**JavaScript Hoisting**

**Hoisting** is a phenomenon in JavaScript by which you can access variables and functions even before you initialized them.

console.log(x);

getName();

var x=7;

function getName(){

    console.log("Namaste guru");

}

In the above example we are trying to access x and function getName even-before we initialized them.

If we look at the output x would return undefined where as function would return namaste JavaScript.

Reason: As we know there is execution context in JavaScript which has memory block and code block. For variable x initially a memory space is created as undefined. In case of function all the lines inside the block are stored in memory. Hence function is able to print “Namaste guru” and x is returning undefined.

Graphical user interface, text, application

Description automatically generated

**Let and Const**

**Syntax Error Vs Reference Error Vs Type Error**

Let and const are hoisted but are hoisted in a different way.

They stay in temporal dead zone for the time being

console.log(b);

let a=10;

var b=100;

The output would return a special space holder=undefined. This is nothing but hoisting

Whereas if you try to console.log(a). We get Uncaught Reference Error: Cannot access 'a' before initialization.

But if let is also hoisted we should see output as undefined, but why are we getting Reference Error.

Answer: In case of var memory is allocated and attached to global object(window) i.e., they are in global scope.

In case of let variable memory is allocated but it is in a separate scope called script, it is not attached to global object, it cannot be accessed unless the variable is initialized.

Temporal Dead Zone: It is the time between variable is hoisted and variable is initialized with some value.

Reference Error:

Case 1)-When the variables are in temporal dead zone you cannot access them. It can be accessed only once the value is initialized.

Case 2)- If you try access a variable which is not defined anywhere in the program

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Syntax error:** if you try initializing let variable with same name again to a different value syntax error is thrown and no code is executed.

Graphical user interface, text, application, email

Description automatically generated

Here ideally at least console.log(b) should output undefined. Because of syntax error no code is run.

In case of **let** you can declare a variable and initialize it later.

In case of **const** you cannot declare a variable and initialize it later. Const is more strict

Graphical user interface, text, application, email

Description automatically generated

**Type Error:** if you try initializing **const** variable with same name again to a different value type error is thrown and no code is executed.

Graphical user interface, text, application, email

Description automatically generated

**Conclusion**

It is recommended to use const most of the time, whenever you don’t want change variable value later in the code, it is stricter.

If not const use let because it has temporal dead zone and you wont run into undefined error.

In day-to-day coding don’t use var.

Always put your declarations and initializations on the top of the code. This leads to temporal dead zone time=0.