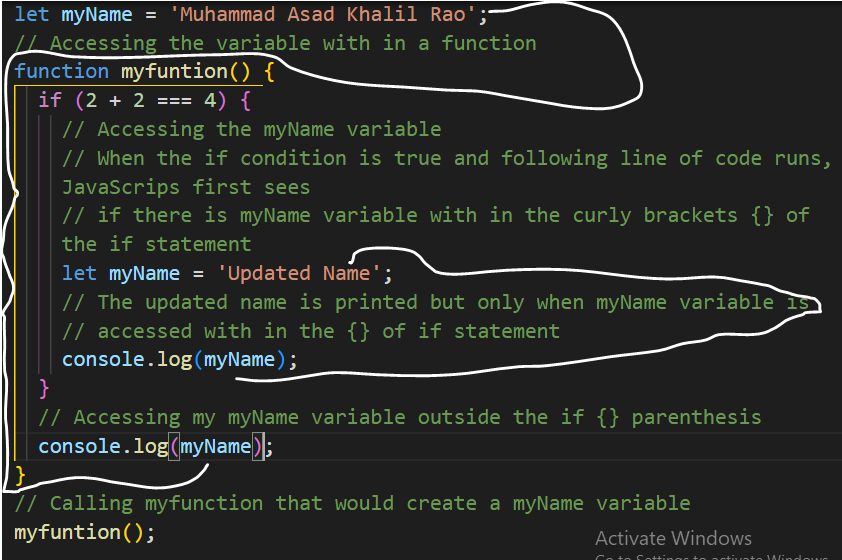
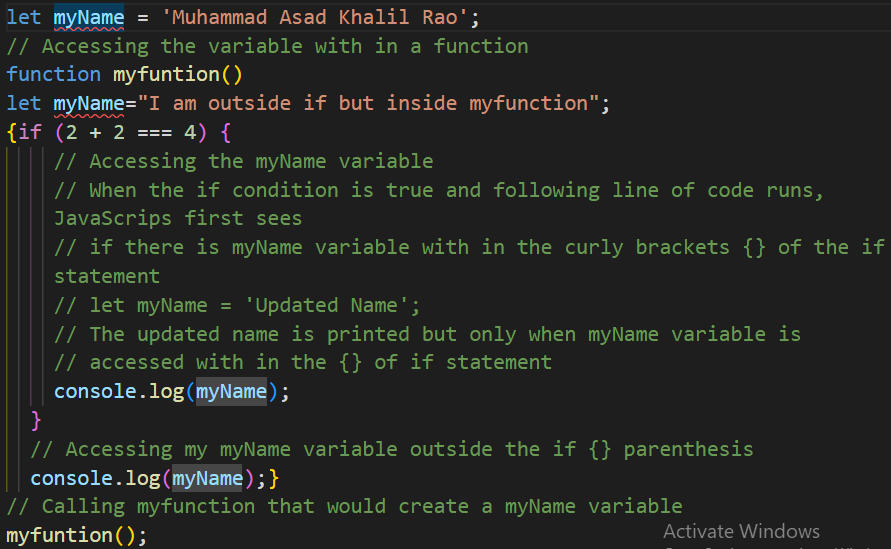


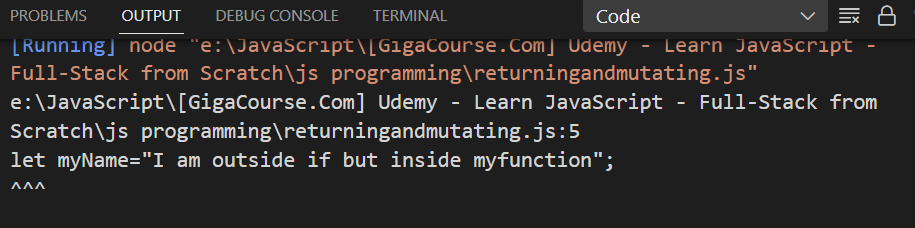
variable has global scope as it is not inside any {}

