1. Merge array and remove duplicated numbers

Viết hàm mergeArray(a, b) nhận vào 2 mảng số nguyên dương và trả về mảng gộp của cả 2 mảng số và loại bỏ đi các số bị trùng nhau.

Lưu ý:

a, b có thể là bất cứ kiểu dữ liệu gì

Khi a, b là mảng thì a, b sẽ là mảng các số nguyên dương

Mỗi mảng a, b sẽ chứa những con số không trùng nhau

Ví dụ:

```
mergeArray(1, true) --> []

mergeArray([], []) --> []

mergeArray([], [1, 2, 3]) --> [1, 2, 3]

mergeArray([1, 2, 3], [2, 3, 4]) --> [1, 2, 3, 4] vì 2, 3 trùng nhau nên khi gộp
mảng chỉ giữ lại một số
```

2. Tìm số xuất hiện nhiều nhất trong mảng

Viết hàm mostFrequent(numberList) nhận vào là một mảng số và trả về số có số lần xuất hiện nhiều nhất.

Lưu ý:

Tham số truyền vào có thể không phải là mảng, lúc đó trả về undefined

Trường hợp mảng rỗng, trả về undefined

Trường hợp có nhiều số có cùng số lần xuất hiện, trả về số đầu tiên

Ví dụ:

```
mostFrequent(true) --> undefined

mostFrequent() --> undefined

mostFrequent([]) --> undefined

mostFrequent([1, 2, 3]) --> 1

mostFrequent([2, 1, 3]) --> 2
```

```
mostFrequent([2, 2, 3, 3, 2, 3, 3]) --> 3 vì 3 xuất hiện 4 lần, còn 2 xuất hiện 3 lần
```

3. Đếm số lượng học sinh có giới tính là nam

Viết hàm countStudents(studentList) nhận vào là một mảng student và trả về số lượng student có gender là male.

Lưu ý

Nếu truyền vào không phải là mảng hoặc mảng rỗng thì trả về 0

Mỗi object student sẽ có 2 keys: id và gender

Thuộc tính gender sẽ có 1 trong 2 giá trị: 'male' hoặc 'female'

Ví dụ:

```
// should return 0 because there is no male student
countStudents([
    { id: 1, gender: 'female' },
    { id: 2, gender: 'female' },
]);
// should return 1 because there is one student with gender='male'
countStudents([
    { id: 1, gender: 'male' },
    { id: 2, gender: 'female' },
]);
```

4. Tính tổng tiền giỏ hàng

Viết hàm calcCartTotal(cartItemList) nhận vào danh sách các item trong giỏ hàng và trả về tổng tiền của giỏ hàng đó.

Lưu ý

cartItemList có thể không phải là mảng

Trường hợp cartItemList là mảng, thì sẽ là mảng của các object cartItem

Mỗi cartItem sẽ có 3 keys: id, product và quantity

Ví dụ

```
expect(calcCartTotal([])).toBe(0); // 0
expect(calcCartTotal([])).toBe(); // 0
```

```
// should return 500000 = 100000 * 4 + 50000 * 2
calcCartTotal([
    { id: 1, product: { id: 1, price: 100000 }, quantity: 4 },
    { id: 2, product: { id: 2, price: 50000 }, quantity: 2 },
])
```

Hãy thử dùng hàm reduce để cài đặt hàm này nhé!

5. Tìm các sản phẩm có giá tiền lớn hơn 100.000đ

Viết hàm filterProducts(productList) nhận vào danh sách sản phẩm và trả về các sản phẩm có giá lớn hơn 100000.

Ví dụ:

```
filterProducts({}); // []
filterProducts([]); // []
// should return [] because there is no product having price > 100000
filterProducts([
    { id: 1, price: 10000 },
    { id: 2, price: 50000 },
    { id: 3, price: 70000 },
])
filterProducts([
    { id: 1, price: 100000 },
    { id: 2, price: 150000 },
    { id: 3, price: 270000 },
]);

// should return a list of products having price > 100000
// [
//     { id: 2, price: 150000 },
//     { id: 3, price: 270000 },
// ]
```

6. Chia nhỏ mảng thành nhiều mảng con

Viết hàm chunkArray(array, size) để chia nhỏ array thành nhiều mảng con có length là size.

Ví dụ array có 5 phần tử và size = 2 --> tức là chia nhỏ ra thành nhiều mảng con, mỗi mảng có 2 phần tử

Từ đó, suy ra kết quả sẽ được 3 chunks, chunk 1 và chunk 2 có 2 phần tử, và chunk số 3 có 1 phần tử.

Lưu ý:

Với array là mảng bất kỳ, size là kích thước của mảng con cần chia nhỏ.

Trường hợp array không phải là mảng hoặc size không hợp lệ (nhỏ hơn hoặc bằng 0) thì trả về []

Trường hợp array không chia đều được cho các mảng con, thì mảng có phần tử ít nhất nằm ở cuối mảng.

Trường hợp số lượng chunks tạo ra nhiều hơn 20 thì throw new Error('Too many chunks');

Ví dụ:

```
chunkArray({}); // []
chunkArray([]); // []
chunkArray([1, 2, 3], -1); // []
chunkArray([1, 2, 3], 0); // []
chunkArray([1, 2, 3], 2); // [[1, 2], [3]]
chunkArray([1, 2, 3], 1); // [[1], [2], [3]]
chunkArray([1, 2, 3], 3); // [[1, 2, 3]]
```

7. Tìm 2 số trong mảng có tổng bằng một số cho trước

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

Lưu ý:

numberList có thể không phải là mảng

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.

Ví dụ:

```
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]
findSumPair([3, 3, 1, 2], 6); // [3, 3]
```

8. Tìm số bị trùng đầu tiên trong mảng

Viết hàm findFirstDuplicate(numberList) nhận vào mảng số nguyên dương .

Trả về phần tử đầu tiên bị lặp lại trong mảng, nếu không có phần tử nào bị lặp lại trả về -1.

Ví dụ:

```
findFirstDuplicate([]) -> -1

findFirstDuplicate({}) -> -1

findFirstDuplicate([1,2,3]) -> -1

findFirstDuplicate([1,1,3]) -> 1

findFirstDuplicate([1,2,2,1]) -> 2
```

Yêu cầu: chỉ dùng duy nhất 1 vòng for

9. Kiểm tra mảng có phải là dạng "mountain" array không?

Viết hàm validMountainArray(numberList) nhận vào mảng số nguyên. Trả về true nếu mảng là MountainArray, ngược lại trả về false.

Mảng được gọi là MountainArray khi và chỉ khi:

- Mảng đó có ít nhất 3 phần tử
- Tồn tại số nguyên i, 0 < i < numberList.length - 1 sao cho

numberList[0] < numberList[1] < ... < numberList[i - 1] < numberList[i]

và numberList[i] > numberList[i + 1] > ... > numberList[numberList.length - 1].

--> tức nôm na i là đỉnh và giảm dần về đầu và cuối mảng.

Ví dụ:

```
validMountainArray([1]) -> false
validMountainArray(1) -> false
validMountainArray([3, 5, 5]) -> false
validMountainArray([0,1,2]) -> false
validMountainArray([0,3,2,1]) -> true
```