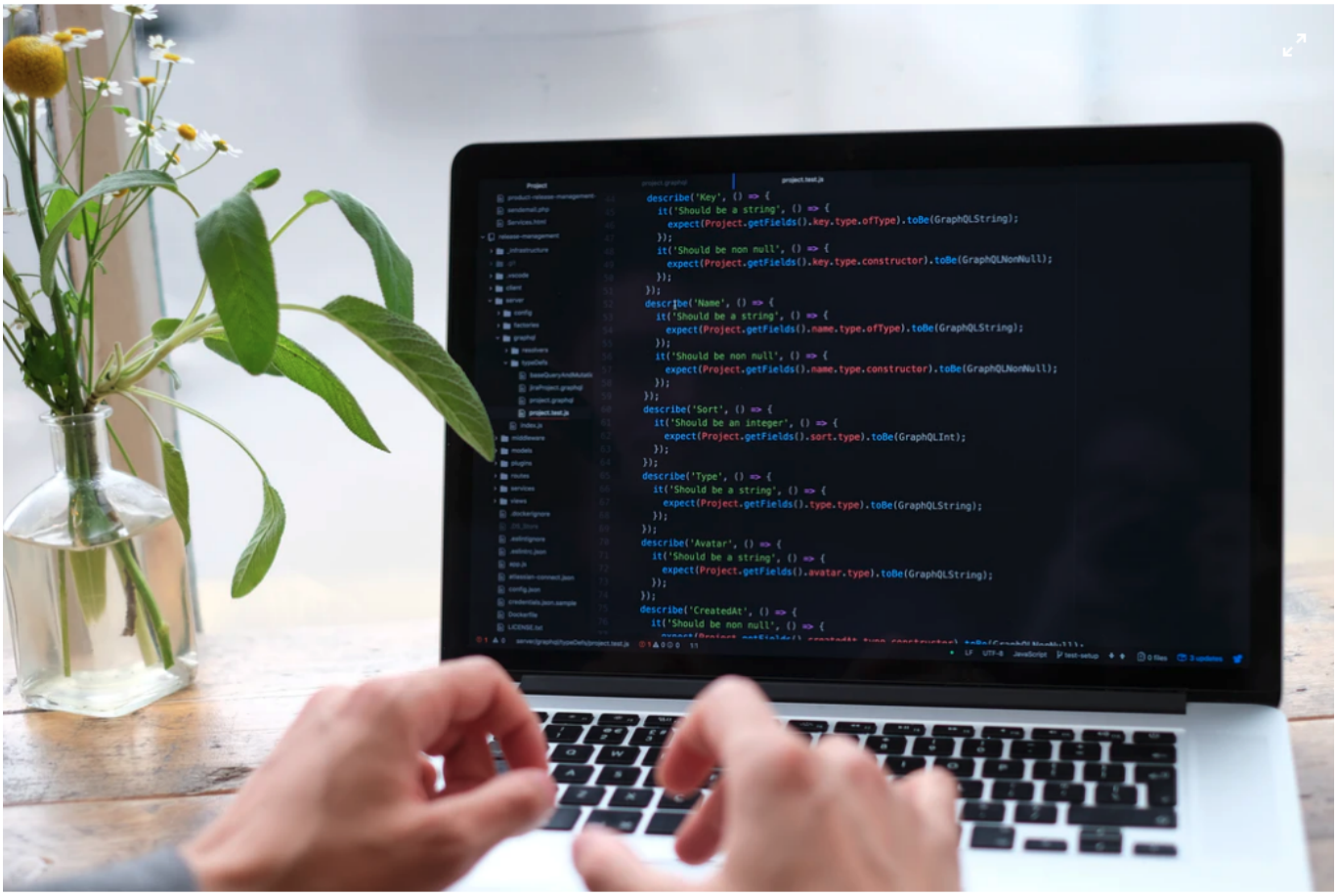
Tổng hợp các bài tập lập trình cơ bản



Nguồn tham khảo:

- 1000 bài tập của thầy Khang
- Edabit
- LeetCode cùng một số nguồn tham khảo khác.

AGENDA

1. Cách làm bài tập trong file pdf này
2. Xử lý số (number)
3. Xử lý chuỗi (string)
4. Xử lý mảng (array): kiểm tra phần tử có trong mảng không?
5. Xử lý mảng (array): tìm kiếm phần tử
6. Xử lý mảng (array): liệt kê
7. Xử lý mảng (array): đếm
8. Xử lý mảng (array): tính tổng
9. Xử lý mảng (array): thêm / xoá phần tử
10. Xử lý mảng (array): mảng con
11. Bài tập tổng hợp luyện tư duy

LƯU Ý: TÀI LIỆU NÀY CHỈ LƯU HÀNH NỘI BỘ

DANH MỤC BÀI TẬP

1. Cách làm bài tập trong file pdf này

#	Step	Desc
1	Pick an exercise	Chọn một bài tập
2	Create two files	Tạo một file `.js` để code và một file `.test.js` để viết unit test. Hoặc lười thì viết code vào file unit test luôn 😅
3	Write unit test	Đọc hiểu function và xác định mong muốn, từ đó định nghĩa các test cases tương ứng
5	Coding	Viết code để cover tất cả test cases
4	Check unit test + fix bugs	Sau khi code xong thì check unit test, nếu pass hết thì okie, ko thì code tiếp 🤣

Cần lưu ý là phải handle cả những trường hợp tham số truyền vào không hợp lệ nhé, đừng chỉ làm happy cases nhen hehe

> Lưu ý: Với mỗi bài tập, hãy cố gắng viết bằng nhiều cách nhất có thể.

Để dễ dàng trao đổi và tham chiếu, mình sẽ đặt mã số cho mỗi bài tập:

- **NUMBER-X**: bài tập về số
- **STRING-X**: bài tập về chuỗi
- **ARRAY-CATEGORY-X**: bài tập về mảng
- **LOGIC-X**: bài tập tổng hợp luyện logic

Trong đó:

- `CATEGORY` là các chủ đề khác nhau trong xử lý mảng.
- `X` là số thứ tự 1, 2, 3, ...

2. Xử lý số (number)

NUMBER-01: Kiểm tra số tăng dần

Viết hàm **isIncreasingNumber(n)** để kiểm tra **n** có phải là số tăng dần hay không?

