# Use of var in js:

The **var** statement declares a **variable**. Variables are containers for storing information. Creating a **variable** in **JavaScript** is called "declaring" a **variable**: **var** carName; After the declaration, the **variable** is empty (it has no value).**Examples of var:**

## **Function Scope**

Variables declared **Locally** (inside a function) have **Function Scope**.

Example :

// code here can NOT use carName  
function myFunction() {  
  var carName = "Volvo";  
  // code here CAN use carName  
}  
  
// code here can NOT use carName

## **JavaScript Block Scope**

Variables declared with the var keyword can not have **Block Scope**.

Variables declared inside a block **{}** can be accessed from outside the block

Example :

{  
  var x = 2;  
}  
// x CAN be used here

## **Redeclaring Variables**

Redeclaring a variable using the var keyword can impose problems.

Redeclaring a variable inside a block will also redeclare the variable outside the block:

Example :

var x = 10;  
// Here x is 10  
{  
  var x = 2;  
  // Here x is 2  
}  
// Here x is 2

## **Loop Scope**

Example:

Using var in a loop:

<script>

var i = 5;

for (var i = 0; i < 2; i++) {

document.write("hello"+" "+"hi"+" ");

// some statements}

document.write( i);

</script> //hello hi hello hi 2

1. **Example**

var hoisting

<script>

carName = "Volvo";

document.write(carName);//output volvo

var carName;

</script>

# Use of let in js:

**let** allows you to declare variables that are limited to a scope of a block statement, or expression on which it is used, unlike the var keyword, which defines a variable globally, or locally to an entire function regardless of block scope

1. **Example of let:**

let greeting = "say Hi";

let times = 4;

if (times > 3) {

let hello = "say Hello instead";

console.log(hello);//"say Hello instead"

}

console.log(hello) ; // hello is not defined

**Example**

1. let x = 1;

if (x === 1) {

let x = 2;

console.log(x);

// expected output: 2

}

console.log(x);

// expected output: 1

1. **Example**

Variables declared by **let** have their scope in the block for which they are defined, as well as in any contained sub-blocks. In this way, **let** works very much like **var**. The main difference is that the scope of a **var** variable is the entire enclosing function:

Variables declared with the let keyword can have Block Scope.

Variables declared inside a block **{}** can not be accessed from outside the block:

{  
  let x = 2;  
}  
// x can NOT be used here

1. **Example**

Using let in a loop:

let i = 5;  
for (let i = 0; i < 10; i++) {  
  // some statements  
}  
// Here i is 5

1. **Example**

Redeclaring a var variable with let, in the same scope, or in the same block, is not allowed:

var x = 2;       // Allowed  
let x = 3;       // Not allowed  
  
{  
  var x = 4;   // Allowed  
  let x = 5   // Not allowed  
}

////////////////////////////////////////////////////////////////

Using in function:

function nodeSimplified(){

let a =10;

console.log(a); // output 10

if(true){

let a=20;

console.log(a); // output 20

}

console.log(a); // output 10

}

# Use of Const in js:

# The const declaration creates a read-only reference to a value. It does not mean the value it holds is immutable, just that the variable identifier cannot be reassigned. For instance, in the case where the content is an object, this means the object's contents (e.g., its properties) can be altered

### **Example**

const PI = 3.141592653589793;  
PI = 3.14;      // This will give an error  
PI = PI + 10;   // This will also give an error

1. **Example**

## **Block Scope**

Declaring a variable with const is similar to let when it comes to **Block Scope**.

The x declared in the block, in this example, is not the same as the x declared outside the block:

var x = 10;  
// Here x is 10  
{  
  const x = 2;  
  // Here x is 2  
}  
// Here x is 10

1. **Example**

## **Assigned when Declared**

JavaScript const variables must be assigned a value when they are declared:

incorrect

const PI;  
PI = 3.14159265359;  
correct

const PI = 3.14159265359;

1. **Example**

## **Constant Objects can Change**

You can change the properties of a constant object:

// You can create a const object:  
const car = {type:"Fiat", model:"500", color:"white"};  
  
// You can change a property:  
car.color = "red";  
  
// You can add a property:  
car.owner = "Johnson";

1. **Example**

## **Constant Arrays can Change**

constant

<p>Declaring a constant array does NOT make the elements unchangeble:</p>

<script>

// Create an Array:

const cars = ["Saab", "Volvo", "BMW"];

// Change an element:

cars[0] = "Toyota";

// Add an element:

cars.push("Audi");

// Display the Array:

document.write(cars);

</script>