```
const numbers = [1, -2, 3, -4, 5];

const positiveNumbers = numbers.filter((num) => {
  return num > 0;
});

const doubledNumbers = positiveNumbers.map((posNum) => {
  return posNum * 2;
});
console.log(doubledNumbers); // [2, 6, 10]
```

```
const numbers = [1, -2, 3, -4, 5];

const result = numbers
  .filter((num) => {
    return num > 0;
  })
  .map((posNum) => {
    return posNum * 2;
  });
console.log(result); // [2, 6, 10]
```

Chú ý: *Phương thức nào có thể nối được tiếp còn tùy thuộc vào kiểu dữ liệu của phương thức trước trả về là gì. Ví dụ nếu phương thức trước trả về một chuỗi thì phương thức nối tiếp theo phải là phương thức sử dụng với chuỗi, tương tự với các kiểu dữ liệu khác.*

Arrow Function - Cú pháp return ngắn gọn

Với hàm viết bằng arrow function, khi phía sau => không phải là một block code mà là một giá trị thì tương đường với hàm return lại giá trị đó.

Ví dụ 1:

```
const sum = (a, b) => {
  return a + b;
};
```

```
const sum = (a, b) => a + b;
```

Ví dụ 2:

```
const numbers = [1, -2, 3, -4, 5];

const result = numbers.filter((num) => num > 0).map((posNum) => posNum * 2);
console.log(result); // [2, 6, 10]
```

Chú ý: Nếu kết quả trả về là một object thì kết quả trong hàm phải được bọc trong cặp ngoặc ( ).

```
const makeAnObject = (input) => ({ name: input });
```

Bài thực hành số 4

Bài thực hành số 4

Không sử dụng vòng lặp for, while.

Bài 1: Sử dụng mảng trong Bài 2 của bài thực hành số 3. Từ mảng đó tạo ra một chuỗi gồm tên của các bạn nữ theo dạng “Mai, Linh, Vân” và lưu chuỗi đó vào một biến.

Bài 2: Viết một hàm nhận tham số là một chuỗi là một đoạn văn bản. Hàm trả về một mảng gồm các phần tử là các từ trong đoạn văn trên mà có chữ “a” ở trong đó.

Sorting

Sorting

Sắp xếp giá trị chuỗi:

```
["c", "f", "a", "d", "b"].sort(); // ['a', 'b', 'c', 'd', 'f']
["c", "f", "a", "d", "b"].sort().reverse(); // ['f', 'd', 'c', 'b', 'a']

[23, 3, 2, 12, 1].sort(); // [1, 12, 2, 23, 3] ❌ Sai kết quả
```

Sắp xếp giá trị số:

```
[23, 3, 2, 12, 1].sort((a, b) => a - b); // [1, 2, 3, 12, 23]

[23, 3, 2, 12, 1].sort((a, b) => b - a); // [23, 12, 3, 2, 1]
// HOẶC
[23, 3, 2, 12, 1].sort((a, b) => a - b).reverse(); // [23, 12, 3, 2, 1]
```