- 0 < n < 1000000
- Số tăng dần có ít nhất 2 chữ số
- Chữ số bên phải luôn lớn hơn chữ số bên trái

Trả về true nếu là số tăng dần, ngược lại trả về false.

Ví dụ:

- isIncreasingNumber(11) --> false
- isIncreasingNumber(123) --> true
- isIncreasingNumber(12321) --> false

NUMBER-02: Kiểm tra số giảm dần

Viết hàm **isDecreasingNumber(n)** để kiểm tra **n** có phải là số giảm dần hay không?

- 0 < n < 1000000
- Số giảm dần có ít nhất 2 chữ số
- Chữ số bên phải luôn nhỏ hơn chữ số bên trái

Trả về true nếu là số giảm dần, ngược lại trả về false.

Ví dụ:

- isDecreasingNumber(11) --> false
- isDecreasingNumber(321) --> true
- isDecreasingNumber(12321) --> false

NUMBER-03: Kiểm tra số tăng dần đều theo khoảng cách cho trước

Viết hàm **isIncreasingNumberByDistance(n, x)** để kiểm tra **n** có phải là số tăng dần đều với khoảng cách giữa 2 chữ số là **x** không?

- 0 < n < 1000000, 0 < x < 5
- Số tăng dần có ít nhất 2 chữ số
- Chữ số bên phải luôn lớn hơn chữ số bên trái
- Khoảng cách của 2 chữ số liền kề là **x**

Trả về true nếu là số tăng dần đều theo khoảng cách x, ngược lại trả về false.

Ví dụ:

- isIncreasingNumberByDistance(11, 1) --> false
- isIncreasingNumberByDistance(123, 1) --> true
- isIncreasingNumberByDistance(135, 2) --> true
- isIncreasingNumberByDistance(135, 1) --> false

NUMBER-04: Kiểm tra số nguyên tố

Viết hàm isPrime(n) để kiểm tra n có phải là số nguyên tố không?

- Số nguyên tố là số chỉ chia hết cho 1 và chính nó.
- 0 < n < 100000

Trả về true nếu là số nguyên tố, ngược lại trả về false.

Ví dụ:

- isPrime(3) --> true
- isPrime(4) --> false
- isPrime(11) --> true

NUMBER-05: Kiểm tra số chính phương

Viết hàm isPerfectSquare(n) để kiểm tra n có phải là số chính phương không?

- 0 < n < 100000
- Số chính phương là số có căn bậc 2 của nó là một số tự nhiên.
- Trả về true nếu là số chính phương, ngược lại trả về false.

Ví dụ: 4, 9, 25, 36 là các số chính phương.

NUMBER-06: Kiểm tra số hoàn hảo

Viết hàm **isPerfectNumber(n)** để kiểm tra n có phải là số hoàn hảo hay không?

- Với n thoả điều kiện 1 < n < 1000
- Số hoàn hảo là số mà tổng của tất cả ước số (không tính chính nó, tức từ 1 đến n - 1) bằng chính nó.
- Trả về true nếu đúng, ngược lại trả về false

Ví dụ: 6 = 1 + 2 + 3 (như vậy 6 là một số hoàn hảo)

> Gợi ý: không nhất thiết phải chạy tới (n - 1) để tìm ra tất cả các ước số của n

---

NUMBER-07: Kiểm tra số đối xứng

Viết hàm **isSymetricNumber(n)** để kiểm tra n có phải là số đối xứng hay không?

- Với n thoả điều kiện 1 < n < 1000000
- Số đối xứng là số đọc từ trái sang phải cũng giống như đọc từ phải sang trái.
- Trả về true nếu đúng, ngược lại trả về false

Ví dụ:

- isSymetricNumber(10) --> false
- isSymetricNumber(11) --> true
- isSymetricNumber(12321) --> true

NUMBER-08: Kiểm tra số có tổng chữ số chia hết cho 10

Viết hàm **isDivisibleBy10(n)** để kiểm tra n (0 < n < 1000000) có tổng chữ số chia hết cho 10 hay không?

Ví dụ:

- isDivisibleBy10(10) --> false
- isDivisibleBy10(1234) --> true vì 1 + 2 + 3 + 4 = 10 chia hết cho 10
- isDivisibleBy10(123455) --> true vì 1 + 2 + 3 + 4 + 5 + 5 = 20 chia hết cho 10

NUMBER-09: Kiểm tra số có tổng của 2 chữ số bằng số cho trước

Viết hàm **hasTwoDigitsWithSum(n, sum)** để kiểm tra **n** (9 < n < 1000000) có chứa **2 chữ số bất kỳ** nào có tổng bằng **sum** (0 < sum < 19) cho trước không?

Ví dụ:

- hasTwoDigitsWithSum(10, 1) --> true vì 0 + 1 = 1
- hasTwoDigitsWithSum(101, 3) --> false vì không có 2 chữ số nào cộng lại bằng 3
- hasTwoDigitsWithSum(11, 2) --> true

---

3. Xử lý chuỗi (string)

STRING-01: Đếm số từ có trong câu

Viết hàm **countWords(str)** để đếm số từ có trong **str**.

Ví dụ: countWords('easy frontend') --> 2

STRING-02: Thống kê số lượng từ trong câu

Viết hàm statisticsWords(str) để đếm tần số xuất hiện của môi từ trong str.

Trả về **một object** với:

- **key** là từ có xuất hiện trong str
- **value** là số lần xuất hiện của **key**

Ví dụ:

```
statisticsWords('easy frontend easy')
// should return an object like this:
// { easy: 2, frontend: 1 }
```

STRING-03: Thống kê ký tự có trong câu

Viết hàm **statisticsCharacters(str)** để thống kê số lượng ký tự có trong câu.

Trả về **một object** với:

- **key** là character có xuất hiện trong str. Nếu là khoảng trắng thì key="space".
- **value** là số lần xuất hiện của **key**

Ví dụ:

```
statisticsCharacters('aa b cc ')
// should return an object like this:
// { a: 2, b: 1, c: 2, space: 3 }
```

STRING-04: Đếm số lượng email có trong câu

Viết hàm **countEmails(str)** để đếm số lượng email có trong **str**

- Email là một chuỗi có chứa ký tự @ ở giữa và không phải ở cuối từ.
- Email có thể có một số domain như: `.com`, `.vn`, `.com.vn`. Domain sẽ xuất hiện ở cuối từ.
- Email có định dạng là **X@Y**. Trong đó **X** có độ dài ít nhất 3 ký tự, **Y** có độ dài ít nhất 3 ký tự (ko tính phần domain nêu trên)

Trả về số lượng email tìm thấy từ chuỗi truyền vào.

