**What is Shadowing?**

**A variable is declared in a certain scope having** the same name defined on its outer scope and when we call the variable from the inner scope, the value assigned to the variable in the inner scope is the value that will be stored in the variable in the memory space.

**Using Var**

function func() {

    var a = 'BLAH';

    if (true) {

        var a = 'BLAH BLAH';

        console.log(a);

    }

    console.log(a);

}

func();

**Output:**

BLAH BLAH

BLAH BLAH

**Using Let**

function func() {

    let a = 'BLAH';

    if (true) {

        let a = 'BLAH BLAH';

        console.log(a);

    }

    console.log(a);

}

func();

**Output:**

BLAH BLAH

BLAh

**Using const**

function func() {

    const a = 'BLAH';

    if (true) {

        const a = 'BLAH BLAH';

        console.log(a);

    }

    console.log(a);

}

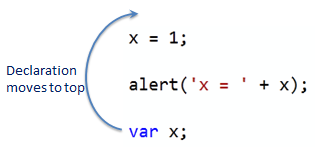
func();

**Output:**

BLAH BLAH

BLAh

**Hoisting**



*Image shown is for variable hoisting, same will be followed for function hoisting as well.*

Hoisting is a Js mechanism where variables and function declarations are moved to the top of their scope before code executes. This part is done when the program is being compiled (before the execution part).

function func() {

    a = 'BLAH';

    if (true) {

        var a = 'BLAH BLAH';  // New value assigned

        console.log(a);

    }

console.log(a);

}

func();

**Output:**

BLAH BLAH

BLAH BLAH

What is JavaScript Temporal Dead Zone

The **Temporal Dead Zone** is a behavior in JavaScript that occurs when declaring a variable with the let and const keywords, but not with var. A shorthand you’ll often hear to describe it is that “Let’s don’t hoist,” but this is not technically true.

In JavaScript, variable declarations (but not assignments) are [hoisted to the top of the scope](https://wsvincent.com/javascript-hoisting/). The code below works as expected:

But if you reverse the order and try to run console.log on a variable before creating it?

**function** myFunc(){

console.log(greeting);

**var** greeting **=** "Hello World!";

} myFunc();*// undfined*

The output will be undefined because the **variable declaration** is [hoisted](https://wsvincent.com/javascript-hoisting/) to the top of the scope. So in the eyes of the JavaScript interpreter, the code really looks as follows:

**function** myFunc(){

**var** greeting;

console.log(greeting);

greeting **=** 'Hello World!'

} *// nill*

The JavaScript interpreter works in a two-step process:

* **compile time**: run through all code looking for variable/function declarations
* **runtime**: execute the code including assignments and function invocations

Therefore on the first line, greeting is defined but has no assignment. JavaScript automatically provides the value undefined to defined variables without any variable. On line 2 the result will be undefined because the assignment does not occur until line three.

But if you use either the let or const keywords to declare a variable, this same code will throw an error:

**function** myFunc() {

console.log(greeting);

**let** greeting **=** 'Hello World!';

}

myFunc(); *// ReferenceError: greeting is not defined*

**function** myFunc() {

console.log(greeting);

**const** greeting **=** 'Hello World!';

}

myFunc(); *// ReferenceError: greeting is not defined*

This is the **Temporal Dead Zone** where we’re trying to access a variable that has not yet been initialized (it has been declared and therefore exists, but has no value, not even undefined). It’s common to hear the phrase, “let/const don’t hoist” as a shorthand to remember this behavior. But technically something else is going on.

When we use the var keyword, two things actually happen:

1. at **compile time**, the variable is added to the enclosing lexical scope
2. at **runtime**, when the scope is entered any variables added to the lexical environment are initialized to the undefined value so they are available to use in the scope

The let and const keywords do step 1–so technically they do hoist–but not step 2, the assignment to undefined. Therefore it’s more accurate to say that let/const **do hoist but don’t get initialized.**

The end result is the same. Just remember that when using let/const you should *always move variable declarations to the top of your scope* to avoid the **temporal dead zone**