VAR, LET, CONST

VAR

The var is the oldest keyword to declare a variable in JavaScript. It has the Global scoped or function scoped which means variables defined outside the function can be accessed globally, and variables defined inside a particular function can be accessed within the function.

Index.js

```
var a = 10
function f() {
    var b = 20
    console.log(a, b)
}
f();
console.log(a);
```

LET

The let keyword is an improved version of the var keyword. It is introduced in the ES6 or EcmaScript 2015. These variables has the block scope. It can’t be accessible outside the particular code block ({block}).

Index.js

```
let a = 10;
function f() {
  if (true) {
    let b = 9
    // It prints 9
    console.log(b);
  }
  // It gives error as it
  // defined in if block
  console.log(b);
}
f()
// It prints 10
console.log(a)
```

CONST

ES2015 (ES6) introduced the const keyword to define a new variable. Variables declared using the JavaScript const keyword cannot be re-assigned.

- Cannot be reassigned.
- It has Block Scope
- It can be assigned to the variable on the declaration line.
- It’s a Primitive value.
- The property of a const object can be changed but it cannot be changed to a reference to the new object
- The values inside the const array can be changed, it can add new items to const arrays but it cannot reference a new array.
- Re-declaring of a const variable inside different block scopes is allowed.
- Cannot be Hoisted.
- Creates only read-only references to value.

PROBLEMS WITH VAR

a. Function Scope
b. Redeclaration
c. Access before Initialisation.

A. FUNCTION SCOPE

The scope is nothing but boundaries represented by {}. There are two kinds of blocks we usually deal with in javascript. Function level and Normal blocks.

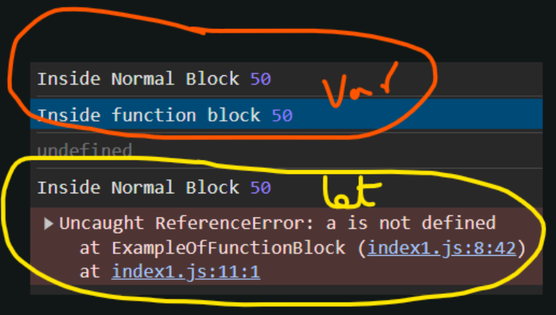
Function block {} which we write immediately after function declaration.

```
function ExampleOfFunctionBlock() {
  if (true) {
    var a = 50;
    console.log("Inside Normal Block", a);
    // Result of a will be 50 }

  }
    console.log("Inside function block", a);
    // Result of a will be 50
}
ExampleOfFunctionBlock();
```

```
function ExampleOfFunctionBlock() {
  if (true) {
    let a = 50;
    console.log("Inside Normal Block", a);
    // Result of a will be 50 }

  }
    console.log("Inside function block", a);
    // Result of a will be 50
}
ExampleOfFunctionBlock();
```



B. REDECLARATION

When we use var, we can write the same variable name multiple times within the scope as the new variable will override the previous one.

```
function reDeclareOfSameVariable() {
    var a = 70;
    if (true) {
        var a = 90;
    }
    console.log("The value of a is:", a);
}
reDeclareOfSameVariable();
// The value of a is: 90
```

```
function reDeclareOfSameVariable() {
    let a = 70;
    if (true) {
        let a = 90;
    }
    console.log("The value of a is:", a);
}
reDeclareOfSameVariable();
// The value of a is: 70
```

ACCESS BEFORE INITIALISATION

When we declare the variable through var keyword, We can access the variable before initialization as a result of undefined which as a programmer we don't want.

Index.js

```
console.log(x); // undefined
var x = 10;
```

Index.js

```
console.log(x); // ReferenceError:
Cannot access 'x' before initialization
let x = 10;
```