> Lưu ý: không sử dụng regular expression.

Ví dụ:

- countEmails('my email should be abc@super.com') --> 1
- countEmails('my email should be a@a.com') --> 0
- countEmails('my email should be @ bla .com') --> 0
- countEmails('my email should be easy@frontend.com or easy@frontend.com.vn or easy@frontend.vn') --> 3

STRING-05: Đếm số lượng url có trong câu

Viết hàm **countURLs(str)** để đếm số lượng URLs có trong câu.

- URL có định dạng: protocol://domain.com/path-name
- URL có thể bắt đầu bằng protocol như: http, https, ws hoặc wss
- Domain có thể là .com, .com.vn hoặc .vn
- Phần tên domain phải có ít nhất 3 ký tự, như abc.com, chứ `ab.com` là không hợp lệ

Trả về số lượng URLs tìm thấy trong câu.

> Lưu ý: không sử dụng regular expression.

Ví dụ:

- countURLs('my website is: http://ezfrontend.com you can visit it') --> 1
- countURLs('my website is: https://ez.com you can visit it') --> 0 vì phần tên domain chỉ có 2 ký tự `ez` nên không hợp lệ

STRING-06: Hiển thị địa chỉ người dùng

Viết hàm **getDisplayedAddress(address)** để ghép chuỗi từ các thuộc tính của object **address** thành một chuỗi address hoàn chỉnh

- Chuỗi trả về có dạng: "address.number address.street, address.ward, address.district, address.city"
- Lưu ý object **address** không phải lúc nào cũng có đầy đủ các thông tin trên.

Ví dụ:

```
getDisplayedAddress({
  number: 123,
  street: 'Nguyen Cong Tru',
  ward: 'Ward 11',
  district: 'Binh Thanh District',
  city: 'HCMC'
});
// should return '123 Nguyen Cong Tru, Ward 11, Binh Thanh District, HCMC'

getDisplayedAddress({
  street: 'Nguyen Cong Tru',
  district: 'Binh Thanh District'
});
// should return 'Nguyen Cong Tru, Binh Thanh District'
```

> Lưu ý: để ý dấu phẩy và khoảng trắng.

STRING-07: Thay thế params thành value tương ứng

Viết hàm **fillPath(path, params)** để thay thế các chuỗi params trong **path** bằng các giá trị tương ứng trong object **params**.

Ví dụ:

```
fillPath('/products/:productId', { productId: 123 }); // '/products/123'

fillPath('/categories/:categoryId/products/:productId', {
  categoryId: 1,
  productId: 2
});
// should return '/categories/1/products/2'

fillPath('/categories/:categoryId/products/:productId', { productId: 2 });
//  should return '/categories/:categoryId/products/2'
```

4. Xử lý mảng (array): kiểm tra phần tử có trong mảng không?

ARRAY-CHECK-01: Kiểm tra có số chẵn lớn hơn n trong mảng không

Viết hàm **hasEvenNumberGreaterThanN(numberList, n)** để kiểm tra trong mảng số **numberList** truyền vào có số chẵn mà lớn hơn **n** hay không?

```
hasEvenNumberGreaterThanN([2, 3, 4], 5); // false
hasEvenNumberGreaterThanN([2, 3, 4, 6], 5); // true
```

ARRAY-CHECK-02: Kiểm tra có số lẻ và chia hết cho 3 không

Viết hàm **hasOddNumberDivisibleBy3(numberList)** để kiểm tra trong mảng numberList có chứa số lẻ nào chia hết cho 3 không?

```
hasOddNumberDivisibleBy3([1, 5, 7]); // false
hasOddNumberDivisibleBy3([1, 6]); // true
```

ARRAY-CHECK-03: Kiểm tra có từ easy frontend trong mảng không

Viết hàm **hasEasyFrontend(wordList)** để kiểm tra xem nội dung tất cả các từ có chứa **easy** và **frontend** không nhé

Trả về `true` nếu:

- Một phần tử có chứa cả 2 từ: "easy" và "frontend"
- Một phần tử chứa từ "easy" và một phần tử khác chứa "frontend"

Ngược lại trả về `false`.

```
hasEasyFrontend(['easy', '']); // false
hasEasyFrontend(['easy', 'frontend']); // true
hasEasyFrontend(['easy frontend', '']); // true
```

ARRAY-CHECK-04: Kiểm tra có giá trị truthy nào trong mảng không

Viết hàm **hasTruthy(arr)** để kiểm tra xem trong mảng **arr** có giá trị **truthy** không?

```
hasTruthy([]); // false
hasTruthy([false, '']); // false
hasTruthy([false, 123]); // true
```

ARRAY-CHECK-05: Kiểm tra có giá trị falsy nào trong mảng không

Viết hàm **hasFalsy(arr)** để kiểm tra xem trong mảng **arr** có giá trị **falsy** không?

```
hasFalsy([]); // false
hasFalsy([false, '']); // true
hasFalsy([false, 123]); // true
hasFalsy(['easy', 123]); // false
```

ARRAY-CHECK-06: Kiểm tra có student có id nào đó trong mảng không

Viết hàm **hasStudent(studentList, studentId)** để kiểm tra trong mảng **studentList** có student nào có id là **studentId** không?

```
const studentList = [
  { id: 1, name: 'Easy' },
  { id: 2, name: 'Frontend' },
];
hasStudent(studentList, 1); // true
hasStudent(studentList, 3); // false
```

ARRAY-CHECK-07: Kiểm tra có student giới tính là Nữ tên Alice không

Viết hàm **hasAlice(studentList)** để kiểm tra trong mảng **studentList** có student nữ nào tên là Alice không? (không phân biệt hoa thường)

```
const studentList = [
  { id: 1, name: 'Alice', gender: 'male' },
  { id: 2, name: 'aliCE', gender: 'female' },
  { id: 3, name: 'Easy Frontend', gender: 'male' },
];
hasAlice(studentList); // true
```

ARRAY-CHECK-08: Kiểm tra có sản phẩm nhỏ hơn giá cho trước và free ship không

Viết hàm **hasFreeShip(productList, price)** để kiểm tra xem có product nào freeship và giá nhỏ hơn **price** không?

```
const productList = [
  { id: 1,  name: 'Iphone 16', isFreeShip: false, price: 10200500 },
  { id: 2,  name: 'Iphone 16 Pro Max', isFreeShip: true, price: 1500000 },
];
hasFreeShip(productList, 10000000); // true vì id số 2 thoả điều kiện đề bài
```

ARRAY-CHECK-09: Kiểm tra mảng có tăng dần không

Viết hàm **isIncreasingNumberList(numberList)** để kiểm tra xem mảng số này có phải tăng dần không?

