

**Closure:** A closure is a function that remembers variables from its outer scope even after the outer function finishes. Used in data privacy and function factories.

**Switch without break:** Without break, execution falls through to the next case. This can cause unintended logic execution.

**null vs undefined:** undefined means a variable is declared but not assigned. null is an intentional empty value.

**Prototype:** Prototype allows objects to inherit properties and methods. It is the base of JavaScript inheritance.

**Why ES6:** ES6 introduced modern syntax for cleaner, scalable code. It improved readability and developer productivity.

**ES5 vs ES6:** ES5 uses var and function-based syntax. ES6 adds let/const, arrow functions, classes, modules.

**Block scope:** Block scope limits variable access to {} blocks. let and const are block scoped.

**Arrow function:** Arrow functions are shorter syntax for functions. They don't have their own this.

**Arrow function this:** Arrow functions inherit this from their parent scope. This avoids context binding issues.

**Template literals:** Template literals use backticks for string interpolation. They make strings readable and dynamic.

**Destructuring:** Destructuring extracts values from arrays or objects. Object uses keys, array uses position.

**ES5 to ES6:** ES5 uses function keyword. ES6 replaces it with arrow functions.

**Swap numbers:** Use arithmetic or destructuring assignment to swap without temp variable.

**Sync vs Async:** Synchronous blocks execution. Asynchronous allows non-blocking operations.

**Promise:** A promise represents a future value of an async operation. It handles async results.

**Promise states:** Pending: initial, Fulfilled: success, Rejected: failure.

**Callback vs Promise:** Callbacks cause nesting issues. Promises provide better chaining and error handling.

**Async/Await:** Async/await simplifies promise handling. It looks like synchronous code.

**Why async/await:** It improves readability and error handling using try/catch.

**Unhandled rejection:** Unhandled rejections cause runtime warnings or crashes. Always handle errors.

**Optional chaining:** Optional chaining prevents errors when accessing nested properties. It returns undefined safely.

**Nullish coalescing:** ?? provides default only for null or undefined values.

**|| vs ??:** || checks all falsy values. ?? checks only null or undefined.

**Event loop:** Event loop manages async tasks execution. It checks queues continuously.

**Microtask vs Callback:** Microtasks have higher priority than callbacks. Promises use microtask queue.

**setTimeout:** setTimeout delays execution. It is asynchronous but not precise timing.