Intermediate JavaScript Programming

LESSON 1: Advanced JavaScript Fundamentals

Learning Objectives:

By the end of this lesson, participants will be able to:

- Understand the difference between `var`, `let`, and `const`.
- Use closures to retain scope.
- Avoid polluting the global namespace.
- Organize code with ES6 module syntax.

Lesson Outline:

I. Variable Declarations and Scope (10 min)

- `var` is function-scoped and allows redeclaration.
- `let` and `const` are block-scoped and prevent redeclaration.
- `const` must be initialized and cannot be reassigned.
- Prefer `let` and `const` for predictable behavior and scoping.

Exercise 1: Predict the output

```
function testScope() {
  var x = 1;
  if (true) {
    let x = 2;
    console.log(x); // ?
  }
  console.log(x); // ?
}
```

Answer:

```
2
1
```

Short Answer:

Why is `let` safer than `var` inside a block? (Expected Answer: `let` is block-scoped, preventing accidental overrides.)

II. Closures (15 min)

- A closure is a function that retains access to its lexical scope.
- Useful for data privacy and function factories.

Example:

```
function makeCounter() {
  let count = 0;
  return function() {
    count++;
    return count;
  };
}

const counter = makeCounter();
console.log(counter()); // 1
console.log(counter()); // 2
```

Exercise 2:

Write a function `makeMultiplier(factor)` that returns a function multiplying input by `factor`.

Answer:

```
function makeMultiplier(factor) {
  return function(x) {
    return x * factor;
  };
}
const double = makeMultiplier(2);
console.log(double(5)); // 10
```

III. Avoiding Global Variables (10 min)

- Variables declared without `let`, `const`, or `var` become global.
- Too many global variables increase the risk of name collisions.

Bad:

```
function badPractice() {
  accidentalGlobal = 42; // Becomes a global variable
}
```

Better:

```
function goodPractice() {
  let local = 42; // Proper scope
}
```

Exercise 3:

Refactor this code to avoid creating global variables.

```
function start() {
  counter = 0;
}
```

Answer:

```
function start() {
  let counter = 0;
}
```

IV. Modules and import/export (15 min)

- Use `export` to expose functions or constants.
- Use `import` to include them in other files.
- Encourages better code organization.

Example:

```
// mathUtils.js
export function square(x) {
  return x * x;
}
```

```
// main.js
import { square } from './mathUtils.js';
console.log(square(5)); // 25
```

Exercise 4:

Split this code into two modules, one exporting `greet`, one importing and calling it.

```
function greet(name) {
  return `Hello, ${name}`;
}
console.log(greet("Greg"));
```

Answer:

```
// greeter.js
export function greet(name) {
  return `Hello, ${name}`;
}

// main.js
import { greet } from './greeter.js';
console.log(greet("Greg"));
```

Notes on Module Use in Different Environments

1. **In Web Browsers**:
   - Use `<script type="module">` to enable `import`/`export`:

```
<script type="module" src="main.js"></script>
```

- Modules must be served over HTTP (not opened from the file system).
- You must use relative paths (e.g., `./utils.js`).

2. **In Node.js**:
   - Use `.mjs` extension **or** add `"type": "module"` to `package.json`.
   - Otherwise, Node uses CommonJS syntax (`require`, `module.exports`).

3. **In Frameworks (React/Vue)**:
   - Tools like Webpack or Vite support module imports like `import React from 'react';`.

Browser-based example:

```
// utils.js
export function sayHello(name) {
  return `Hello, ${name}`;
}
```

```
// main.js
import { sayHello } from './utils.js';
document.body.textContent = sayHello("World");
```

```
<!-- index.html -->
<!DOCTYPE html>
<html>
<head><title>Module Example</title></head>
<body>
```

```
  <script type="module" src="main.js"></script>
</body>
</html>
```

V. Recap & Q&A (10 min)

- Review of scoping, closures, global avoidance, and modules.
- Emphasize modern best practices using `let`, `const`, and modules.

Final Multiple-Choice Question:

Which statement about closures is TRUE? A. Closures do not remember variables after returning. B. Closures only work with global variables. C. Closures retain access to the scope in which they were created. D. Closures only work with `var` declarations.

(Answer: C. Closures retain access to the scope in which they were created.)