# Week 01 — JS Basic 1 — Lý thuyết

## 1) Biến: var / let / const — phạm vi, hoisting, TDZ

• var có phạm vi theo hàm (function scope), được hoisting (khai báo được "kéo lên" đầu phạm vi, giá trị mặc định là undefined).

• let/const có phạm vi theo khối (block scope). Tồn tại vùng TDZ (Temporal Dead Zone): không thể truy cập trước khi khai báo.

• const cố định binding, không cho gán lại biến; với object/array, có thể thay đổi thuộc tính/phần tử bên trong.

// Hoisting với var  
console.log(a); // undefined  
var a = 10;  
  
// TDZ với let  
try { console.log(b); } catch (e) { console.log("TDZ:", e.message); }  
let b = 20;  
  
// const: binding không đổi nhưng object bên trong vẫn thay đổi  
const user = {name: "An"};  
user.name = "Bình"; // OK  
// user = {} // Lỗi: gán lại binding const

## 2) Kiểu dữ liệu nguyên thủy & typeof; truthy/falsy

• Nguyên thủy: number, string, boolean, null, undefined, symbol, bigint.

• Một số giá trị falsy: 0, NaN, '', null, undefined, false. Còn lại đa phần là truthy ([], {}, '0', ' ').

• typeof null === 'object' là một di sản lịch sử của JS.

typeof 42; // 'number'  
typeof 'hi'; // 'string'  
typeof true; // 'boolean'  
typeof undefined; // 'undefined'  
typeof null; // 'object' (đặc thù lịch sử)  
typeof Symbol('x'); // 'symbol'  
typeof 10n; // 'bigint'  
  
Boolean(''); // false  
Boolean('0'); // true  
Boolean([]); // true  
Boolean({}); // true

## 3) Ép kiểu: tường minh & ngầm định

• Tường minh: Number(), String(), Boolean(), parseInt()/parseFloat() (chú ý cơ số với parseInt).

• Ngầm định (coercion) xảy ra với +, ==, chuỗi nối, v.v. Khuyến nghị hạn chế so sánh lỏng (==).

Number('10') // 10  
String(10) // '10'  
Boolean(0) // false  
parseInt('08',10) // 8 (nên chỉ định cơ số)  
  
// Coercion ngầm  
'3' \* 2 // 6 (chuỗi bị ép sang number)  
'3' + 2 // '32' (toán tử + nối chuỗi nếu có chuỗi)  
+'10' // 10 (unary +)  
  
// isNaN vs Number.isNaN  
isNaN('abc') // true (coercion trước khi kiểm tra)  
Number.isNaN('abc') // false (không ép kiểu)  
Number.isNaN(NaN) // true

## 4) Toán tử số học, so sánh, logic; ?? và ?. (optional chaining)

• Ưu tiên dùng === thay vì == để tránh ép kiểu ngầm.

• || trả về toán hạng truthy đầu tiên; ?? chỉ coi null/undefined là "rỗng"; ?. giúp truy cập an toàn các thuộc tính lồng nhau.

// So sánh  
'10' == 10 // true (coercion) -> Tránh  
'10' === 10 // false (nghiêm ngặt) -> Nên dùng  
  
// Logic  
const displayName = input || 'Guest'; // nếu input falsy => 'Guest'  
const safeVal = maybeNull ?? 'fallback'; // chỉ null/undefined mới rơi về 'fallback'  
  
// Optional chaining  
const city = user && user.address && user.address.city; // Cũ: rườm rà  
const city2 = user?.address?.city; // Mới: ngắn gọn

## 5) Hàm cơ bản: declaration, expression, arrow; default param; return

• Declaration được hoisting; expression/arrow không.

• Arrow function không có this/arguments riêng, phù hợp hàm ngắn/pure function.

• Dùng tham số mặc định để chống undefined; ưu tiên early-return để giảm lồng nhau.

// Declaration  
function sum(a=0, b=0){  
 return a + b;  
}  
  
// Expression  
const isEven = function(n){  
 return typeof n === 'number' && n % 2 === 0;  
};  
  
// Arrow (ngắn gọn)  
const greet = (name='bạn') => `Hello, ${name}!`;  
  
// Early-return + validate  
function safeAdd(a, b){  
 const x = Number(a), y = Number(b);  
 if (Number.isNaN(x) || Number.isNaN(y)) return 'Invalid input';  
 return x + y;  
}

## 6) Template literals & định dạng số

• Dùng backtick `...` và nội suy ${expr}; hỗ trợ xuống dòng nhiều hàng.

• Định dạng số: toFixed, Intl.NumberFormat('vi-VN').

const name = 'Lan', age = 20;  
const s = `Xin chào, tôi là ${name}, ${age} tuổi.`;  
  
// Định dạng  
const total = 1234567.8912;  
total.toFixed(0); // '1234568'  
new Intl.NumberFormat('vi-VN').format(total); // '1.234.568'

## 7) Mẫu thực hành hữu ích (pattern)

• Guard clause cho validate đầu vào; tách hàm nhỏ, pure function; đặt tên rõ ràng; console.log có nhãn.

function calcRect(width, height){  
 const w = Number(width), h = Number(height);  
 if (!Number.isFinite(w) || !Number.isFinite(h) || w<=0 || h<=0) {  
 return 'Invalid input';  
 }  
 const P = 2\*(w+h), S = w\*h;  
 return { P, S };  
}  
  
// Test nhanh  
console.log(calcRect('3','4')); // {P:14, S:12}  
console.log(calcRect('a','4')); // 'Invalid input'