Day 2 — Control Flow + Arrays & Objects

1. Control Flow

Control flow determines how the program decides what code to execute depending on conditions.

Common structures: if/else, switch, and loops.

a) if / else

Used to execute code only when a condition is true. Example: if (score >= 90) {

console.log("Excellent"); } else if (score >= 70) { console.log("Good"); } else { console.log("Try

again"); }

b) switch

Used when you want to test a single value against multiple cases. Example: switch(day) { case

"Monday": console.log("Start of week"); break; case "Friday": console.log("Weekend soon"); break;

default: console.log("Normal day"); }

c) Loops

Loops allow repetition of code. - for: when you know how many times to iterate. - while: continues

while a condition is true. - do...while: runs once before checking. - for...of: iterates over values in

arrays. - for...in: iterates over keys in objects.

2. Arrays

Arrays store ordered lists of values. Access elements using indexes starting from 0. let numbers =

[1, 2, 3]; console.log(numbers[0]); // 1 Common methods: push(), pop(), shift(), unshift(), slice(),

splice(), map(), filter(), reduce().

3. Objects

Objects hold key-value pairs. Example: const user = { name: "Ali", age: 25 }; user.country = "Egypt";

// Add property console.log(user.name); // 'Ali'

4. Spread & Destructuring

Spread (...) copies or merges arrays/objects. let arr1 = [1,2], arr2 = [3,4]; let merged = [...arr1,

...arr2]; // [1,2,3,4] Destructuring allows unpacking values: const { name, age } = user; const [first,

second] = [10, 20];

5. List Transforms (map / filter / reduce)

- map(): transforms each element. [1,2,3].map(n => n*2); // [2,4,6] - filter(): selects elements by

condition. [1,2,3,4].filter(n => n%2===0); // [2,4] - reduce(): accumulates all into one value.

[1,2,3].reduce((a,b)=>a+b,0); // 6

6. Safe Object Merges

Combine multiple objects safely: const defaults = { theme: "light", lang: "en" }; const user = { lang:

"ar" }; const merged = { ...defaults, ...user }; // { theme: "light", lang: "ar" } To merge nested objects

safely, merge their sub-keys separately.

7. Tiny Collection Utilities

const utils = { sum: arr => arr.reduce((a,b)=>a+b,0), average: arr =>

arr.reduce((a,b)=>a+b,0)/arr.length, unique: arr => [...new Set(arr)], mergeObjects: (...objs) =>

objs.reduce((acc,obj)=>({...acc,...obj}),{}) }; Examples: utils.sum([1,2,3]); // 6 utils.average([2,4,6]); //

4 utils.unique([1,1,2]); // [1,2] utils.mergeObjects({a:1},{b:2}); // {a:1,b:2}

8. Summary

- Use if/switch to control program logic. - Handle arrays and objects with modern syntax. - Utilize

map, filter, and reduce for elegant transformations. - Always favor immutability and safe merging

patterns.