# Week 03 — JS Advanced 1 — Lý thuyết

## 1) Object & Destructuring; Rest/Spread; Tư duy bất biến (immutability)

• Destructuring giúp lấy nhanh thuộc tính/phần tử; rest (...) gom phần còn lại; spread (...) sao chép nông (shallow copy).

• Immutability: hạn chế mutate dữ liệu gốc; tạo bản sao rồi thay đổi để dễ debug/undo.

const user = { id: 1, name: 'An', role: 'user', address: { city: 'HN' } };  
const { name, role, address: { city } } = user; // destructuring lồng  
const { id, ...rest } = user; // rest  
  
// spread: sao chép nông  
const updated = { ...user, role: 'admin' }; // user không đổi  
const arr = [1,2,3];  
const arr2 = [...arr, 4]; // [1,2,3,4]  
  
// Chú ý: spread là shallow — address bên trong vẫn cùng tham chiếu  
updated.address.city = 'HCM';  
console.log(user.address.city); // 'HCM' (cùng tham chiếu)

## 2) Mảng nâng cao: map / filter / reduce / find / some / every / sort / forEach

• map: chuyển đổi từng phần tử. filter: lọc theo điều kiện. reduce: gộp về 1 giá trị.

• find: trả về phần tử đầu tiên khớp. some/every: kiểm tra tồn tại/tất cả. forEach: duyệt có tác dụng phụ.

const users = [  
 {id:1, name:'A', age:17},  
 {id:2, name:'B', age:22},  
 {id:3, name:'C', age:18}  
];  
  
const adultNames = users.filter(u=>u.age>=18).map(u=>u.name); // ['B','C']  
const firstAdult = users.find(u=>u.age>=18); // {id:2,...}  
const hasTeen = users.some(u=>u.age<18); // true  
const allAdult = users.every(u=>u.age>=18); // false  
  
// reduce: tổng số tuổi  
const totalAge = users.reduce((sum,u)=>sum+u.age, 0);

## 3) Sắp xếp (sort) đúng cho số/chữ; tie-breaker

• sort không ổn nếu không truyền comparator (so sánh chuỗi). Comparator: trả <0, 0, >0.

• Tách tiêu chí: sắp theo price tăng, nếu bằng thì theo name dùng localeCompare.

const products = [  
 {name:'Bút', price: 20000},  
 {name:'Vở', price: 10000},  
 {name:'Tẩy', price: 10000},  
];  
  
// Không mutate gốc: copy trước rồi sort  
const sorted = [...products].sort((a,b)=> a.price - b.price || a.name.localeCompare(b.name));  
console.log(sorted);

## 4) Mẫu reduce hay dùng: groupBy, đếm tần suất, indexing

• groupBy: gom phần tử theo khóa. Histogram: đếm tần suất. Indexing: map id → object để truy cập O(1).

const orders = [  
 {id:1, cat:'A', amount:100},  
 {id:2, cat:'B', amount:200},  
 {id:3, cat:'A', amount:150},  
];  
  
// groupBy theo cat  
const byCat = orders.reduce((acc,o)=>{  
 (acc[o.cat] ||= []).push(o);  
 return acc;  
}, {});  
  
// histogram theo cat  
const hist = orders.reduce((acc,o)=>{  
 acc[o.cat] = (acc[o.cat]||0) + 1;  
 return acc;  
}, {});  
  
// index theo id  
const index = orders.reduce((acc,o)=> (acc[o.id]=o, acc), {});

## 5) Closure & Scope & IIFE

• Closure: hàm nhớ môi trường khi tạo ra nó; dùng để đóng gói trạng thái (counter) hoặc tạo factory.