Mảng tăng dần cần thoả các điều kiện sau:

1. Có ít nhất 2 phần tử.
2. Phần tử sau luôn lớn hơn phần tử trước nó.

```
isIncreasingNumberList([1, 1]); // false
isIncreasingNumberList([1, 2]); // true
isIncreasingNumberList([2, 1]); // false
```

ARRAY-CHECK-10: Kiểm tra mảng có giảm dần không

Viết hàm **isIncreasingNumberList(numberList)** để kiểm tra xem mảng số này có phải giảm dần không?

Mảng tăng dần cần thoả các điều kiện sau:

1. Có ít nhất 2 phần tử.
2. Phần tử sau luôn nhỏ hơn phần tử trước nó.

```
isIncreasingNumberList([1, 1]); // false
isIncreasingNumberList([1, 2]); // false
isIncreasingNumberList([2, 1]); // true
```

ARRAY-CHECK-11: Kiểm tra mang có đối xứng không

Viết hàm **isSymmetricList(numberList)** để kiểm tra mảng có tính chất đổi xứng không?

- Mảng đối xứng là mảng đọc từ trái qua phải giống như đọc từ phải sang trái.
- Mảng đối xứng phải có ít nhất một phần tử.

```
isSymmetricList([]); // false
isSymmetricList([1]); // true
isSymmetricList([1, 2, 2, 1]); // true
isSymmetricList([1, 2, 3]); // false
```

ARRAY-CHECK-12: Kiểm tra mảng có số fibonaci nhỏ hơn 100 không?

Viết hàm **hasFibonaciNumber(numberList)** để kiểm tra xem trong mảng **numberList** có chứa một số fibonaci nhỏ hơn 100 không?

- Việc đầu tiên là viết hàm để generate ra cái list fibonaci numbers nhỏ hơn 100. Nếu bạn chưa biết fibonaci là gì thì đọc ở đây nhé
- Việc tiếp theo là kiểm tra trong mảng đầu vào có chứa con số fibonaci nhỏ hơn 100 không? Trả về true nếu có, ngược lại trả về false.

```
hasFibonaciNumber([4, 6, 7]); // false
hasFibonaciNumber([0, 1, 2, 3, 4, 5]); // true
hasFibonaciNumber([89]); // true
```

5. Xử lý mảng (array): tìm kiếm phần tử

ARRAY-FIND-01: Tìm số dương nhỏ nhất trong mảng

Viết hàm **findMinPositive(numberList)** để tìm ra số dương nhỏ nhất trong mảng.

```
findMinPositive([]); // undefined
findMinPositive([-1, -5]); // undefined
findMinPositive([-1, -5, 2, 6]); // 2
```

ARRAY-FIND-02: Tìm từ dài nhất nhưng nhỏ hơn 5 trong mảng

Viết hàm **findLongestWord(wordList)** để tìm ra từ dài nhất trong mảng **wordList**

- Từ cần tìm có độ dài không quá 5
- Từ cần tìm không có chứa khoảng trắng (space)
- Trường hợp không tìm thấy từ thoả điều kiện thì trả về empty string.

```
findLongestWord([]); // ''
findLongestWord(['super', 'cool']); // 'super'

findLongestWord(['super', 'super cool']);
// 'super' vì từ 'super cool' có chứa khoảng trắng và có độ dài lớn hơn 5
nên không tính
```

ARRAY-FIND-03: Tìm số dương chẵn đầu tiên trong mảng

Viết hàm **findFirstPositiveEven(numberList)** để tìm ra số dương chẵn đầu tiên trong mảng **numberList**

```
findFirstPositiveEven([1, 4, 5]); // 4
findFirstPositiveEven([1, 3, 5]); // undefined vì không có số dương chẵn
nào
```

ARRAY-FIND-04: Tìm số âm lẻ cuối cùng trong mảng

Viết hàm **findLastNegativeOdd(numberList)** để tìm ra số âm lẻ cuối cùng trong mảng **numberList**

```
findLastNegativeOdd([-1, -3, -5]); // -5 vì 5 là số âm lẻ cuối cùng
findLastNegativeOdd([1, 3, 5]); // undefined vì không có số âm lẻ nào
trong mảng
```

ARRAY-FIND-05: Tìm số lớn thứ 2 trong mảng

Viết hàm **findSecondLargestNumber(numberList)** để tìm ra số lớn thứ 2 trong mảng **numberList**

```
findSecondLargestNumber([1]); // undefined
findSecondLargestNumber([1, 2]); // 1
findSecondLargestNumber([1, 2, 3, 4]); // 3
```

ARRAY-FIND-06: Tìm số chính phương cuối cùng trong mảng

Viết hàm **findLastPerfectSquare(numberList)** để tìm ra số chính phương cuối cùng trong mảng **numberList**

```
findLastPerfectSquare([2, 3, 5]) // undefined
findLastPerfectSquare([4, 16, 25, 36, 40]) // 36
```

ARRAY-FIND-07: Tìm email đầu tiên trong mảng

Viết hàm **findFirstEmail(strList)** để tìm ra email hợp lệ đầu tiên có trong mảng **strList**

Điều kiện của một email hợp lệ:

- Email là một chuỗi có chứa ký tự @ ở giữa và không phải ở cuối từ.
- Email có thể có một số domain như: `.com`, `.vn`, `.com.vn`. Domain sẽ xuất hiện ở cuối từ.
- Email có định dạng là **X@Y**. Trong đó **X** có độ dài ít nhất 3 ký tự, **Y** có độ dài ít nhất 3 ký tự (ko tính phần domain nêu trên)
- Trường hợp không tìm thấy email thoả yêu cầu thì trả về undefined.

```
findFirstEmail(['abc@easy.frontend', 'abc@gmail.com']); // 'abc@gmail.com'
findFirstEmail(['abc@easy.frontend']); // undefined
```

