ANSWER 3-

‘Var’ is a keyword used to declare variables in JavaScript, and it has been available since the early versions of the language. On the other hand, ‘Let’ is a newer keyword introduced in ECMAScript 2015 (ES6) to provide block-level scoping and address some of the issues associated with var.

The main difference between var and let lies in their scope and behavior:

Scope: Variables declared with var have function-level scope or global scope, depending on where they are declared. This means that they are accessible throughout the entire function or, if declared outside of any function, they become global variables accessible throughout the entire program. Variables declared with let, however, have block-level scope. A block is any section of code wrapped within curly braces {}. Variables declared with let are only accessible within the block they are defined in, including nested blocks.

Hoisting: Variables declared with var are hoisted to the top of their scope during the compilation phase of JavaScript execution. This means you can use var variables before they are declared, although they will have the value undefined until they are assigned a value. Variables declared with let are also hoisted, but they are placed in the Temporal Dead Zone (TDZ) until they are actually declared in the code. If you try to access a let variable before its declaration, JavaScript will throw an error.

Re-declaration: Variables declared with var can be re-declared within the same scope without any issues. This can lead to accidental redeclaration bugs and can make the code harder to maintain. Variables declared with let, however, cannot be re-declared within the same scope. If you try to redeclare a let variable with the same name, JavaScript will throw an error.

To summarize, the main differences between var and let are in their scope (function-level vs. block-level), hoisting behavior, and re-declaration rules. let provides more predictable and safer variable scoping within blocks, while var has broader and potentially more error-prone scoping rules.