

# LẬP TRÌNH JAVASCRIPT CƠ BẢN

*MẢNG ĐỐI TƯỢNG VÀ CÁC KỸ THUẬT THAO TÁC, SẮP XẾP*



## **Nội dung:**

1. Trừu tượng hóa dữ liệu, mảng các đối tượng
2. Các kỹ thuật thao tác với mảng các đối tượng



# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Trừu tượng hóa dữ liệu:


➤ Là **phương pháp tích hợp các kiểu dữ liệu đơn thành kiểu dữ liệu phức** nhằm mô tả biểu diễn thông tin của một khái niệm hay một đối tượng trong thế giới thực.

➤ Trong JavaScript, để thực hiện việc trừu tượng hóa dữ liệu, ta sử dụng **object**.

```
const object = {  
  hello: 'world'  
}
```

PROPERTY VALUE

PROPERTY NAME (KEY)





# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Trừu tượng hóa dữ liệu:

### ➤ Ví dụ:

- Đối tượng phân số (**phanso** kiểu **phanso**) gồm: tử số (**tuso** kiểu **number**), mẫu số (**mauso** kiểu **number**)
- Đối tượng điểm (**diem** kiểu **diem**) trong mặt phẳng Oxy gồm: hoành độ x (**x** kiểu **number**), tung độ y (**y** kiểu **number**)

```
const object = {  
  hello: 'world'  
}
```

PROPERTY VALUE

PROPERTY NAME (KEY)





# Trừu tượng hóa dữ liệu, mảng các đối tượng

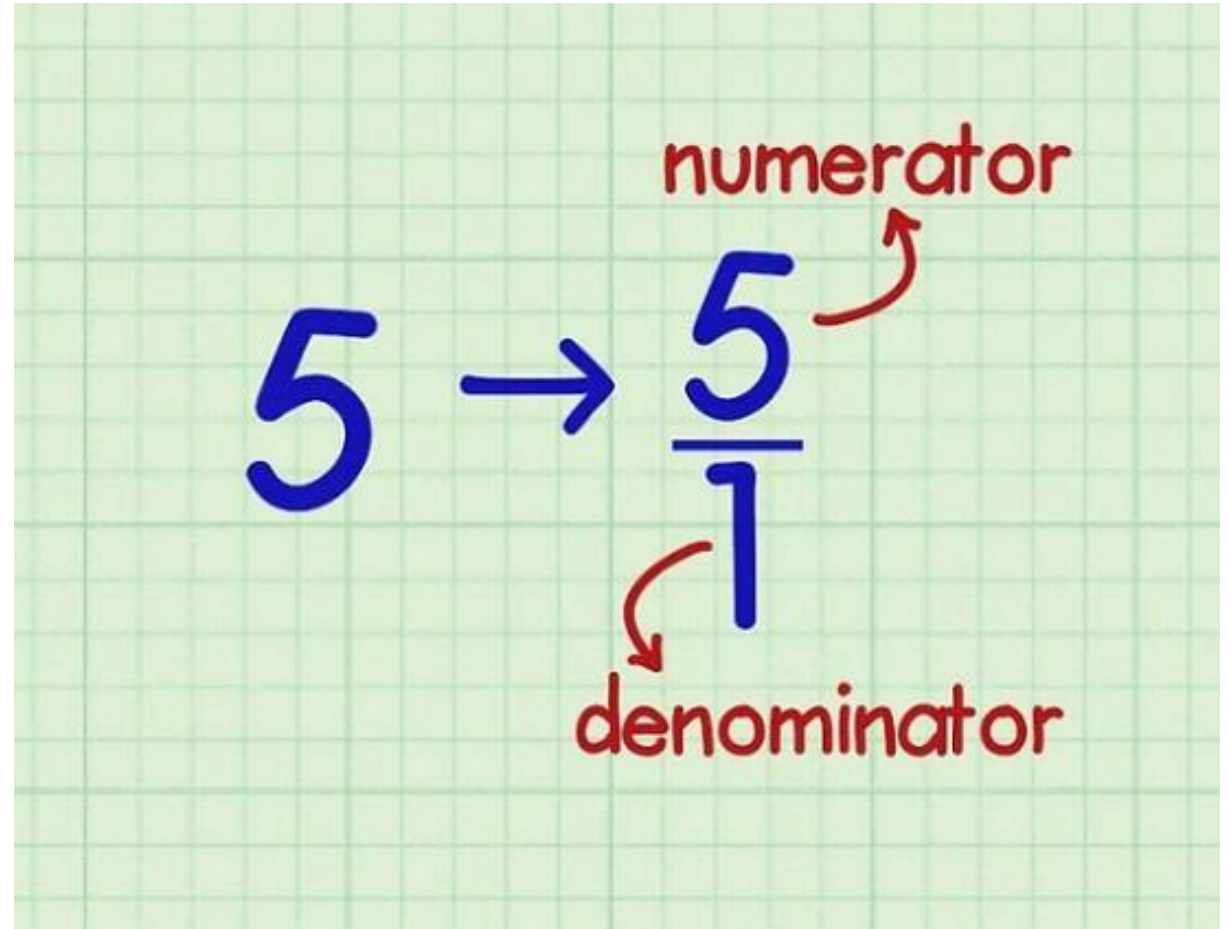
## ❑ Trừu tượng hóa dữ liệu:

➤ Ví dụ 1: Khai báo một phân số

```
var fraction = {  
    numerator: 0,  
    denominator: 0  
}
```

▪ Hoặc:

```
var fraction = new Object();  
fraction.numerator = 0;  
fraction.denominator = 0;
```