ARRAY-FIND-08: Tìm url cuối cùng trong mảng

Viết hàm **findLastUrl(strList)** để tìm ra URL cuối cùng có trong mảng **strList**

Điều kiện cho một URL hợp lệ:

- URL có định dạng: protocol://domain.com/path-name
- URL có thể bắt đầu bằng protocol như: http, https, ws hoặc wss
- Domain có thể là .com, .com.vn hoặc .vn
- Phần tên domain phải có ít nhất 3 ký tự, như abc.com, chứ `ab.com` là không hợp lệ

```
findLastUrl([]); // undefined
findLastUrl(['https://google.com', 'wss://chat.sample.com']);
// 'wss://chat.sample.com'
```

ARRAY-FIND-09: Tìm vị trí của student có id cho trước

Viết hàm **findStudentById(studentList, studentId)** để tìm ra vị trí của student có id bằng với **studentId**.

```
const studentList = [
  { id: 1, name: 'Easy' },
  { id: 2, name: 'Frontend' },
]
findStudentById(studentList, 1); // 0
findStudentById(studentList, 3); // -1
```

ARRAY-FIND-10: Tìm vị trí của sản phẩm có mã code là = abc

Viết hàm **findProductByCode(productList, code)** để tìm ra vị trí product có mã sản phẩm bằng với **code**.

```
const productList = [
  { id: 1, code: 'ip01', name: 'IPhone 16' },
  { id: 2, code: 'ip02', name: 'IPhone 16 Promax' },
]
findProductByCode(productList, 'ip01'); // 0
findProductByCode(productList, 'ip03'); // -1
```

ARRAY-FIND-11: Tìm student có điểm môn toán lớn nhất đầu tiên

Viết hàm **findStudentHavingHighestMark(studentList)** để tìm ra student để điểm môn toán cao nhất.

```
const studentList = [
  { id: 1, name: 'Alice', math: 9 },
  { id: 2, name: 'Bob', math: 10 },
  { id: 3, name: 'John', math: 10 },
];
findStudentHavingHighestMark(studentList); // { id: 2, name: 'Bob', math:
10 }
```

ARRAY-FIND-12: Tìm student có điểm trung bình môn thấp nhất cuối cùng

Viết hàm findLastStudentHavingMinAvg(studentList) để tìm ra student có điểm trung bình môn thấp nhất cuối cùng trong mảng.

```
const studentList = [
  { id: 1, name: 'Alice', math:  9, english: 9 },
  { id: 2, name: 'Bob', math: 5, english: 5 },
  { id: 3, name: 'John', math: 5, english: 5 },
];
findStudentHavingHighestMark(studentList);
// { id: 3, name: 'John', math: 5, english: 5 }
```

ARRAY-FIND-13: Tìm student đầu tiên có tất cả điểm >= 5, trừ một môn

Viết hàm **findStudent(studentList)** để tìm ra student đầu tiên có trong mảng có tất cả các điểm đều lớn hơn hoặc bằng 5, tuy nhiên có một môn dưới 5.

```
const studentList = [
  { id: 1, name: 'Alice', marks: { math: 9, english: 9, programming: 5 }
},
  { id: 2, name: 'Bob', marks: { math: 2, english: 3, programming: 5 } },
  { id: 3, name: 'John', marks: { math: 4, english: 7, programming: 9 } },
  { id: 4, name: 'Sarah', marks: { math: 2, english: 8, programming: 10 }
},
];
findStudent(studentList);
// { id: 3, name: 'John', marks: { math: 4, english: 7, programming: 9 } }
// vì đây là student đầu chỉ có duy nhất một môn dưới 5 điểm
```

ARRAY-FIND-14: Tìm sản phẩm đầu tiên có free ship

Viết hàm **findProductHavingFreeShip(productList)** để tìm ra sản phẩm đầu tiên có free ship.

```
const productList = [
  { id: 1, name: 'IPhone 5', isFreeShip: false },
  { id: 2, name: 'IPhone X', isFreeShip: true },
  { id: 3, name: 'IPhone 12 Pro', isFreeShip: true },
];
findProductHavingFreeShip(productList);
// { id: 2, name: 'IPhone X', isFreeShip: true }
```

ARRAY-FIND-15: Tìm sản phẩm đầu tiên có tên bắt đầu bằng chữ iphone (không phân biệt hoa thường)

Viết hàm **findFirstIPhone(productList)** để tìm ra sản phẩm dòng IPhone đầu tiên

```
const productList = [
  { id: 1, name: 'Samsung' },
  { id: 2, name: 'IPHONE X' },
  { id: 3, name: 'iPhone 12 Pro' },
];
findFirstIPhone(productList);
// { id: 2, name: 'IPHONE X' }
```

6. Xử lý mảng (array): liệt kê

ARRAY-FILTER-01: Liệt kê các số trong khoảng [a, b]

Viết hàm **generateNumberInRange(a, b)** để tạo ra một mảng các số từ a đến b (có bao gồm a và b)

```
generateNumberInRange(1, 5); // [1, 2, 3, 4, 5]
generateNumberInRange(5, 7); // [5, 6, 7]
generateNumberInRange(7, 5); // []
```

ARRAY-FILTER-02: Tìm tất cả các số mà nó lớn hơn số đứng trước nó

Viết hàm **findNumbers(numberList)** để tìm ra các số thoả điều kiện là **lớn hơn số đứng trước nó** (không tính số đầu tiên)

```
findNumbers([1]); // []
findNumbers([2, 5, 3, 7]); // [5, 7] vì 5 > 2 và 7 > 3
```

ARRAY-FILTER-03: Dãy số fibonaci nhỏ hơn n

Viết hàm **generateFibonaci(n)** để tạo ra mảng các số fibonaci nhỏ hơn n (0 < n < 1000)

Fibonaci là dãy số tự nhiên có 2 phần tử đầu tiên là 0 và 1.

