problem 1:The Persistent Counter

Yes.counterA() creates maintain independent counters.Each call to createCounter() creates a new scope .

console.log(counterA()) -> gives output as 1

console.log(counterA()) -> gives output as 2

console.log(counterB()) -> gives output as 1

Problem 2: The Delayed Greeter

Because the variable i is declared with var, all the delayed functions share the same i.

When the loop finishes, i is 3, and names[3] doesn’t exist so it’s undefined.

we can fix it by using let instead of var.

Section 2 Assignments :

Part 1: Variable Scope and Hoisting - The Case of the Missing Declaration

If the declared variable is var we get undefined as output.

If the variable is let or const we get error.

If var is declared inside the function it does not affect anything outside the function.

There's a global mysteryVariable, and a separate one inside the function.

The one inside the function only works inside that function.

Task:

1)console.log(mysteryVariable);

let mysteryVariable = 10;

console.log(mysteryVariable);

function revealMystery() {

console.log("Inside revealMystery:", mysteryVariable);

let mysteryVariable = 20;

console.log("Inside revealMystery (after declaration):", mysteryVariable);

}

revealMystery();

console.log("After revealMystery:", mysteryVariable);

->with var ,the variable is hoisted and initialized with undefined,so we can access it before its declared.

->with let ,the vatiable is hoisted but not initialized,so accessing it before the declaration causes a referenceError.

2) **Hoisting** moves variable and function declarations to the top of their scope before running the code.  
When you use **var**, the variable is hoisted and you can access it before it's declared (but it will be undefined).  
With **let** and **const**, the variables are also hoisted, but they aren’t initialized . Instead, they stay in a **“temporal dead zone”**. If you try to use them before that, you’ll get a **ReferenceError**.

Part 2: The Perils of this - Object Methods and Arrow Functions  
Task:

1) inside the setTimeout callback, this does **not** refer to the user object but to the global context, so this.name gives undefined.

2)By saving this in a variable like self the inner function can use this variable for correctly reference.

3) Arrow functions stores this from the surrounding function, so this continues to point to the user object this is known as lexical binding.

4) const obj = {

name: "Indhu",

showName: function() {

setTimeout(() => {

console.log(this.name); // Sam

}, 1000);

}

};

obj.showName();

Part 3: The Power of Closures - Maintaining State

->The increment and decrement functions can access the count variable because they are inside setupCounter.  
->Each time setupCounter is called, it creates a new count variable.  
So, counterOne and counterTwo have separate, independent count values.

Task:

1)closure means a function remembers and uses that variable from the place it is created even after execution. Increment and decrement functions remember the count from setupCounter.  
so each counter keeps its own count separately.so they don't affect each other.

2) function createGreeting(greeting) {

return function(name) {

return greeting + ", " + name + "!";

};

}

const sayHello = createGreeting("Hello");

console.log(sayHello("World")); // Output: Hello, World!

3) function createSecretHolder(secret) {

return {

getSecret: function() {

return secret;}’

setSecret: function(newSecret) {

secret = newSecret;

}

};

const holder = createSecretHolder("mySecret");

console.log(holder.getSecret()); // Output: mySecret

holder.setSecret("newSecret");

console.log(holder.getSecret()); // Output: newSecret

4) function createSecretHolder(secret) {

return {

getSecret: function() {

return secret;

},

setSecret: function(newSecret) {

secret = newSecret;

}

};

}

const holder = createSecretHolder("mySecret");

console.log(holder.getSecret()); // Output: mySecret

holder.setSecret("newSecret");

console.log(holder.getSecret()); // Output: newSecret

console.log(holder.secret); // Output: undefined