# Trừu tượng hóa dữ liệu, mảng các đối tượng

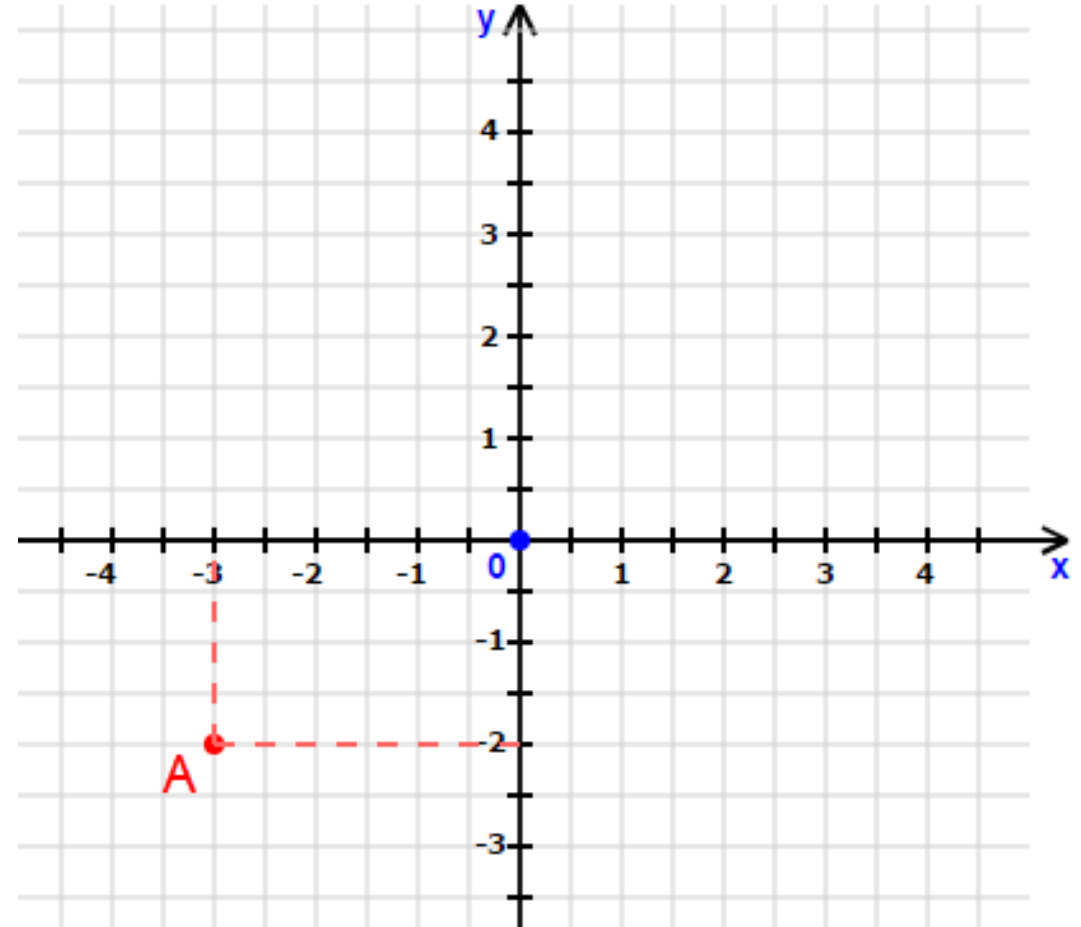
## ❑ Trừu tượng hóa dữ liệu:

➤ Ví dụ 2: Khai báo 1 điểm (Oxy)

```
var point = {  
    x: 0,  
    y: 0  
}
```

▪ Hoặc:

```
var point = new Object();  
point.x = 0;  
point.y = 0;
```



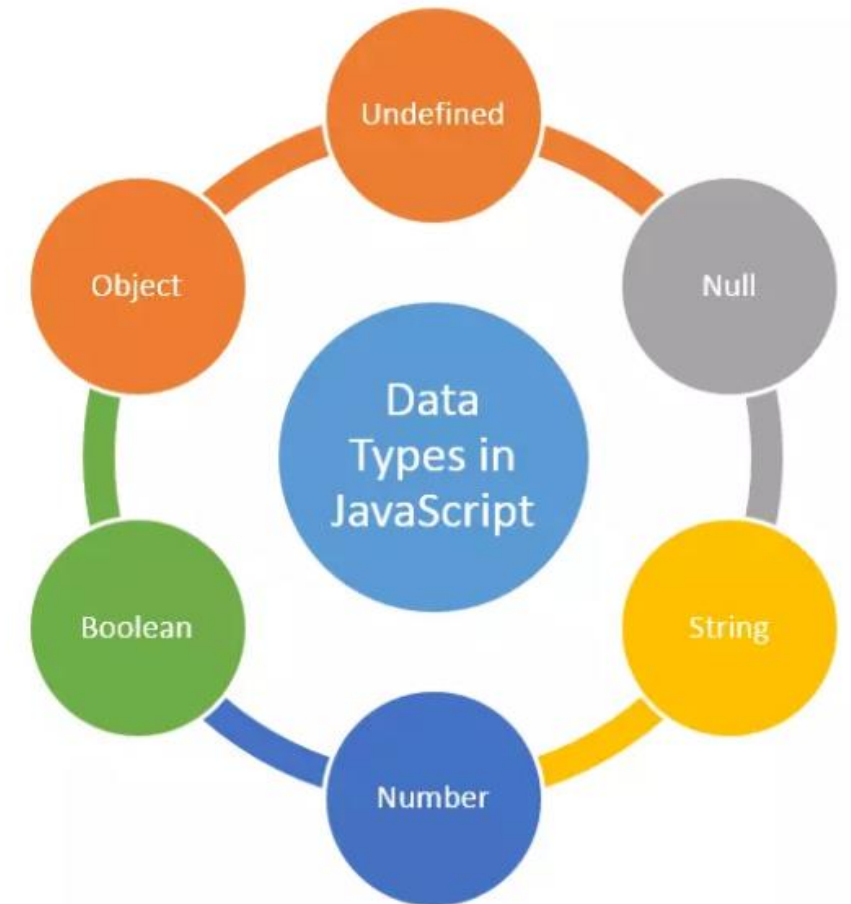
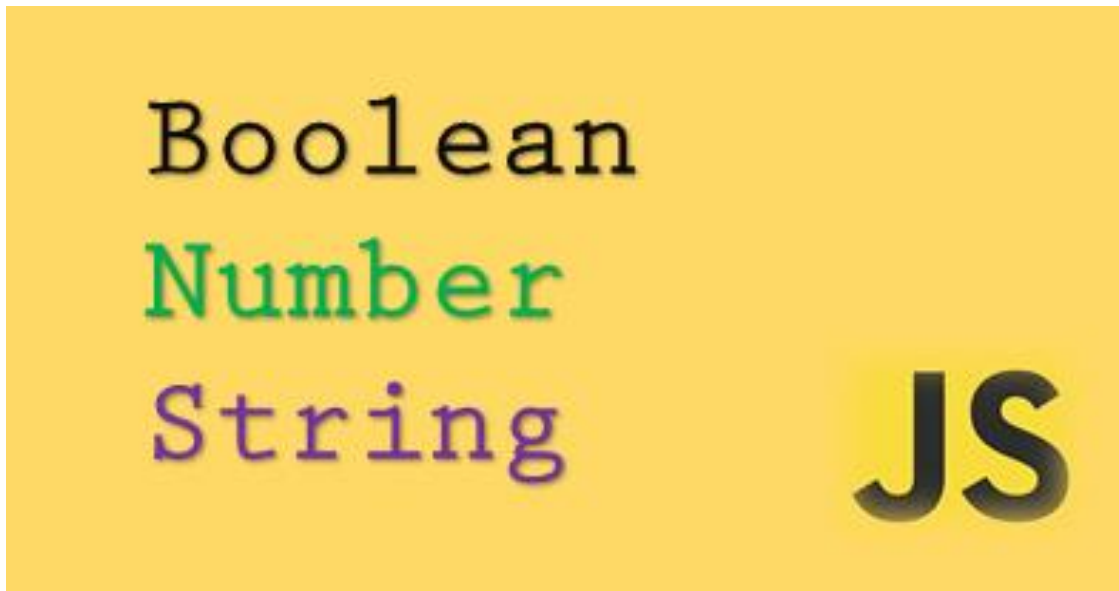