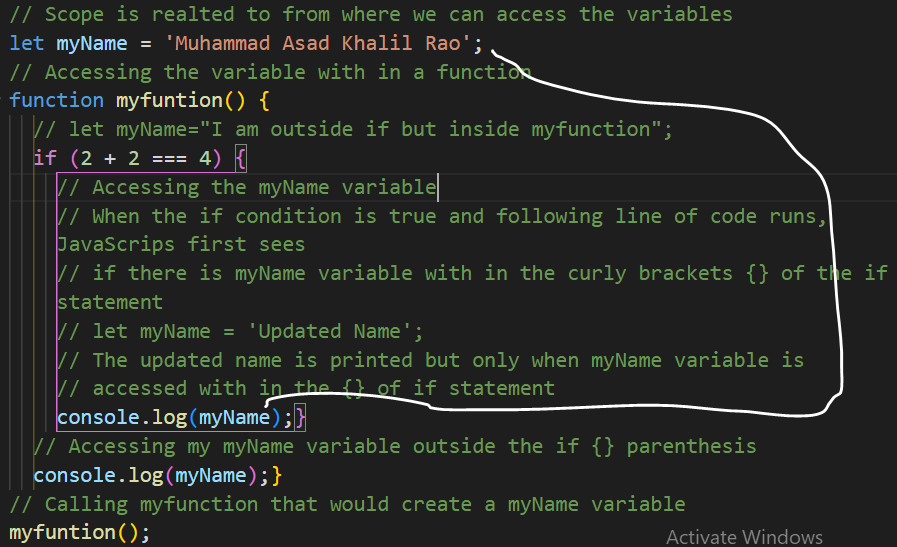




If the JavaScript sees that there is no myName variable with in the {} parenthesis of the if (current local scope). The JavaScript will look one level outside like outside the {} of if , with in the body of the myfunction.







JavaScript will look from if then one level up, myfunction and then outside the myfunction(global scope of the variable).

When a line of code runs looking for a variable, JavaScript is first going to look for that variable as close to home as possible with in the current scope, variable is not found then JavaScript is going to keep looking for the variable one level up through the scope chain until desired variable is found

let myName = 'Inside the global scope';

// Accessing the variable with in a function

function myfuntion() {

  // below myName is completely independent variable, not the updated or

  // overwritten original variable

  let myName = 'Inside myfunction';

  if (2 + 2 === 4) {

    // Accessing the myName variable

    // When the if condition is true and following line of code runs, JavaScrips first sees

    // if there is myName variable with in the curly brackets {} of the if statement

    // below myName is completely independent variable, not the updated or

    // overwritten original variable

    let myName = 'Inside if Statement';

    // The updated name is printed but only when myName variable is

    // accessed with in the {} of if statement

    console.log(myName);

  }

  // Accessing my myName variable outside the if {} parenthesis

  console.log(myName);

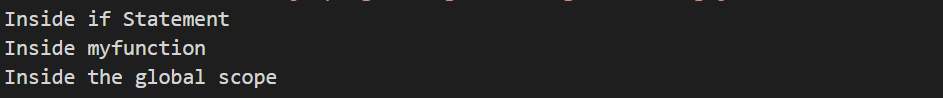
}

// Calling myfunction that would create a myName variable

myfuntion();

console.log(myName);

Hence, we have different values of the different variable of same label.



Value of the Variable myName inside if statement is 

Value of the Variable myName inside myfunction is 

Value of the Variable myName outside myfunction is 

Let statement creates a new variable within current local scope({}). It does not matter if that same label or name has been used out of scope.

Hence, we have three different variables of three different values but of same label (name).

// Scope is realted to from where we can access the variables

// is scoped globally

// let keyword creates an independent variable and new variable

let myName = 'Inside the global scope';

// Accessing the variable with in a function

function myfuntion() {

  // below let key word creates an independent and new variable that is scope to {} of the mfunction original variable is  not the updated or

  // overwritten

  let myName = 'Inside myfunction';

  if (2 + 2 === 4) {

    // Accessing the myName variable

    // When the if condition is true and following line of code runs, JavaScrips first sees

    // if there is myName variable with in the curly brackets {} of the if statement

    // let keyword creates an independent new variable that is scoped to {} of if statement, original variable is not the updated or overwritten

    let myName = 'Inside if Statement';

    // The updated name is printed but only when myName variable is

    // accessed with in the {} of if statement

    console.log(myName);

  }

  // Accessing my myName variable outside the if {} parenthesis

  console.log(myName);

} // Calling myfunction that would create a myName variable

myfuntion();

console.log(myName);

**Mutating Global Variable:**

Mutating global scoped variable inside the if condition.

