Understanding Scopes and Hoistin

10000 CODERS
Right path for a Bright Career.

Scopes and Hoisting:

Scopes govern the accessibility of variables and functions within your code.

Hoisting, on the other hand, is a unique behavior where declarations are treated differently than assignments in JavaScript. Understanding these concepts is essential for writing clean, predictable, and maintainable JavaScript code.

What is Scope?

- Scope defines the accessibility (visibility) of variables and functions within your program.
- Different parts of your code have distinct scopes.
- Variables declared within a scope can only be accessed from within that scope.

Types of Scopes in JavaScript

- **Global Scope**: Variables declared outside any function reside in the global scope. They are accessible from anywhere in your program (use with caution to avoid naming conflicts).
- **Function Scope**: Variables declared within a function are only accessible from within that function.
- **Block Scope (ES6):** Variables declared using let or const within curly braces {} are only accessible within that block.

Global Scope Example

- var globalVar = "I'm a global variable";
- function showGlobalVar() {

  console.log(globalVar); // Accessible here

  }

  showGlobalVar();

globalVar = "Global variable modified";

console.log(globalVar); // Accessible and modified here

Function Scope Example

```
function myFunction() {
    var functionVar = "I'm a function variable";
    console.log(functionVar); // Accessible here
}
myFunction();
console.log(functionVar); // Error: functionVar is not defined
function outerFunction() {
    function innerFunction() {
        console.log("I'm a function scoped function");
    }
    innerFunction(); // Accessible here
}
outerFunction();
innerFunction(); // Error: innerFunction is not defined
```

Block Scope Example

```
{
    let blockVar = "I'm a block variable";
   console.log(blockVar); // Accessible here
}
// console.log(blockVar); // Error: blockVar is not defined
for (let i = 0; i < 3; i++) {
   // Block-scoped variable inside for loop
   console.log(i); // Accessible here
}
// console.log(i); // Error: i is not defined
if (true) {
   // Block-scoped variable inside if statement
   const blockConst = "I'm a block constant";
   console.log(blockConst); // Accessible here
}
// console.log(blockConst); // Error: blockConst is not defined
```

Combining All Scope Example

```
var globalVar = "Global";
function exampleFunction() {
    var functionVar = "Function";
    console.log(globalVar); // Accessible: Global
    console.log(functionVar); // Accessible: Function
    if (true) {
      const blockConst = "BlockConst";
      console.log(blockConst); // Accessible: BlockConst
    }
    // console.log(blockVar); // Error: blockVar is not defined
    // console.log(blockConst); // Error: blockConst is not defined
}   exampleFunction();
```

Understanding Hoisting

- Hoisting is a JavaScript behavior where variable and function declarations are treated differently than assignments.
- During code execution, the JavaScript engine hoists all declarations (not assignments) to the top of their scope.
- This creates an illusion that variables

  can be accessed before

  they are declared.

Hoisting with var

- var declarations are hoisted to the top of their scope.
- You can access var variables before their declaration, but their value will be undefined.

Ex :

```
console.log(message); // undefined
var message = "Hello, World!";
```

Hoisting with Let and Const

- Variables declared with let and const are also hoisted, but unlike var, they are not initialized with undefined. Accessing them before their declaration results in a ReferenceError.

Ex :

```
console.log(hoistedLet); // Error: Cannot access 'hoistedLet' before initialization
let hoistedLet = "I am not hoisted!";
console.log(hoistedLet); // Output: "I am not hoisted!“

console.log(hoistedConst); // Error: Cannot access 'hoistedConst' before initialization
const hoistedConst = "I am not hoisted either!";
console.log(hoistedConst); // Output: "I am not hoisted either!"
```

Hoisting with Functions

- Function declarations are hoisted completely, meaning you can call a function before its declaration.

Ex :

```
hoistedFunction(); // Output: "I am a hoisted function!"
function hoistedFunction() {
    console.log("I am a hoisted function!");
}
--------------------------------------------------------------------------------
-
console.log(notHoistedFunction); // Output: undefined (the variable is hoisted but not the function)
var notHoistedFunction = function() {
    console.log("I am not hoisted!");
};
// notHoistedFunction(); // Error: notHoistedFunction is not a function
notHoistedFunction(); // Output: "I am not hoisted!"
```

Conclusion

Control Variable Accessibility: Scopes define where variables and functions are visible, preventing conflicts and promoting code organization.

Predict Code Behavior: Understanding hoisting clarifies how variable and function declarations are treated, leading to more predictable code execution.

Write Maintainable JavaScript: By mastering scopes and hoisting, you write cleaner, more maintainable JavaScript that's easier to understand and debug.