# Trừu tượng hóa dữ liệu, mảng các đối tượng

❑ Để giải quyết một bài toán ta có thể sử dụng phương pháp **trừu tượng hóa dữ liệu**. Có 3 bước chính như sau:

➤ **Bước 1:** Xác định các kiểu dữ liệu.

- Kiểu dữ liệu nào đã có sẵn
- Kiểu dữ liệu nào phải định nghĩa mới



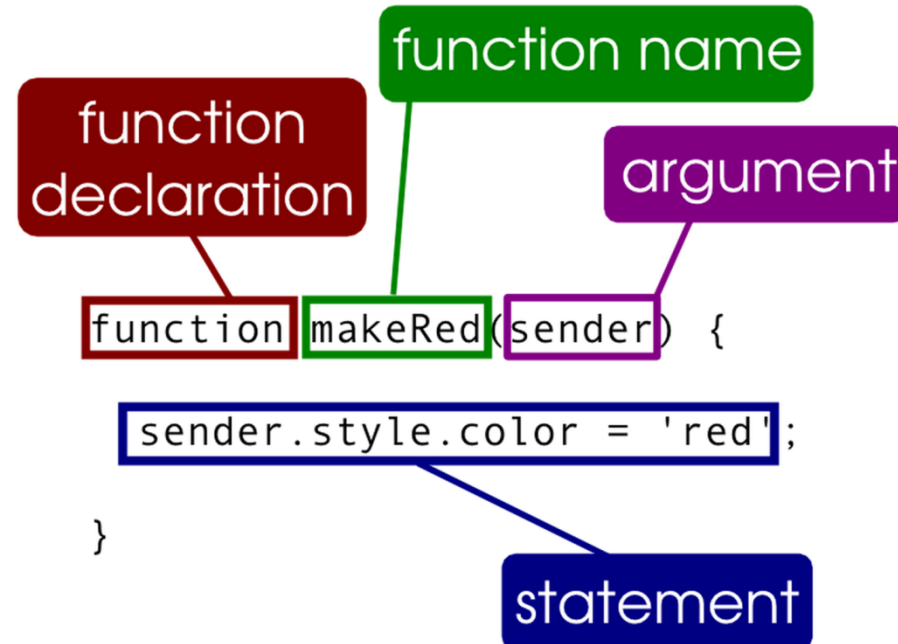


# Trừu tượng hóa dữ liệu, mảng các đối tượng

❑ Để giải quyết một bài toán ta có thể sử dụng phương pháp **trừu tượng hóa dữ liệu**. Có 3 bước chính như sau:

➤ **Bước 2:** Thiết kế hàm.

▪ Bài toán giải quyết cần phải có bao nhiêu hàm, tên hàm, các tham số đầu vào, kiểu dữ liệu trả về như thế nào.







# Trừu tượng hóa dữ liệu, mảng các đối tượng

❑ Để giải quyết một bài toán ta có thể sử dụng phương pháp **trừu tượng hóa dữ liệu**. Có 3 bước chính như sau:

➤ **Bước 3**: Định nghĩa hàm.

