ES6 Declaration Assignments:

Let's craft some assignments to solidify your understanding of let and const in ES6 (ECMAScript 2015) JavaScript. These will cover scoping, reassignment, and best practices.

Basic Scoping and Declaration:

1. Variable Shadowing:

JavaScript

let x = 10;

function myFunction() {

let x = 20;

console.log(x); // What will be logged?

}

myFunction();

console.log(x); // What will be logged?

// Explain the output and the concept of variable shadowing.

1. Const Reassignment (Tricky!):

JavaScript

const myArray = [1, 2, 3];

myArray.push(4);

console.log(myArray); // Will this work? Why or why not?

const myObject = { a: 1 };

myObject.b = 2;

console.log(myObject); // Will this work? Why or why not?

// Explain the difference between reassigning a const variable and mutating its properties.

1. Block Scope Practice:

JavaScript

if (true) {

let blockVar = "I'm block scoped";

const blockConst = 123;

}

console.log(blockVar); // Will this cause an error? Why?

console.log(blockConst); // Will this cause an error? Why?

// Explain how block scope works with let and const.

Deeper Understanding and Usage:

1. Looping with let (Important!):

JavaScript

for (let i = 0; i < 5; i++) {

setTimeout(() => console.log(i), 1000);

}

// Explain the output. How is this different from using var? Why is let preferred in loops?

1. Const and Immutability (Important!):

JavaScript

const user = { name: "Alice", age: 30 };

// Try to make the user object truly immutable using Object.freeze().

// Demonstrate that you cannot reassign or add properties after freezing.

1. Best Practices:
   * When should you use const? What does it communicate about your intentions?
   * When is let more appropriate?
   * Why should you avoid var in modern JavaScript?

Challenging Scenarios:

1. Closure and let:

JavaScript

function createCounter() {

let count = 0;

return {

increment: () => ++count,

decrement: () => --count,

};

}

const counter = createCounter();

console.log(counter.increment()); // What will be logged?

console.log(counter.decrement()); // What will be logged?

// Explain how closure works with let and how the counter maintains its state.

1. let in Nested Functions:

JavaScript

function outerFunction() {

let outerVar = "Outer";

function innerFunction() {

let innerVar = "Inner";

console.log(outerVar); // Can innerFunction access outerVar?

console.log(innerVar);

}

innerFunction();

console.log(innerVar); // Will this cause an error? Why?

}

outerFunction();

These assignments will help you explore the nuances of let and const, covering scoping rules, reassignment behavior, and best practices. Understanding these concepts is essential for writing robust and maintainable JavaScript code. Let me know if you have any other questions or would like more variations on these exercises!