Phân tử tiếp theo sẽ bằng tổng 2 phần tử trước đó:

```
f(n) = f(n - 1) + f(n - 2)
```

```
generateFibonaci(5); // [0, 1, 1, 2, 3]
generateFibonaci(10); // [0, 1, 1, 2, 3, 5, 8]
```

ARRAY-FILTER-04: Dãy số nguyên tố nhỏ hơn n

Viết hàm **generatePrimeNumbers(n)** để tạo ra mảng các số nguyên tố nhỏ hơn n (0 < n < 1000)

```
generatePrimeNumbers(10); // [2, 3, 5, 7]
generatePrimeNumbers(20); // [2, 3, 5, 7, 11, 13, 17, 19]
```

ARRAY-FILTER-05: Tìm tất cả các số bằng với số dương chẵn đầu tiên trong mảng

Viết hàm **findAllNumbers(numberList)** để tìm ra tất cả các số bằng với số dương chẵn đầu tiên trong mảng.

- Trường hợp mảng không có số dương chẵn thì trả về mảng rỗng.
- Mảng kết quả không bao gồm số dương chẵn đầu tiên.

```
findAllNumbers([1, 3, 5]); // []
findAllNumbers([1, 2, 5]);
// [] bởi vì có duy một số dương chẵn đầu tiên, mà số này không bao gồm
trong mảng kết quả

findAllNumbers([1, 4, 5, -6, 4, 5, 4]);
// [4, 4] vì có 3 số 4, bỏ đi một số đầu tiên thì còn lại 2 số
```

ARRAY-FILTER-06: Tìm tất cả các số mà bắt đầu bằng chữ số lẻ

Viết hàm **findAllNumbers(numberList)** để tìm ra tất cả các số có chữ số đầu tiên là chữ số lẻ.

```
findAllNumbers([1, 5, 6]); // [1, 5]
findAllNumbers([234, 421, 123]); // [123]
```

ARRAY-FILTER-07: Tìm tất cả student có điểm toán nhỏ hơn 5

Viết hàm **findAllStudents(studentList)** để tìm ra tất cả student có điểm toán nhỏ hơn 5

```
const studentList = [
  { id: 1, name: 'Alice', math: 9 },
  { id: 2, name: 'Bob', math: 4 },
  { id: 3, name: 'John', math: 0 },
];
findAllStudents(studentList);
// should return the following array:
// [
//   { id: 2, name: 'Bob', math: 4 },
//   { id: 3, name: 'John', math: 0 },
// ]
```

ARRAY-FILTER-08: Tìm tất cả sản phẩm có hỗ trợ free ship

Viết hàm **findAllProducts(productList)** để tìm ra tất cả product có **isFreeShip=true**.

```
const productList = [
  { id: 1, name: 'iphone X', isFreeShip: true },
  { id: 2, name: 'iphone 16 Pro', isFreeShip: true },
  { id: 3, name: 'iphone 20 Pro', isFreeShip: false },
];
findAllProducts(productList);
// should return the following list:
// [
//   { id: 1, name: 'iphone X', isFreeShip: true },
//   { id: 2, name: 'iphone 16 Pro', isFreeShip: true },
// ]
```

ARRAY-FILTER-09: Tìm tất cả sản phẩm thuộc dòng iphone và còn hàng trong kho

Viết hàm **findAllIphones(productList)** để tìm ra tất cả các product có tên chứa chữ iphone (ko phân biệt hoa thường) và vẫn còn hàng trong kho (amount > 0)

```
const productList = [
  { id: 1, name: 'Luxury IPhone X', amount: 1 },
  { id: 2, name: 'Super Cool iphone 16 Pro', amount: 0 },
  { id: 3, name: 'iphone 20 Pro', amount: 2 },
];
findAllIphones(productList);
// should return the following list:
// [
//   { id: 1, name: 'Luxury IPhone X', amount: 1 },
//   { id: 3, name: 'iphone 20 Pro', amount: 2 },
// ];
```

---

7. Xử lý mảng (array): đếm

ARRAY-COUNT-01: Đếm số dương chẵn có trong mảng

Viết hàm **countPositiveEvenNumbers(numberList)** để đếm có bao nhiêu số dương chẵn có trong mảng.

```
countPositiveEvenNumbers([-2, -1]); // 0
countPositiveEvenNumbers([-2, -1, 1, 2, 4]); // 2 (2 và 4 thoả điều kiện)
```

ARRAY-COUNT-02: Đếm số mà nó nhỏ hơn số liền trước của nó

Viết hàm **countNumbers(numberList)** để đếm xem trong mảng có bao nhiêu số mà nó nhỏ hơn số đừng liền trước của nó.

```
countNumbers([1, 2, 3]); // 0
countNumbers([1, 2, 3, 10, 9, 8]); // 2 (9 và 8 thoả điều kiện)
```

ARRAY-COUNT-03: Đếm số từ có độ dài lớn hơn n

Viết hàm **countWords(wordList, n)** để đếm xem có bao nhiêu từ có độ dài lớn hơn hoặc bằng **n**

```
countWords(['easy', 'frontend'], 4); // 2
countWords(['easy', 'frontend'], 5); // 1
countWords(['easy', 'frontend'], 10); // 0
```

ARRAY-COUNT-04: Đếm số lượng các số khác nhau có trong mảng

Viết hàm **countUniqueNumbers(numberList)** để tìm ra có bao nhiêu số khác nhau trong mảng.

```
countUniqueNumbers([]); // 0
countUniqueNumbers([1, 1, 1]); // 1
countUniqueNumbers([1, 2, 3]); // 3
countUniqueNumbers([1, 2, 2, 3, 1]); // 3
```

ARRAY-COUNT-05: Đếm số lượng các số có trong mảng a mà không có trong mảng b

Viết hàm **countNumbersNotInB(a, b)** để đếm có bao nhiêu số xuất hiện trong mảng **a**, mà không có trong mảng **b**

```
countNumbersNotInB([1, 2, 3], [4, 5]); // 3
countNumbersNotInB([1, 2, 3], [1, 2, 3]); // 0
countNumbersNotInB([1, 2, 3], [3, 4, 5]); // 2
```

ARRAY-COUNT-06: Đếm số lượng students có điểm trung bình lớn hơn hoặc bằng avgMark

Viết hàm **countStudents(studentList, avgMark)** để đếm số lượng student có điểm trung bình môn lớn hơn hoặc bằng **avgMark**

Cách tính điểm trung bình môn:

- Giả sử mình không biết trước sẽ có bao nhiêu môn
- Dựa vào object **marks** để xác định, có bao nhiêu keys là bấy nhiêu môn và tính điểm trung bình môn tương ứng.
- Một student list sẽ đảm bảo là số lượng keys trong object marks sẽ giống nhau.

```
const studentList = [
  { id: 1, name: 'Alice', marks: { math: 8 } },
  { id: 2, name: 'Bob', marks: { math: 9 } },
]
countStudents(studentList, 7); // 2
```

```
const studentList = [
  { id: 1, name: 'Alice', marks: { math: 5, english: 6 } },
  { id: 2, name: 'Bob', marks: { math: 9, english: 8 } },
]
countStudents(studentList, 7); // 1
```

---

8. Xử lý mảng (array): tính tổng

ARRAY-SUM-01: Tổng các số chẵn mà nó lớn hơn số trước đó.