▪ Tiến hành định nghĩa các hàm đã thiết kế và khai báo trong bước 2.

```
function greet(name) {  
    // code  
}  
  
greet(name);  
// code
```

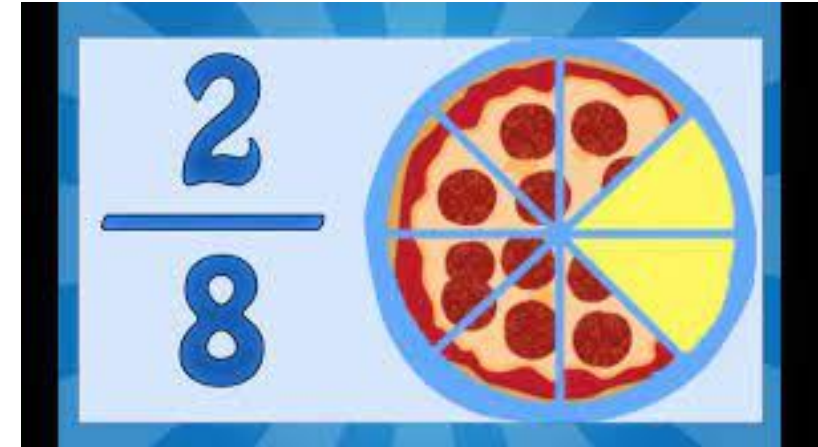
function call



# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Thực hành chủ đề **phân số**:

- **Khai báo** kiểu dữ liệu.
- **Thiết kế** các hàm dưới đây
- **Định nghĩa** các hàm dưới đây:
  - Nhập phân số
  - Xuất phân số
  - Tìm UCLN của phân số
  - Rút gọn phân số
  - Tính tổng 2 phân số
  - Tính tích 2 phân số



- Kiểm tra phân số tối giản
- Kiểm tra phân số dương
- Quy đồng 2 phân số
- So sánh 2 phân số



# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Thực hành chủ đề **phân số**:

### ➤ **Khai báo** kiểu dữ liệu:

```
var fraction = {  
    numerator: 0,  
    denominator: 0  
}
```

### ▪ Hoặc:

```
var fraction = new Object();  
fraction.numerator = 0;  
fraction.denominator = 0;
```





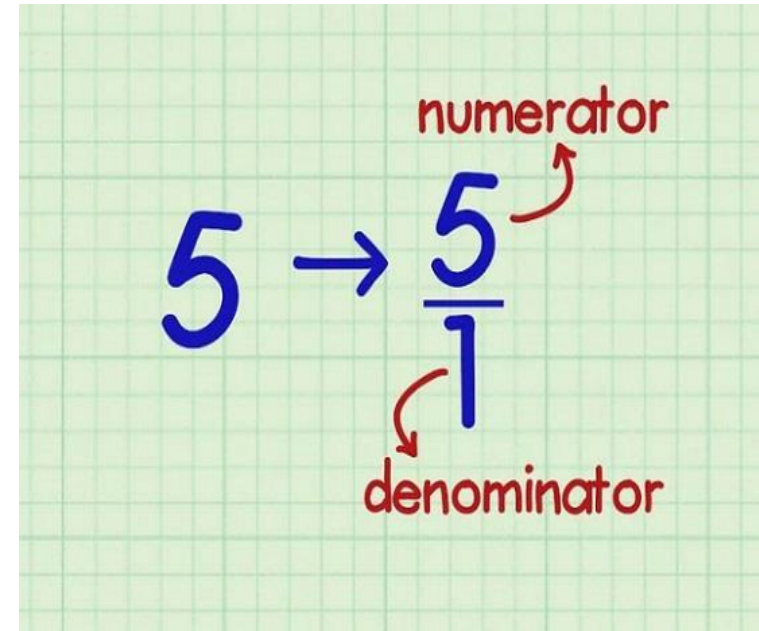
# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Thực hành chủ đề **phân số**:

➤ Hàm nhập/xuất phân số:

```
function inputFraction() {  
    fraction.numerator = prompt("Tử số: ");  
    fraction.denominator = prompt("Mẫu số: ");  
    return fraction;  
}
```

```
function outputFraction(fraction) {  
    console.log(fraction.numerator + "/" + fraction.denominator);  
}
```





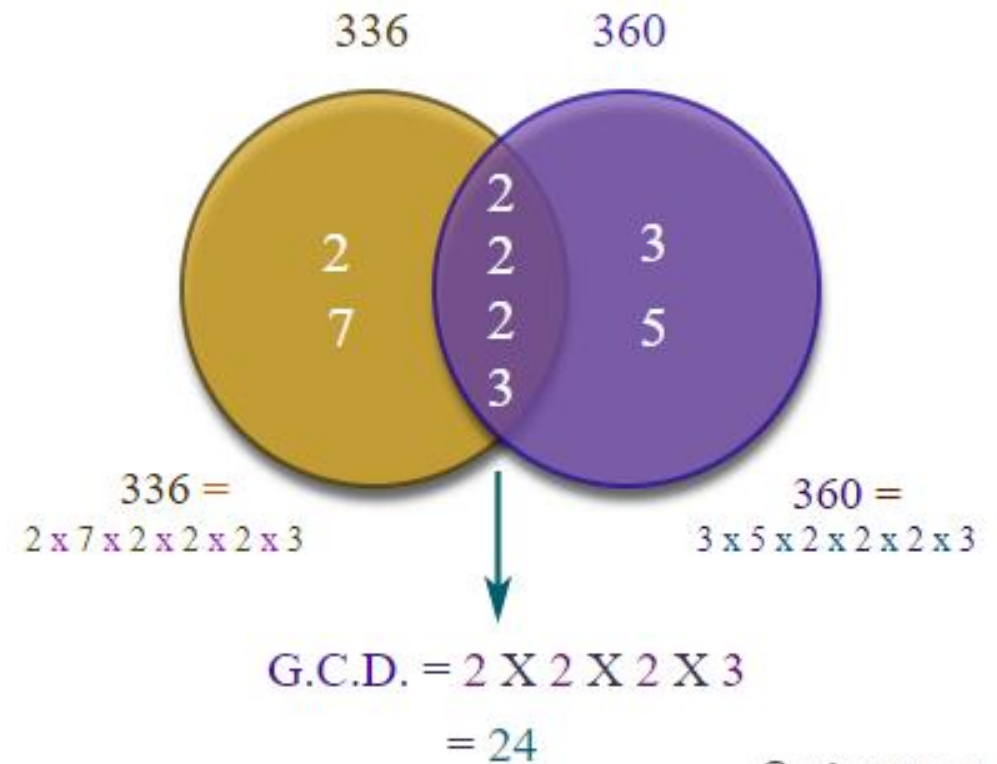
# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Thực hành chủ đề **phân số**:

