Title: Scope and Closures in JavaScript

1. What Is Scope
   Scope determines where variables can be accessed in your code.
   There are three main types:

- Global Scope: accessible everywhere.
- Function Scope: variables defined inside a function.
- Block Scope: variables defined inside {} when using let or const.
  Example:
  let x = 10; // global
  function showNumber() {
  let y = 5; // local
  console.log(x + y);
  }
  showNumber();

2. Lexical Scope
   JavaScript uses lexical (static) scope, meaning inner functions can access variables from their outer functions.
   Example:
   function outer() {
   let message = "Hello";
   function inner() {
   console.log(message); // can access outer variable
   }
   inner();
   }
   outer();

3. What Is a Closure
   A closure happens when a function remembers the variables from where it was created, even after that outer function finishes.
   Example:
   function counter() {
   let count = 0;
   return function () {

```
count++;
console.log(count);
};
}
const increment = counter();
increment(); // 1
increment(); // 2
```

Here, the inner function keeps access to count, even though counter() has already finished running.

4.

Why Closures Are Useful
Closures are powerful for:

- Creating private variables
- Writing factory functions
- Managing state across function calls

Example:

```
function createUser(name) {
return {
getName: () => name,
};
}
const user = createUser("Sherif");
user.getName(); // "Sherif"
```