# **JavaScript Var:**

## JavaScript variables are containers for storing data values. There are two types of variables in JavaScript: local variable and global variable. All JavaScript **variables** must be**identified** with **unique names**. These unique names are called **identifiers**. Creating a variable in JavaScript is called "declaring" a variable. Declare a JavaScript variable with the var keyword: var carName;

## 

## In this example, x, y, and z, are variables:

## var x = 5; var y = 6; var z = x + y;

* **Examples:**

## <html >

## <body>

## <script>

## var x = 10;

## var y = 20;

## var z=x+y;

## document.write(z);

## </script>

## </body>

## </html>

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

## Using var in a loop:

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

## Console.log (x);

## var x=5;

console.log (x);

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

# **JavaScript Let:**

## Let is used for variable declaration in JavaScript and let is block scoped. The let keyword is not fully supported in Internet Explorer 11 or earlier.

* **Examples:**

## let x = 1;

## if (x === 1) {

## let x = 2;

## console.log(x);

## // expected output: 2

## }

## console.log(x);

## // expected output: 1

## 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 **{}** cannot be accessed from outside the block:

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

## Using let in a loop:

## let i = 5; for (let i = 0; i < 10; i++) {    // some statement

## }

## Here i is 5

## 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 }

## console.log(x)

## let x=5;

## console.log(x);

## Output= ERROR.

# **JavaScript Const:**

## Variables defined with const behave like let variables. 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.

## **Examples:**

1. <html>

## <body>

## <h2>JavaScript const</h2>

## <p>You cannot change a primitive value.</p>

## <p id="demo"></p>

## <script>

## try {

## const PI = 3.141592653589793;

## PI = 3.14;

## }

## catch (err) {

## document.getElementById("demo").innerHTML = err;

## }

## </script>

## </body>

## </html>

## Constant Arrays can Change

## You can change the elements of a constant array:

## // You can create a constant array: const cars = ["Saab", "Volvo", "BMW"]; // You can change an element: cars[0] = "Toyota"; // You can add an element: cars.push("Audi");

## 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";

## 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;

## 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

## 