Q1 1- JavaScript has two main types of hoisting: variable hoisting and function hoisting. 2- It allows you to call methods and access properties of a parent class from a subclass. 3- let ->Variables declared with let have block-level scope const -> Variables declared with const are also block-scoped var -> Variables declared with var have function-level scope 4- 5- readonly allows the content to be viewed but not edited. The content is selectable, and the user can copy it. disabled makes the textarea non-interactive. The content is visually disabled, and the user can't interact with it in any way. 6- Absolute units -> Pixels ,inches Relative Units -> Em ,Rem Viewpport Units-> vm ,vh ,vmin, vmax 7- allows you to specify the preferred font or a list of font choices for text elements. 8- Method 1: Flexbox Method 2: Absolute Positioning and Transform

Q2 1- T 2- T 3- T 4- F 5- F 6- T 7- T 8- T

Q3 1- asynchronous, non-blocking, single-threaded language 2- Abstraction

Q4 1- Error: The Fails! Error: The Fails! Error: The Fails! 2- Lydia Hallie 3- 6 8 4- orange 5- 6-0 1 4 2 3 7- i is not defined 8- Hello World 10 9- [59.52, 83.7, 93] 10- ['batman', 'bane'] Q5 1- function recursiveStringLength(str) { if (str === '') { return 0; } else { return 1 + recursiveStringLength(str.slice(1)); } }

const inputString = "Hello, world!"; const length = recursiveStringLength(inputString); console.log("Length:", length); // Output: Length: 13

2- function printMultiplicationTable() { for (let i = 1; i <= 12; i++) { for (let j = 1; j <= 12; j++) { const result = i * j; console.log(`${i} * ${j} = ${result}`); } console.log(); } }

printMultiplicationTable();

3- function getElementsAtOddPositions(list) { const oddPositionElements = []; for (let i = 0; i < list.length; i++) { if (i % 2 !== 0) { oddPositionElements.push(list[i]); } } return oddPositionElements; }

const inputList = [1, 2, 3, 4, 5, 6, 7, 8, 9]; const oddPositionElements = getElementsAtOddPositions(inputList); console.log("Elements at Odd Positions:", oddPositionElements);

4- 5-html <!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <link rel="stylesheet" href="styles.css"> <title>Background Generator</title> </head> <body> <h1>Background Generator</h1> <input type="color" id="color1" value="#ff5733"> <input type="color" id="color2" value="#2c3e50"> <button id="generateBtn">Generate</button>   <script src="script.js"></script> </body> </html> css body { margin: 0; display: flex; justify-content: center; align-items: center; min-height: 100vh; background: linear-gradient(to right, #ff5733, #2c3e50); font-family: Arial, sans-serif; }

.container { text-align: center; background: rgba(255, 255, 255, 0.8); padding: 20px; border-radius: 10px; box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.2); }

h1 { margin-top: 0; }

.controls { margin-top: 20px; }

input[type="color"] { margin: 0 5px; padding: 5px; }

button { padding: 5px 10px; background-color: #3498db; border: none; color: white; border-radius: 5px; cursor: pointer; }

button:hover { background-color: #2980b9; } js const color1Input = document.getElementById("color1"); const color2Input = document.getElementById("color2"); const generateBtn = document.getElementById("generateBtn"); const container = document.querySelector(".container");

function generateBackground() { const color1 = color1Input.value; const color2 = color2Input.value; container.style.background = `linear-gradient(to right, ${color1}, ${color2})`; }

generateBtn.addEventListener("click", generateBackground);

// Initial background generation generateBackground();