➤ Hàm tìm ước số chung lớn nhất (UCLN):

```
function findGCD (a, b) {  
    while (a != b) {  
        if (a > b) a -= b;  
        else b -= a;  
    }  
    return a;  
}
```

// GCD: Greatest Common Divisor



© w3resource.com



# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Thực hành chủ đề **phân số**:

➤ Hàm rút gọn phân số:

```
function reduceFraction(fraction) {  
    let fractionGCD = findGCD(fraction.numerator, fraction.denominator);  
    fraction.numerator = fraction.numerator / fractionGCD;  
    fraction.denominator = fraction.denominator / fractionGCD;  
    return fraction;  
}
```



# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Thực hành chủ đề **phân số**:

➤ Hàm tính tổng 2 phân số:

```
function sumFractions(fractionX, fractionY) {  
    let fractionTemp = new Object();  
    fractionTemp.numerator =  
    (fractionX.numerator * fractionY.denominator) +  
    (fractionY.numerator * fractionX.denominator);  
    fractionTemp.denominator =  
    fractionX.denominator * fractionY.denominator;  
    return fractionTemp;  
}
```

$$\begin{matrix} x & & y & & temp \\ \frac{a}{b} & + & \frac{c}{d} & = & \frac{ad+cb}{bd} \end{matrix}$$



# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Thực hành chủ đề **phân số**:

➤ Hàm tính tích 2 phân số:

```
function multiplyFractions(fractionX, fractionY) {  
    let fractionTemp = new Object();  
    fractionTemp.numerator =  
fractionX.numerator * fractionY.numerator;  
    fractionTemp.denominator =  
fractionX.denominator * fractionY.denominator;  
    return fractionTemp;  
}
```

$$\begin{matrix} x & y & & temp \\ \frac{a}{b} & + & \frac{c}{d} & = & \frac{ac}{bd} \end{matrix}$$





# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Thực hành chủ đề **phân số**:

➤ Kiểm tra phân số tối giản:

```
function isSimplestFraction(fraction) {  
    let result = findGCD(  
fraction.numerator, fraction.denominator);  
    if (result == 1)  
        return 1;  
    return 0;  
}
```

$$\frac{3}{6} = \frac{1}{2}$$

The diagram illustrates the simplification of the fraction  $\frac{3}{6}$  to  $\frac{1}{2}$ . Two large yellow curved arrows point from the numerator and denominator of the left fraction to those of the right fraction. Above the top arrow is the text  $:3$ , and below the bottom arrow is the text  $:3$ , indicating that both the numerator and denominator are divided by 3.

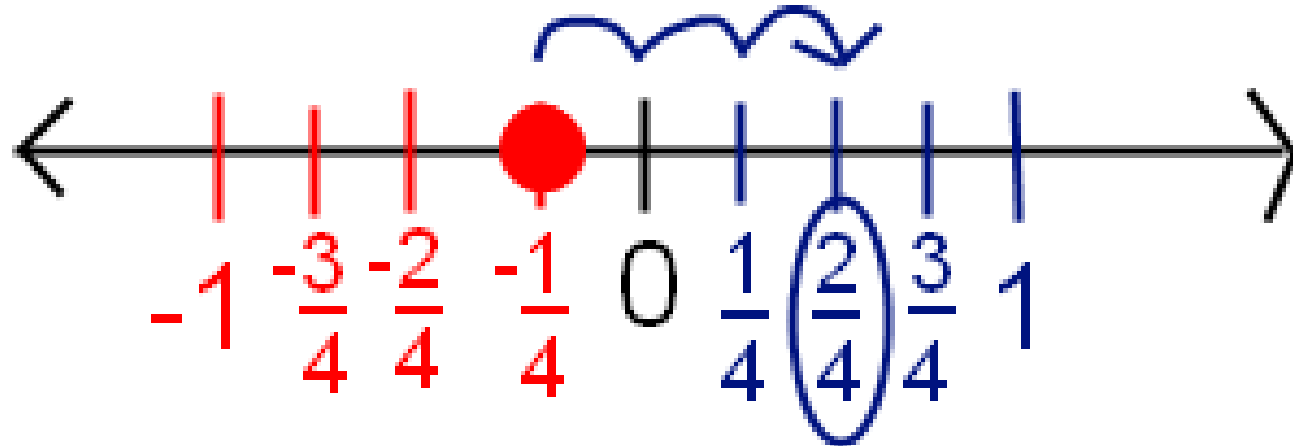


# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Thực hành chủ đề **phân số**:

➤ Kiểm tra phân số dương:

```
function isPositiveFraction(fraction) {  
    if (fraction.numerator * fraction.denominator > 0)  
        return 1;  
    return 0;  
}
```





# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Thực hành chủ đề **phân số**:

➤ Quy đồng mẫu số phân số:

```
function homogenizeDenominator (fractionX, fractionY) {  
    let bc = fractionX.denominator * fractionY.denominator;  
    fractionX.numerator = fractionX.numerator * fractionY.denominator;  
    fractionY.numerator = fractionY.numerator * fractionX.denominator;  
    fractionX.denominator = bc;  
    fractionY.denominator = bc;  
}
```



# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Thực hành chủ đề **phân số**:

