The statement is partially correct but requires some clarification. Let's break it down to better understand the **scopes** in Node.js and the browser:

Scope in Node.js vs. Browser

1. **Browser Environment**:
   - In browsers, the **global scope** is tied to the `window` object.
   - Variables declared with `var` in the global scope become properties of the `window` object. However, variables declared with `let` or `const` do not.
   - For example:

```
var x = 10;       // Adds x to the global `window` object
let y = 20;       // Not added to the `window` object
const z = 30;     // Not added to the `window` object

console.log(window.x); // 10
console.log(window.y); // undefined
console.log(window.z); // undefined
```

2. **Node.js Environment**:
   - In Node.js, there is no `window` object. Instead, the global object is called `global`.
   - Unlike browsers, the top-level scope in Node.js modules is not the global scope; it is the **module scope**.
   - For example:

```
var x = 10;       // Defined in the module scope, not added to `global`
let y = 20;       // Also module-scoped
const z = 30;     // Module-scoped

console.log(global.x); // undefined
```

   To explicitly add a variable to the global object in Node.js:

```
global.a = 50; // Adds `a` to the global scope in Node.js
console.log(global.a); // 50
```

Key Differences

Feature	Browser	Node.js
Global Object	`window`	`global`

Feature	Browser	Node.js
Top-Level Scope	Global (in scripts outside functions)	Module-scoped
`var` **Behavior**	Becomes a property of `window`	Local to the module; does not attach to `global`.
`let` **and** `const`	Block-scoped, not added to `window`.	Block-scoped, not added to `global`.
Implicit Globals	Attached to `window`.	Attached to `global`.

Example: Top-Level Scope Behavior

In Browser:

```
var a = 10;       // Global scope, added to `window`
let b = 20;       // Block scope, not added to `window`

console.log(window.a); // 10
console.log(window.b); // undefined
```

In Node.js:

```
var a = 10;       // Module-scoped, not global
let b = 20;       // Block-scoped, not global

console.log(global.a); // undefined
console.log(global.b); // undefined
```

Conclusion

- **Node.js** uses module scope by default, meaning variables declared at the top level of a file are local to that file/module and do not leak into the global scope (unless explicitly added to `global`).
- **Browser** code is more prone to polluting the global scope, especially with `var` or undeclared variables.

Understanding these differences helps avoid accidental global variable creation and ensures your code behaves as expected in both environments!