**Key Take Away**

Inside code can reach outside code for variables , but outside code cannot reach inside side for variables.

**Code:**

// let keyword creates an independent variable and new variable

// myName is a global scoped variable

let myName = 'Inside the global scope';

function myfuntion() {

  // Mutating global scoped myName variabel inside the myfunction:

  // JavaScripe will see the myName variable and see whether myName variable exists inside myfuntion(current local scope) ,if myName variable does not then it will look one level up inside global scope,in the global scope myName exists hence the myName variable will be updated inside myfunction

  // Accessing global scoped myName variabel and setting its value to a string.

  myName = 'Inside myfunction';

  if (2 + 2 === 4) {

    // Mutating  global scoped myName variable inside the if condition:

    // JavaScripe will see the myName variable and see whether myName variable already exists inside if{}(current local scope) ,if myName variable does not then it will look one level up inside myfunction ,if it does not exits inside the myfunction then JavaScript will check one level up inside the global scope , in the global scope myName exists hence the myName variable will be updated inside if {}.

    // Accessing global scoped myName variabel and setting its value to a string.

    myName = 'Inside if Statement';

    console.log(myName);

  }

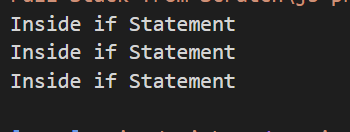
  // Accessing my myName variable outside the if {} parenthesis

  console.log(myName);

} // Calling myfunction that would create a myName variable

myfuntion();

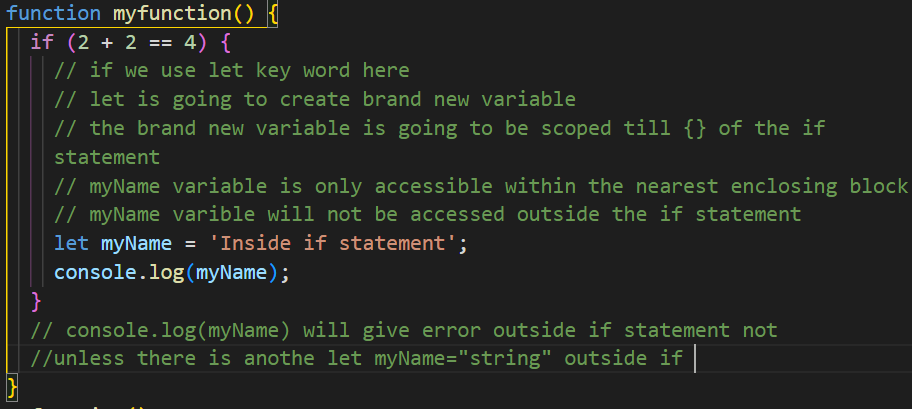
**Output:**



**In JavaScript variables can also be created using var**

**Difference between var and let**

|  |  |
| --- | --- |
| let | var |
| let uses block scope {} | **Var uses function scope** |
| if (2 + 2 == 4) {      // if we use let key word here      // let is going to create brand new variable      // the brand new variable is going to be scoped till {} of the if    statement      // myName variable is only accessible within the nearest enclosing block      // myName varible will not be accessed outside the if statement      let myName = 'Inside if statement';      console.log(myName);    }  Let is going to look for the nearest enclosing block.  If statement uses block (curly brackets of if statement {})  Curly brackets of if statement, while loop, for loop, body of a function is a block. |  |