➤ So sánh 2 phân số:

```
function compareFractions (fractionX, fractionY) {  
    let a = fractionX.numerator / fractionX.denominator;  
    let b = fractionY.numerator / fractionY.denominator;  
    if (a > b) { return 1; }  
    if (a < b) { return -1; }  
    return 0;  
}
```

➤ Các **giá trị trả về** mang **ý nghĩa** như sau:

- **1**: phân số X > phân số Y
- **-1**: phân số X < phân số Y
- **0**: phân số X = phân số Y



# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Mảng các đối tượng:

### ➤ Đối tượng car (xe)

```
var car = {  
    color: "purple",  
    capacity: 7  
}
```

### ▪ Hoặc:

```
var car = new Object();  
car.color = "purple";  
car.capacity = 7;
```



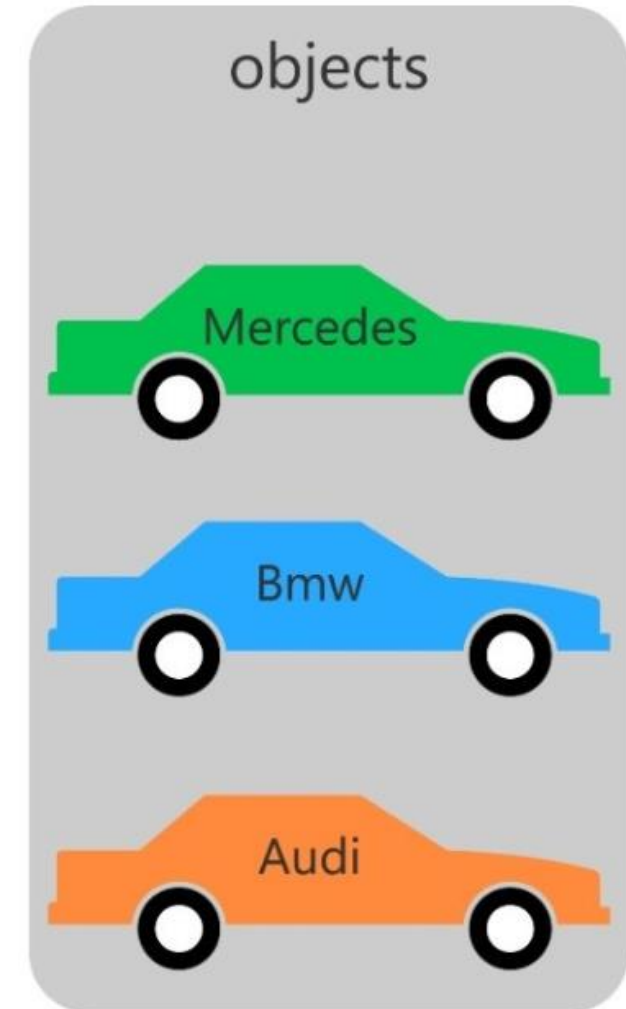


# Trừu tượng hóa dữ liệu, mảng các đối tượng

## ❑ Mảng các đối tượng:

➤ Nhập/xuất đối tượng car (xe):

```
function inputCar() {  
    var newCar = new Object();  
    newCar.color = prompt("Color: ");  
    newCar.capacity = prompt("Capacity: ");  
    return newCar;  
}  
  
function outputCar(newCar) {  
    console.log(newCar);  
}
```





# Các kỹ thuật thao tác với mảng các đối tượng

## ❑ Kỹ thuật nhập/xuất mảng:

➤ Nhập/xuất mảng các đối tượng car (xe):

```
var arr = [];  
var n;
```

```
function inputCar() { ... }  
function outputCar() { ... }
```

```
function inputCarArray() {  
    n = prompt("Enter n: ");  
    for (var i=0; i<n; i++) {  
        arr[i] = inputCar();  
    }  
}
```

```
function outputCarArray() {  
    for (var i=0; i<n; i++) {  
        outputCar();  
    }  
}
```



# Các kỹ thuật thao tác với mảng các đối tượng

## ❑ Kỹ thuật sắp xếp mảng dùng thuật toán:

- Sắp xếp mảng các đối tượng car (xe) theo sức chứa (capacity) của xe:

```
function interchangeSort () { // sắp xếp mảng tăng dần
```

```
  for (var i = 0; i <= n-2; i++) {
```

```
    for (var j = i + 1; j <= n - 1; j++) {
```

```
      if (Number(arr[i].capacity) < Number(arr[j].capacity)) {
```

```
        swap(arr[i], arr[j]);
```

```
      }
```

```
    }
```

```
  }
```

```
let temp = arr[i];  
arr[i]=arr[j];  
arr[j]=temp;
```





# Các kỹ thuật thao tác với mảng các đối tượng

## ❑ Kỹ thuật sắp xếp mảng tự viết thuật toán:

- Sắp xếp mảng các đối tượng car (xe) theo sức chứa (capacity) của xe:

// sắp xếp mảng giảm dần

```
var arr = [];
```

```
var n;
```

```
car1 = arr[i];
```

```
car2 = arr[i+1];
```

```
if (car1.capacity < car2.capacity) { return 1; }
```

```
if (car1.capacity > car2.capacity) { return -1; }
```

```
return 0;
```

- Các **giá trị trả về** mang **ý nghĩa** như sau:

- **1**: sức chứa car1 > sức chứa car2
- **-1**: sức chứa car1 < sức chứa car2
- **0**: sức chứa car1 = sức chứa car2



# Các kỹ thuật thao tác với mảng các đối tượng

## ❑ Kỹ thuật sắp xếp mảng sử dụng hàm tiện ích **array.sort**:

➤ Sắp xếp mảng các đối tượng car (xe) theo sức chứa (capacity) của xe:

```
let arr = []; // tự nhập nhé
```

```
var carArraySorted = arr.sort((car1, car2) =>
```

```
(car1.capacity) < car2.capacity) ? 1: (car1.capacity) > car2.capacity) ? -1 : 0);
```