Bài thực hành số 5

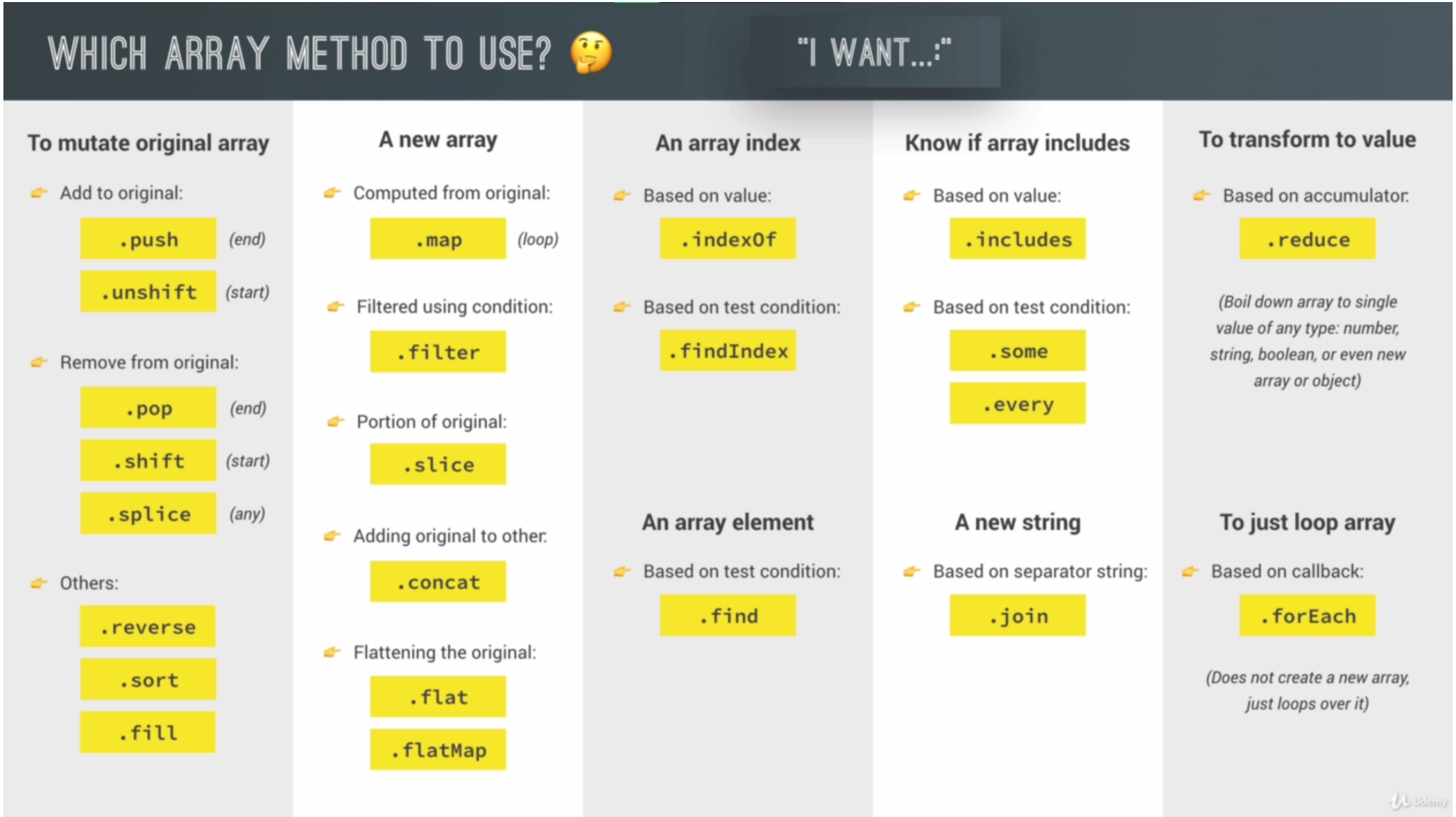
Bài thực hành số 5

Bài 1: Viết một hàm nhận 2 tham số, tham số thứ nhất là một mảng gồm các số, tham số thứ 2 là một số. Hàm trả về số lớn nhất theo tham số thứ 2. Ví dụ tham số thứ 2 là 2, hàm trả về số lớn thứ hai; tham số thứ 2 là 3, hàm trả về số lớn thứ ba... Trả về undefined nếu tham số 2 không hợp lệ.

Bài 2: Giả sử bạn có một danh sách tên danh bạ ngẫu nhiên. Bạn cần tổng hợp các cái tên này vào một cuốn sổ chia theo danh mục theo chữ cái đầu tiên. Viết một hàm nhận tham số là danh sách tên kia theo dạng mảng, hàm trả về tên danh mục cuối cùng theo mảng đã nhận.

Bài 3 (Nâng cao): Viết một hàm nhận 2 tham số, tham số 1 là một mảng gồm các số, tham số 2 là một số. Hàm kiểm tra xem trong mảng ở tham số một có tồn tại 2 phần tử có tổng bằng tham số thứ 2 hay không. Nếu có trả về mảng có 2 số đó, nếu không có trả về null.

Array Methods Cheat Sheet



Hoàn thành JavaScript – Buổi 5

Good job!