• IIFE: hàm tự chạy ngay để tạo phạm vi cô lập (ngày xưa thay cho module).

function makeCounter(start=0){  
 let c = start; // biến 'c' được đóng gói trong closure  
 return {  
 next(){ return ++c; },  
 reset(){ c = 0; }  
 };  
}  
const counter = makeCounter(5);  
counter.next(); // 6  
counter.next(); // 7  
  
// IIFE  
(function(){  
 const secret = 42;  
 console.log('IIFE running...', secret);  
})();

## 6) this, call/apply/bind; arrow function không có this riêng

• this phụ thuộc cách gọi hàm. Khi tách method ra khỏi object, this mất ngữ cảnh → dùng bind.

• call/apply để gọi với this cụ thể; apply nhận mảng tham số. Arrow function kế thừa this ở nơi định nghĩa.

const logger = {  
 prefix: '[LOG]',  
 log(msg){ console.log(this.prefix, msg); }  
};  
  
logger.log('hello'); // [LOG] hello  
const f = logger.log; // tách ra => this mất  
try{ f('oops'); }catch(e){ console.log('this lost'); }  
  
const bound = logger.log.bind(logger);  
bound('ok'); // [LOG] ok  
  
function sum(a,b){ return a+b; }  
sum.call(null, 2, 3); // 5  
sum.apply(null, [2,3]); // 5  
  
// Arrow kế thừa this  
const obj = {  
 x: 10,  
 getXArrow: () => this.x, // this = ngữ cảnh file/module, không phải obj  
 getX(){  
 const inner = () => this.x; // arrow kế thừa this của getX (là obj)  
 return inner();  
 }  
};  
obj.getX(); // 10  
obj.getXArrow(); // thường là undefined trong strict mode

## 7) JSON: stringify / parse; replacer / reviver; clone sâu

• JSON.stringify(obj) → chuỗi; JSON.parse(str) → object.

• replacer (trong stringify) để loại bớt/biến đổi; reviver (trong parse) để khôi phục kiểu (vd Date).

• JSON clone không giữ function/Date/undefined/NaN/Infinity. Dùng structuredClone (trình duyệt mới) để clone sâu an toàn hơn.

const obj = { a:1, b:2, when: new Date() };  
const json = JSON.stringify(obj, (k,v)=> k==='b'? undefined : v); // loại bỏ b  
// reviver khôi phục Date  
const back = JSON.parse(json, (k,v)=> k==='when' ? new Date(v) : v);  
  
const deep = structuredClone ? structuredClone(obj) : JSON.parse(JSON.stringify(obj));

## 8) Tiện ích thường dùng: entries, uniqueBy, groupBy(fn)

• Object.entries/fromEntries giúp chuyển qua lại giữa object ↔ entries.

• uniqueBy: khử trùng lặp theo khóa; groupBy tổng quát theo hàm key.

function toEntries(obj){ return Object.entries(obj); }  
function fromEntries(entries){ return Object.fromEntries(entries); }  
  
function uniqueBy(arr, keyFn){  
 const seen = new Set();  
 const out = [];  
 for(const x of arr){  
 const k = keyFn(x);  
 if(!seen.has(k)){ seen.add(k); out.push(x); }  
 }  
 return out;  
}  
  
function groupBy(arr, keyFn){  
 return arr.reduce((acc, item)=>{  
 const k = keyFn(item);  
 (acc[k] ||= []).push(item);  
 return acc;  
 }, {});  
}  
  
// Ví dụ  
const people = [  
 {id:1, name:'An', city:'HN'},  
 {id:2, name:'Bình', city:'HCM'},  
 {id:3, name:'An', city:'HN'}  
];  
const uniqByName = uniqueBy(people, p=>p.name); // giữ bản ghi đầu tiên theo name  
const byCity = groupBy(people, p=>p.city);

## 9) Lỗi thường gặp & mẹo

• Quên copy trước khi sort/splice → mutate mảng gốc ngoài ý muốn.

• Dùng function thường trong callback nhưng cần this của object → đổi sang arrow/bind đúng ngữ cảnh.

• Nhầm lẫn giữa map (trả mảng mới) và forEach (không trả kết quả).

• Dùng JSON clone cho object chứa Date/func → mất dữ liệu kiểu; cân nhắc structuredClone hoặc tự xử lý.