### ▪ Lưu ý:

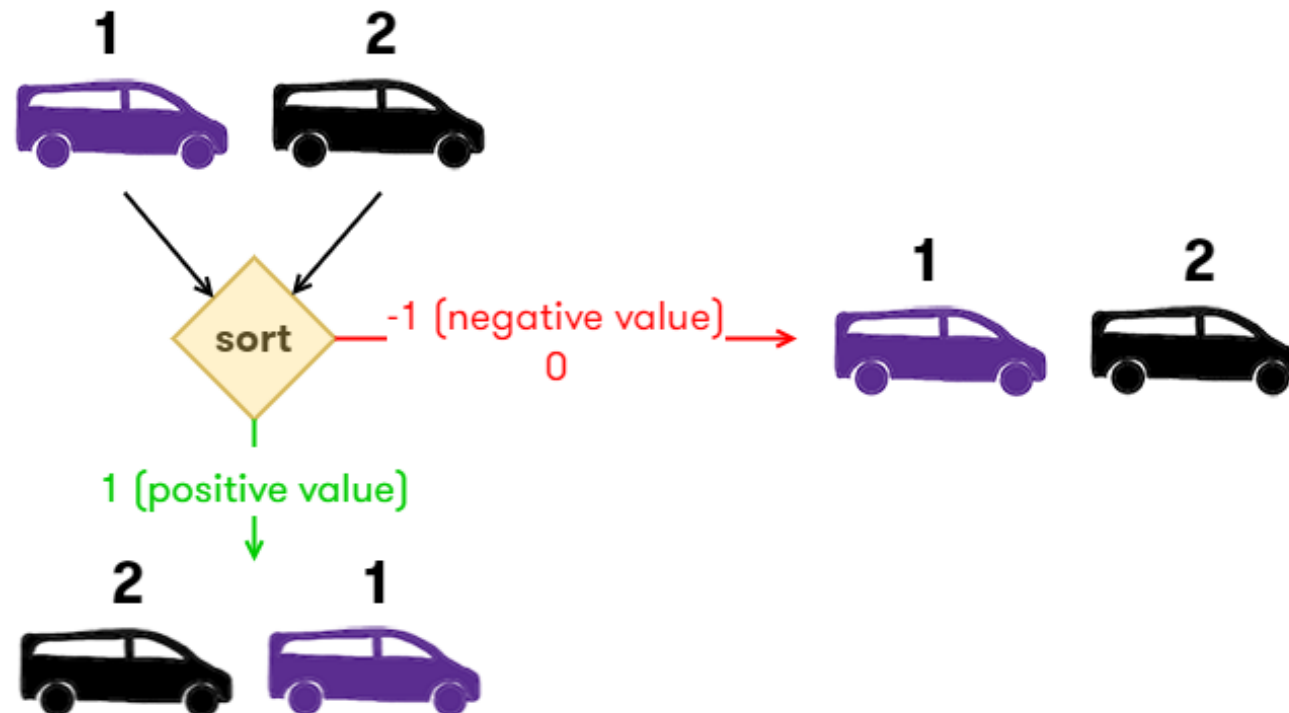
- **array.sort**: **array** nghĩa là **tên mảng muốn sử dụng** (arr, carArr, ...)
- **carArraySorted**: là mảng đã được sắp xếp
- sắp xếp **car2, car1** (nếu sức chứa car1 < sức chứa car2)
- sắp xếp **car1, car2** (nếu sức chứa car1 > sức chứa car2)



# Các kỹ thuật thao tác với mảng các đối tượng

❑ Kỹ thuật sắp xếp mảng sử dụng hàm tiện ích **array.sort**:

- Hình minh họa cách thức hoạt động hàm **array.sort**
- Thứ tự trong khai báo (**car1**, **car2**) và các phép so sánh của hàm sort quyết định sắp xếp **mảng tăng dần** hoặc **giảm dần**.



