(Notes) Closures

Question 1: What is Lexical Scope?
Explanation: Lexical scope refers to how variable names are resolved in nested functions based on where the functions are defined. In the provided code, the function `local` has a local scope where it can access variables defined within itself as well as variables in the global scope, such as the `username` variable.

Question 2: How does Closure work?
Explanation: Closure is a JavaScript feature that allows a function to remember and access its lexical scope even when the function is executed outside that scope. In the code snippet, `makeFunc` creates a closure around the `name` variable, enabling `displayName` to access `name` even after `makeFunc` has finished running.

Question 3: Explain Closure Scope Chain.
Explanation: Closure scope chain refers to the hierarchy of nested functions and their respective scopes that closures have access to. The `sum` function creates nested closures, allowing access to variables like `a`, `b`, `c`, `d`, and `e` from different levels of scope within the chain.

Question 4: What will be the output of the provided code snippet?
Explanation: The code snippet defines an immediately invoked function expression (IIFE) that declares a global variable `count` and a local variable `count` inside the `printCount` function. The output will be **1 followed by 0** because of variable shadowing and the conditional check inside `printCount`.

Question 5: Write a function similar to addSix() using closures.
Explanation: The `createBase` function returns a closure that adds a base number (6 in this case) to an inner number provided as an argument. The `addSix` function created using `createBase(6)` adds 6 to its argument when invoked.

Question 6: How can closures be used for time optimization?
Explanation: The `find` function precomputes an array of squares and returns a closure that

can access this precomputed data. This approach optimizes time by avoiding redundant computations, as demonstrated by the timed execution of `closure(6)` and `closure(50)`.

Question 7: How would you create a private counter using closure?
Explanation: The `counter` function utilizes closure to create a private variable `_counter` that can only be accessed and modified through the returned functions `add` and `retrieve`, ensuring data privacy and controlled access.

Question 8: Explain the Module Pattern and provide an example.
Explanation: The Module Pattern uses an immediately invoked function expression (IIFE) to encapsulate private variables and functions, exposing only a public interface. In the given example, the `module` object has a public method `publicMethod`, while `privateMethod` remains hidden within the module.

Question 9: How can you ensure a function runs only once using closure?
Explanation: The `Like` function returns a closure that tracks the number of times it has been called, ensuring its action (like subscribing) is performed only once. Subsequent calls to the closure display a message indicating that the action has already been performed.

Question 10: Explain the Once Polyfill using closure.
Explanation: The `once` function creates a closure that allows a given function to be executed only once. Upon the first call, the original function runs, and subsequent calls return the result of the initial execution, demonstrating how closure can control function execution.

Question 11: What is Memoize Polyfill, and how does it use closure?
Explanation: The `myMemoize` function is a memoization polyfill that caches function results based on arguments, using closure to store and retrieve cached values efficiently. This improves performance by avoiding redundant computations for repeated function calls with the same arguments.

Question 12: Differentiate between Closure and Scope.
Explanation: Closure refers to a function's ability to retain access to variables from its lexical

scope even after that scope has closed, while scope refers to the visibility and accessibility of variables within a specific context, such as global scope, function scope, or block scope.
