JavaScript Debug & Refactor Lab

Question Sheet

General Instructions

• Do not delete the given code

• Do not run the code immediately

• First predict the output

• Then explain the behavior

• Then fix or refactor the code

• Use modern ES6 syntax where applicable

• Work inside a Git repository as instructed by the instructor

Question 1: Execution Context & Hoisting

console.log(a); var a = 10; function test() { console.log(a); var a = 20; } test(); console.log(a); Tasks: • Predict the output
• Explain the behavior using execution context and hoisting
• Refactor the code to avoid confusion

Ans A) - undefined - undefined - 10
Ans B) “a” is var, so it is hoisted but uninitialised. so at first log its undefined. inside the function it is shadowed, and again as a var hoisted and uninitialised. so again undefined. then “10” for the third log, since by then the value 10 is assigned to global “a”
Ans C) const globalA=10;function test(){console.log(20)}console.log(10),test(),console.log(10); //use let-const and meaningful names. try to ensure proper inits before code

Question 2: Call Stack Tracing

function first() { second(); console.log("First"); } function second() { third(); console.log("Second"); } function third() { console.log("Third"); } first(); Tasks: • Determine the order of execution
• Describe the call stack sequence
• Identify the maximum call stack depth

Ans A) third- -> second -> first
Ans B) first- -> second -> third
Ans C) 3

Question 3: Type Coercion Bug

function calculateTotal(price, tax) { return price + tax; } calculateTotal("100", 18); Tasks: • Identify the bug
• Explain why the bug occurs
• Fix the bug using two different approaches
• Select the safer approach and justify it

Ans A) since a parameter is passed as string, concatenation happes
Ans B) 1> forre typecasting (Number, parseInt) 2> Check inputs (Number.isFinite).
Ans C) 2> typecasting will still not guarantee

Question 4: Scope (var, let, const)

for (var i = 0; i < 3; i++) {} console.log(i); Tasks: • Explain why this code prints the observed value
• Fix the code using let
• Explain the scope difference involved

Ans A) 3 - as “var i” is hoisted
Ans B) let i=0; for (i; i < 3; i++) {} console.log(i);
Ans C) 2> By moving to “let” and outside the loop, the code is much more understanble and predicatable

Question 5: Arrow Functions and this

const counter = { count: 0, increment: () => { this.count++; } }; counter.increment(); console.log(counter.count); Tasks: • Predict the output
• Explain the behavior of this
• Refactor the code so that it works correctly

Ans A) 0
Ans B) Since it is a lambda function, the this keyword points to the global scope. this.count is undefined, and incrementing it becomes NaN. the object property remains 1.
Ans C) const counter = { count: 0, increment: function () { this.count++; } }; counter.increment(); console.log(counter.count);

Question 6: Control Flow Refactoring

function getDayType(day) { if (day === "Saturday") { return "Weekend"; } else if (day === "Sunday") { return "Weekend"; } else { return "Weekday"; } } Tasks: • Refactor the code using a switch statement
• Add handling for invalid input
• Comment on readability and maintainability

Ans A, B) function getDayType(e){switch(e)
{case"Monday":case"Tuesday":case"Wednesday":case"Thursday":case"Friday":return"Weekday";case"Saturday":case"Sunday":return"Weekend";default:return"Invalid day"}}
Ans C) For exact values, switch case is better for readability and mantainabiliity, and also faster for execution.

Question 7: Loop Selection

const scores = [10, 20, 30, 40]; Tasks: • Iterate over the array using a for loop
• Iterate over the array using a for…of loop
• Explain when each loop is preferable

Ans A) for (let score of scores){}
Ans B) for(let i=0;i<scores.length;i++){}
Ans C) for of shortens the syntax, but the normal for loop allows for more sophistacted checks and increments

Question 8: Array Method Refactor

const numbers = [1, 2, 3, 4, 5]; let result = []; for (let i = 0; i < numbers.length; i++) { if (numbers[i] % 2 === 0) { result.push(numbers[i] * 2); } } Tasks: • Refactor the code using filter and map
• Explain why forEach is not suitable here
• Preserve the original code as comments

Ans A) let result = numbers.filter(number=> number%2==0).map(aNumber=> aNumber *2);
Ans B) We will need to declare the array first, whereas given methods make it much shorter and easier.
Ans C) /*const numbers = [1, 2, 3, 4, 5]; let result = []; for (let i = 0; i < numbers.length; i++) { if (numbers[i] % 2 === 0) {
result.push(numbers[i] * 2); } }*/

Question 9: reduce (Advanced)

const cart = [{ item: "Book", price: 200 }, { item: "Pen", price: 50 }, { item: "Bag", price: 750 }]; Tasks: • Compute the total price using reduce
• Add 18% tax to the total
• Handle the case where the cart is empty

Ans A) let totalBeforeTax = cart.reduce((sum,{price})=> sum=sum+price, 0);
Ans B) let totalAfterTax = totalBeforeTax * 1.18;
Ans C) if(cart.length<1) throw new Error(“Cart Emprty”); (Although, reduce can handle an empty array)

Question 10: Real-World Debugging

const users = [{ name: "A", active: true }, { name: "B", active: false }, { name: "C", active: true }]; const activeUsers = users.map(user => { if (user.active) { user.name; } }); console.log(activeUsers); Tasks: • Identify the logical bug without executing the code
• Fix the code using the appropriate array method
• Explain the difference between map and filter in this context

Ans A) since we are using “map” , it has to return a value for each index. also, there are no return statements in the function.
Ans B) we should use filter instead. const activeUsers = users.filter(aUser=> aUser.active).map(aUser=>aUser.name);
Ans C) An array for activeUsers should only have the active users. map returns a new array, where each item somehow (decuced in a function) relates to the same-index item in an orginal array. Filter simply extracts items that match a certain criteria into a new array.