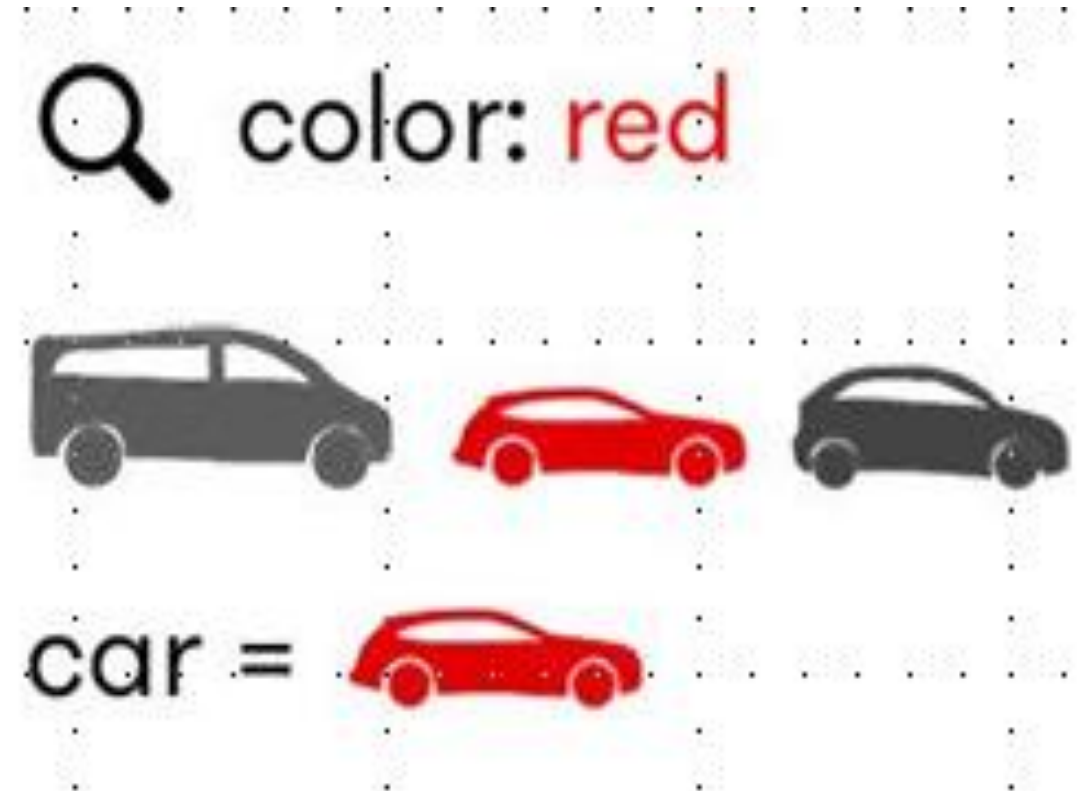


# Các kỹ thuật thao tác với mảng các đối tượng

## ❑ Kỹ thuật đếm số lượng phần tử trong mảng (tự viết):

➤ Đếm số lượng xe màu đỏ:

```
function countRedCar (arr) {  
    var count = 0;  
    for (let i=0; i<n; i++) {  
        if (arr[i].color == "red")) {  
            count++;  
        }  
    }  
    return count;  
}
```



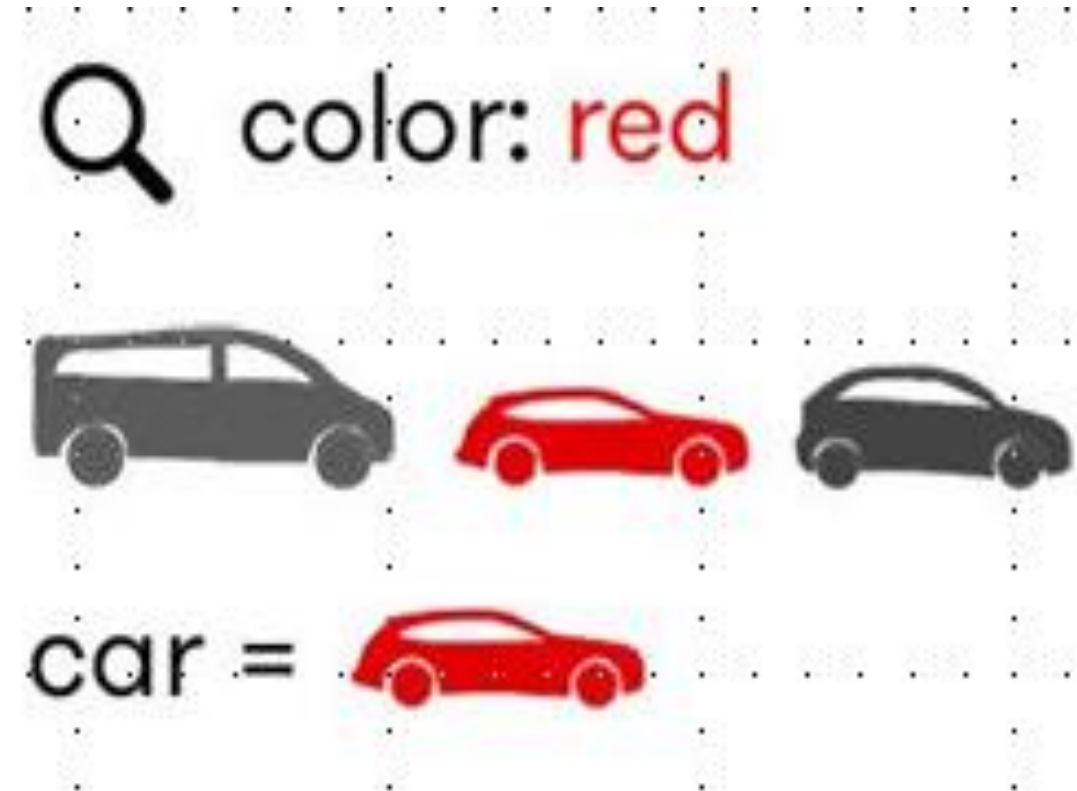


# Các kỹ thuật thao tác với mảng các đối tượng

## ❑ Kỹ thuật tìm kiếm phần tử trong mảng (tự viết):

➤ Tìm xe có sức chứa lớn nhất:

```
function findMaxCapacity (arr) {  
    var max = arr[0];  
    for (let i=0; i<n; i++) {  
        if (compareCar(arr[i], max)==1) {  
            max = arr[i];  
        }  
    }  
    return max;  
}
```





## Các kỹ thuật thao tác với mảng các đối tượng

- ❑ Kỹ thuật tìm kiếm phần tử trong mảng sử dụng hàm tiện ích:
  - **array.find**: tìm kiếm phần tử trong mảng các đối tượng.
  - **array.filter**: tìm kiếm mảng phần tử trong mảng các đối tượng.





# Các kỹ thuật thao tác với mảng các đối tượng

## ❑ Kỹ thuật tìm kiếm phần tử trong mảng sử dụng hàm tiện ích:

➤ Áp dụng **array.find** (trả về 1 phần tử có thuộc tính cần tìm):

```
let redCar = arr.find(car => car.color === "red");  
console.log(redCar);
```

🔍 color: red



car = 



# Các kỹ thuật thao tác với mảng các đối tượng

## ❑ Kỹ thuật tìm kiếm phần tử trong mảng sử dụng hàm tiện ích:

➤ Áp dụng **array.filter** (trả về 1 phần tử có thuộc tính cần tìm):

```
let redCars = arr.filter(car => car.color === "red");  
console.log(redCars);
```

🔍 color: red



car = 





## Tổng kết:

- ☐ Trừu tượng hóa dữ liệu, mảng các đối tượng
- ☐ Các kỹ thuật thao tác với mảng các đối tượng

Let's  
Recap

