**Hoisting in JavaScript**

There’s another layer of how scope works with variable declarations that’s important to understand to avoid inserting bugs in your code.

Hoisting, which is done during the compilation of the program:

* Finds all the variable and function declarations in the code
* ‘Raises’ the variable names and function declarations to the top of their scope before code execution.
* Immediately initializes function declarations
* Postpones handling *initializations,* or assignment of values, until the code is executed.

However: There are subtle hoisting differences between **function declarations, function expressions, and variable declarations**, which are important to understand to avoid errors

1. **Function declarations**

As mentioned above, if we declare a function using the function keyword, then it is possible to call the function before the declaration. The function definition, including the function body, is hoisted to the top of the scope.

Hoisting also occurs for nested functions, so nested functions are raised to the top of the enclosing function scope.

console.log(add(1,2));   
// prints 3  
function add(a,b) { return a + b }

1. **Function expressions** 🡪 obey hoisting rules for variable declarations
   1. var hoisting

var variables are hoisted to the top of the function blocks, but not other types of blocks

🡪 var variables are initialized as undefined when hoisted.

* 1. let and const hoisting

let and const variables are hoisted to the top of their parent block for scope, so any type of block or function can be the parent scope for these variables.

While the names for the let and const variables are hoisted, they are not initialized. Therefore, if used, they will throw ReferenceError.

This rule also applies for function expressions, especially for arrow functions.

Text

Description automatically generated