Viết hàm **sumEvenNumbers(numberList)** để tính tổng các số chẵn có giá trị lớn hơn số liền kế trước đó.

```
sumEvenNumbers([-10, -4, 2, 8, 5]); // 6 vì lấy -4 + 2 + 8 = 6
sumEvenNumbers([2, 8, 5, 4]); // 10 vì lấy 2 + 8 = 10
```

ARRAY-SUM-02: Tổng của các chữ số trong mảng

Viết hàm **sumAllDigits(numberList)** để tính tổng các chữ số của từng số trong mảng **numberList**

```
sumAllDigits([]); // 0
sumAllDigits([4]); // 4
sumAllDigits([123, 4]); // 10 vì lấy 1 + 2 + 3 + 4 = 10
sumAllDigits([1234, 55]); // 20 vì lấy 1 + 2 + 3 + 4 + 5 + 5 = 20
```

ARRAY-SUM-03: Tổng các chữ số nhỏ nhất của số trong mảng

Viết hàm **sumAllMinDigits(numberList)** để tính tổng các chữ số nhỏ nhất của từng số trong mảng **numberList**

```
sumAllMinDigits([]); // 0
sumAllMinDigits([123]); // 1 vì chỉ lấy chữ số nhỏ nhất của 123 là 1
sumAllMinDigits([123, 532]); // 3 (lấy 1 từ 123 và 2 từ 532, tổng bằng 3)
```

ARRAY-SUM-04: Tổng tiền của giỏ hàng

Viết hàm **calcCartTotal(cartItemList)** để tính tổng tiền của giỏ hàng.

```
const cartItemList = [
  { id: 1, product: { id: 1, price: 50000 }, quantity: 1 },
  { id: 2, product: { id: 2, price: 100000 }, quantity: 2 },
];
calcCartTotal(cartItemList); // 250000 = 50 x 1 + 100000 x 2
```

9. Xử lý mảng (array): thêm / xoá phần tử

ARRAY-CD-01: Thêm một phần tử tại vị trí k

Viết hàm **insert(arr, newItem, k)** để thêm một phần tử mới **newItem** vào vị trí **k** (zero based) của mảng **arr**

- Nếu k lớn hơn độ dài của mảng thì thêm vào cuối mảng
- Nếu k bé hơn hoặc bằng 0 thì thêm vào đầu mảng

```
insert([1, 2, 3], 0, -1); // [0, 1, 2, 3]
insert([1, 2, 3], 4, 10); // [1, 2, 3, 4]
insert([1, 2, 3], 10, 2); // [1, 2, 10, 3]
```

ARRAY-CD-02: Xoá một phần tử tại vị trí k

Viết hàm **remove(arr, k, n)** để xoá **n** phần tử ra khỏi mảng **arr** kể từ vị trị **k** (zero based)

- Nếu k < 0 hoặc k >= arr.length thì trả về mảng hiện tại.
- Nếu k hợp lệ thì remove n phần tử kể từ vị trí k và trả về mảng kết quả.
- Nếu n không truyền vào thì remove tới cuối mảng.
- Lưu ý không được làm thay đổi mảng arr truyền vào.

```
remove([1, 2, 3], -1, 10); // [1, 2, 3]
remove([1, 2, 3], 3, 10); // [1, 2, 3]
remove([1, 2, 3], 1, 2); // [1]
remove([1, 2, 3], 0); // []
```

ARRAY-CD-03: Xoá một student có id cho trước

Viết hàm **removeStudentById(studentList, studentId)** để remove student có id là **studentId** ra khỏi mảng **studentList** và trả về mảng mới.

```
const studentList = [
  { id: 1, name: 'Alice' },
  { id: 2, name: 'Bob' },
]

removeStudentById(studentList, 1);
// should return:
// [
//   { id: 2, name: 'Bob' },
// ]


removeStudentById(studentList, 3);
// should return:
// [
//   { id: 1, name: 'Alice' },
//   { id: 2, name: 'Bob' },
// ]
```

ARRAY-CD-04: Cho một mảng đã được sắp xếp tăng dần. Cho một số, nhờ bạn add vào mảng nhưng phải đảm bảo mảng vẫn đc sắp xếp tăng dần.

Viết hàm **insert(numberList, newNumber)** để chèn **newNumber** vào mảng **numberList** sao cho mảng kết quả vẫn là tăng dần.

```
insert([], 3); // [3]
insert([1, 2, 4], 3); // [1, 2, 3, 4]
insert([1, 2, 3], 3); // [1, 2, 3, 3]
```

ARRAY-CD-05: Xoá các số trùng nhau trong mảng

Viết hàm **removeDuplicatedNumbers(numberList)** để xoá tất cả các số xuất hiện hơn 1 lần.

```
removeDuplicatedNumbers([1, 1, 2, 3, 2]); // [3]
removeDuplicatedNumbers([1, 2, 3]); // [1, 2, 3]
removeDuplicatedNumbers([1, 1]); // [1]
```

ARRAY-CD-06: Xoá các số trùng nhau nhưng giữ lại một số

Viết hàm **uniqueArray(numberList)** để xoá các số xuầ hiện hơn 1 lần, nhưng giữ lại 1 số.

```
uniqueArray([1, 1, 2, 2, 3]); // [1, 2, 3]
uniqueArray([1, 1, 1, 1]); // [1]
```

10. Xử lý mảng (array): mảng con

ARRAY-SUBARR-01: Tìm các mảng con tăng dần

Viết hàm **findAllIncreasingSubArr(numberList)** để tìm ra tất cả các mảng con tăng dần có trong mảng numberList.