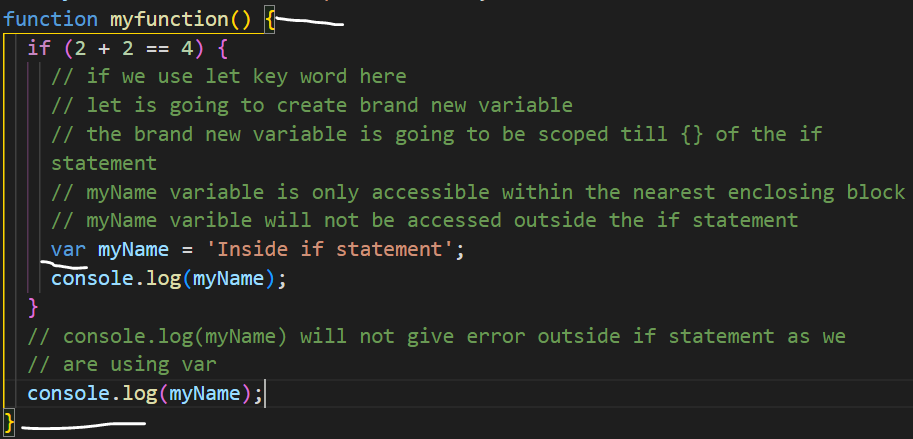


Var keyword uses function scope instead of block scope, var only consider the curly brackets{} of a function.

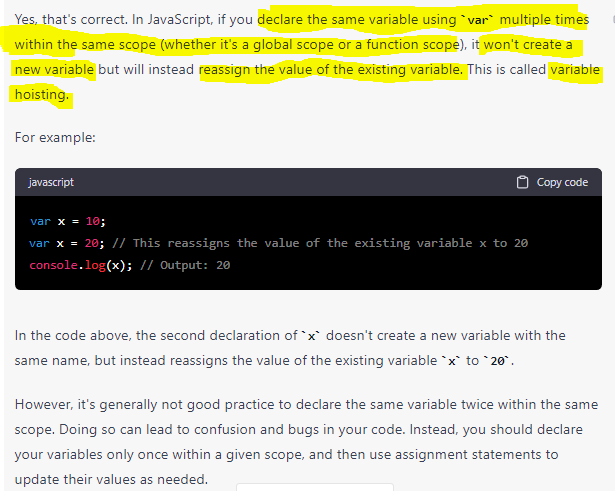
Hence can be accessible outside the if {} curly brackets, but inside {} of the function.

Var uses function scope (curly brackets of the body of the function)instead of block scope.

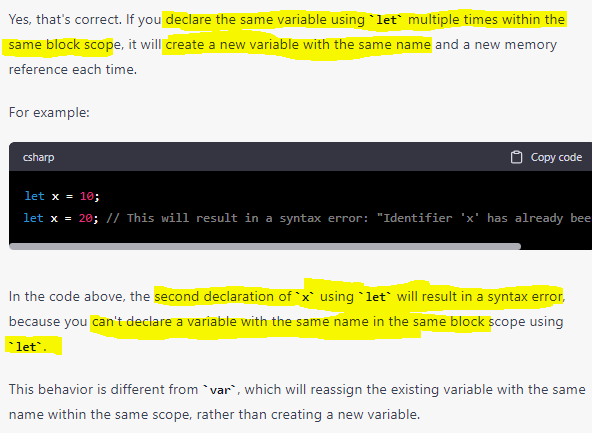
Var variable is accessible anywhere inside enclosing function myfunction

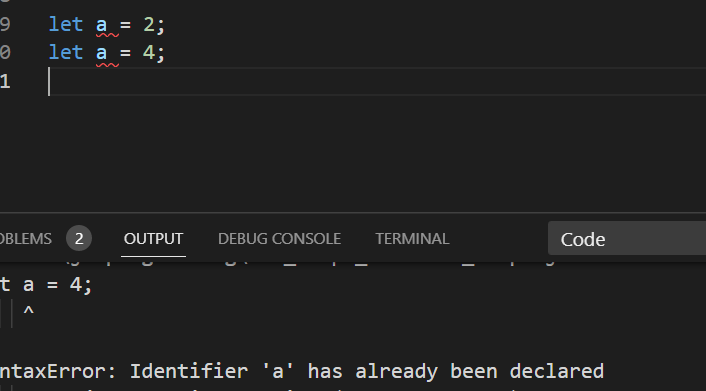


**Variable Hoisting**



Declaring same variable more than once in same block scope





**Main Difference between var and let**

