variable scope in TypeScript involves knowing how variables are accessible and visible within different parts of the code. Below are examples illustrating the different scopes: block scope, function scope, and module scope.

**Block Scope**

Block scope is defined by curly braces {} and applies to variables declared with let and const. These variables are only accessible within the block where they are defined.

**Example:**

function blockScopeExample() {

if (true) {

let blockScopedVariable = "I'm block scoped!";

const anotherBlockScopedVariable = "I'm also block scoped!";

console.log(blockScopedVariable); // Accessible here

console.log(anotherBlockScopedVariable); // Accessible here

}

// console.log(blockScopedVariable); // Error: blockScopedVariable is not defined

// console.log(anotherBlockScopedVariable); // Error: anotherBlockScopedVariable is not defined

} blockScopeExample();

### Function Scope

Function scope applies to variables declared with var. These variables are accessible anywhere within the function they are declared in, regardless of block scope.

function functionScopeExample() { if (true) { var functionScopedVariable = "I'm function scoped!"; } console.log(functionScopedVariable); // Accessible here } functionScopeExample();

### Module Scope

In TypeScript, each file is a module if it contains at least one import or export statement. Variables declared at the top level of a module are scoped to that module and are not accessible outside unless exported.

/ moduleA.ts

export const moduleScopedVariable = "I'm module scoped!";

// moduleB.ts

import { moduleScopedVariable } from './moduleA';

console.log(moduleScopedVariable); // Accessible here

### Global Scope

Variables declared outside of any function or block are in the global scope. They are accessible from anywhere in the code. However, global scope should be used sparingly to avoid conflicts and unpredictable behavior.

let globalVariable = "I'm global!"; function globalScopeExample() { console.log(globalVariable); // Accessible here } console.log(globalVariable); // Accessible here globalScopeExample();

### Shadowing

Shadowing occurs when a variable declared within a certain scope has the same name as a variable declared in an outer scope. The inner variable shadows the outer variable within its scope.

Example:

let name = "Alice"; function shadowingExample() { let name = "Bob"; // Shadows the outer 'name' variable console.log(name); // Outputs: Bob } console.log(name); // Outputs: Alice shadowingExample();

### Best Practices

1. **Use let and const**: Prefer let and const over var to leverage block scope and avoid issues with variable hoisting and unintended scope leaks.
2. **Minimize Global Variables**: Limit the use of global variables to avoid conflicts and unpredictable behavior.
3. **Modularize Code**: Use modules to encapsulate and organize code, reducing the risk of variable conflicts and improving maintainability.

### Summary

* **Block Scope**: Defined by {}. Use let and const for variables confined to a specific block.
* **Function Scope**: Defined within a function. Use var for variables scoped to a function, but this is generally discouraged in favor of let and const.
* **Module Scope**: Applies to top-level variables in a module. Use import and export to manage module scope.
* **Global Scope**: Applies to variables declared outside of any function or block. Use sparingly to avoid conflicts.
* **Shadowing**: Occurs when a variable in a certain scope has the same name as a variable in an outer scope.