> Mảng có tính chất tăng dần phải có ít nhất 2 phần tử và phần tử tại vị trí bất kỳ luôn lớn phần từ liền trước nó (trừ phần tử đầu tiên)

```
findAllIncreasingSubArr([1, 2, 3, -5, -10, 5, 10]);
// should return
// [
//   [1, 2, 3],
//   [-10, 5, 10]
// ]
```

ARRAY-SUBARR-02: Tìm các mảng con giảm dần có ít nhất 3 phần tử

Viết hàm **findAllDecreasingSubArr(numberList)** để tìm ra tất cả các mảng con giảm dần có độ dài ít nhất 3 phần tử.

> Mảng có tính chất giảm dần phải có ít nhất 2 phần tử và phần tử tại vị trí bất kỳ luôn nhỏ phần từ liền trước nó (trừ phần tử đầu tiên)

```
findAllDecreasingSubArr([3, 2, 1, 15, 10, 20]);
// should return
// [
//   [3, 2, 1],
// ]
//
// còn mảng [15, 10] là mảng giảm dần nhưng độ dài chưa đủ 3
```

ARRAY-SUBARR-03: Kiểm tra mảng a có phải là mảng con của mảng b

Viết hàm **isSubArray(a, b)** để kiểm tra xem **a** có phải là mảng con của **b** không?

- Nếu **a** là mảng rỗng thì luôn trả về **true**.
- Nếu **a** có độ dài lớn hơn b thì luôn trả về **false**.
- Trả về **true** nếu toàn bộ mảng a nằm trong mảng b theo đúng thứ tự của từng phần tử trong mảng a.

```
isSubArray([], [1]); // true
isSubArray([1], [1, 2]); // true
isSubArray([1, 2], [2, 3, 4]); // false
isSubArray([1, 2], [4, 10, 1, 2, 3]); // true
```

ARRAY-SUBARR-04: Tìm các mảng con có chứa số dương chẵn liên tiếp

Viết hàm **findAllPositiveEvenSubArr(numberList)** để tìm ra tất cả các mảng con chỉ chứa các số dương chẵn.

```
findAllPositiveEvenSubArr([1, 2, 4, 3, 5, 10, 20]);
// should return
// [
//   [2, 4],
//   [10, 20]
// ]
```

ARRAY-SUBARR-05: Tìm mảng con tăng dần có tổng lớn nhất, trả về con số tổng

Viết hàm **findMaxSumArray(numberList)** để tìm ra mảng con tăng dần có tổng lớn nhất và trả về **tổng của mảng con** đó.

```
findMaxSumArray([]); // 0
findMaxSumArray([1, 2, 3]); // 6
findMaxSumArray([1, 2, 3, 0, 10, 20]);
// should return 30 vì lấy tổng của mảng con [0, 10, 20]
```

ARRAY-SUBARR-06: Tìm mảng con tăng dần có tổng lớn nhất, trả về mảng con đầu tiên

Viết hàm **findMaxSumArray(numberList)** để tìm ra mảng con tăng dần có tổng lớn nhất và trả về **mảng con** đó.

```
findMaxSumArray([]); // []
findMaxSumArray([1, 2, 3, 0, 2, 4]);
// [1, 2, 3] vì
// có 2 mảng con hợp lệ: [1, 2, 3] và [0, 2, 4]
// cả 2 đều có tổng bằng 6, nên return mảng đầu tiên

findMaxSumArray([1, 2, 3]); // [1, 2, 3]
```

---

11. Bài tập tổng hợp luyện tư duy

LOGIC-01: Tìm 2 số có tổng bằng số cho trước

Viết hàm **findSumPair(numberList, targetSum)** để tìm ra 2 số trong mảng **numberList** có tổng bằng **targetSum**

Lưu ý:

- Trường hợp không tìm thấy 2 số thoả yêu cầu thì trả về mảng rỗng
- Trường hợp tìm thấy 2 số thoả yêu cầu thì trả về mảng chứa 2 số đó và sắp xếp tăng dần.

```
findSumPair({}); // []
findSumPair([], 10); // []
findSumPair([1, 2], 2); // [] vì không có 2 số nào có tổng bằng 2
findSumPair([3, 2, 1], 5); // [2, 3] vì 2 + 3 = 5 và sắp xếp tăng dần nên có mảng [2, 3]
```

LOGIC-02: Tìm số bị thiếu cho trong một mảng từ [5..n]

Viết hàm **findMissingNumber(numberList, n)** để tìm ra con số bị thiếu trong dãy số [5..n]

- Mảng numberList sẽ đảm bảo chỉ chứa các con số trong phạm vi từ 5 đến n.
- Mỗi số chỉ xuất hiện một lần trong mảng.
- Chắc chắn sẽ luôn có một con số bị thiếu trong mảng (ko hơn, ko kém)

```
findMissingNumber([5, 6, 8, 9, 10], 10); // 7 vì trong mảng thiếu mất số 7
findMissingNumber([5], 6); // 6 vì trong mảng thiếu mất số 6
```

LOGIC-03: Thống kê số lần xuất hiện của các số trong mảng

Viết hàm **statisticsNumbers(numberList)** để thống kê số lần xuất hiện của các số trong mảng **numberList**.

```
statisticsNumbers([]); // {}

statisticsNumbers([1, 1, 1, 2, 2, 3]);
// should return
// {
//   1: 3,
//   2: 2,
//   3: 1
// }
```

LOGIC-04: Tìm số có số lần xuất hiện nhiều nhất

Viết hàm **findMostFrequentNumber(numberList)** tìm số có số lần xuất hiện nhiều nhất trong mảng numberList, trường hợp có nhiều số cùng tần số xuất hiện thì trả về số đầu tiên.

```
findMostFrequentNumber([1]); // 1
findMostFrequentNumber([1, 2, 3, 2]); // 2
findMostFrequentNumber([1, 2, 3, 2, 3, 4]); // 2
```

Khoá học Javascript cho người mới bắt đầu 2021 🎉

- Tác giả: **Hậu Nguyễn** - Founder Easy Frontend
- Khoá học chỉ được published trên Udemy, không thông qua trung gian.
- Khoá học không bán dạng videos upload trên Google Drive hay bất cứ hình thức nào tương tự.
- Khoá học có nhóm discord để hỗ trợ trong quá trình học tập.

☎️ Liên hệ tác giả để được hỗ trợ:

- ✅ Facebook: https://www.facebook.com/nvhauesmn/
- ✅ Fanpage: https://www.facebook.com/learn.easyfrontend
- ✅ Youtube Channel: https://www.youtube.com/easyfrontend