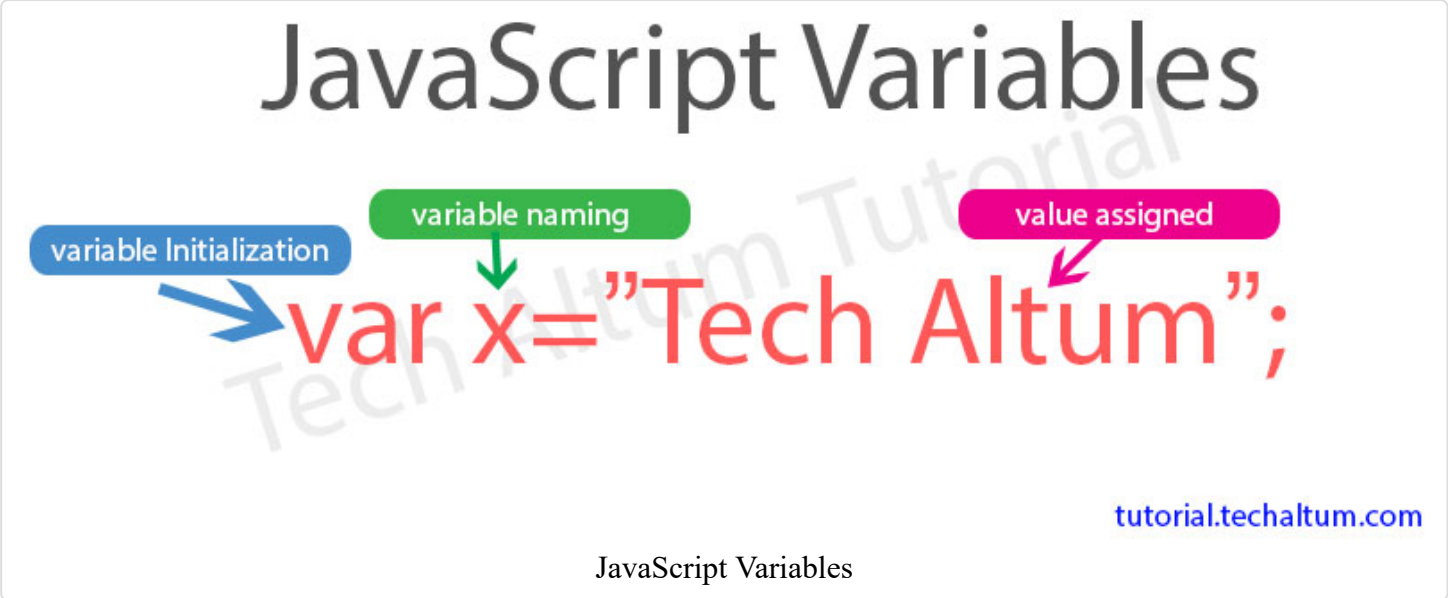
Variables in JavaScript

A **variable** or **var** is storage area used to **store a value** in memory. Consider variable as a container to store data. To create a **variable in javascript**, **var keyword** is used. *var* is a **reserved keyword** in javascript, used to create **variables**. We can also create a variable without using *var* in javascript, but it is not recommended.

JavaScript variables are loosely typed. This is also knows as *dynamic typing*. Means a variable x (`var x`) can store any type of data, like string, number, function, boolean, undefined, null, array or object. Usually in all scripting languages, variables are loosely typed. But in the case of Java or C++ and they are Strictly typed.



JavaScript Variables

Variables are always initialized first with `var` keyword.
For exp `var x` means a variable named x is initialized.
To assign a value in variable, = or *assignment operator* is used, followed by value.

How to declare variables in JavaScript

```
<script>

 var x="Tech Altum";    // value of x is "Tech Altum";
 var y='Tech Altum';    // value of y is also "Tech Altum";
 var z=9;               // value of z is 9 numeric.
 var a=2, b=3, c=5;     // multiple variables in single line
 var u;                 // u is undefined
```

```
</script>
```

Change value of variable

To change value of a variable, assign a new value to the same variable.

```
var x="hello";
console.log(x);         //  "hello"

x="hola";
console.log(x);         // "hola"
```

var keyword is used to declare a variable. To change value of variable, there is no need to use var keyword again, only variable name is enough.

Variable naming convention in JavaScript

To declare a variable in javascript, there are some rules. Here are some rules used to declare variables names in javascript.

variables names cannot include **javascript reserved keywords**, like var, for, in, while, true, false, null, class, let, const, function etc.
variables cannot starts with numbers and hyphen, use alphabets, underscore(_) or dollar ($).
variables can have strings followed by numbers, like x1 and x2 are allowed but 1x, 2x is not allowed.
For separation in variable names, use underscore (_) or camel casing, do not use hyphen (-) separation and white space, i.e, user_name is valid, username is also valid, but user-name is invalid.
variables names are case sensitive, i.e, x1 and X1 are different.

Strict Mode

ECMAScript 5 includes **strict mode** in javascript. **"use strict"** string is used to follow **strict mode**. In strict mode, var is compulsory, but in sloopy mode, we can also create variables without var keyword.

```
<script>
 "use strict";

 var x1="tech altum";                                    // allowed in strict mode
 x2="javascript";                                      // not allowed in strict mode

 for( var i=1;i<=10;i++){}                                                  // allowed
 for( i=1;i<=10;i++){}                                // not allowed, i is not declared

 var x=10;                                                                  // allowed
 var y=010;                                          // not allowed, it's octal notation
</script>
```

Sloppy Mode

By default javascript is written in **sloppy mode**. **Sloppy mode** allow us to create a variable with or without var keyword.

```
<script>
    var x1="tech altum";                                                    // allowed
    x2="javascript";                                                  // also allowed
</script>
```

Variable Hoisting

JavaScript Hoisting is the process to move all variables and functions to the top of code block or scope before execution. **Variable Declaration is hoisted**, but **variable assignment is not hoisted**. This means, a variable x

declared at any point can have `undefined` value before assignment, but after assignment, variable value can be string or number defined.

```
<script>
    console.log(x);                                    // return undefined
    var x="hello js";
    console.log(x);                                   // return "hello js"
</script>
```

It is recommended to declare variables and assign values at the top to avoid **variable hoisting** issue.

Local Vs Global Variables

A variable in javascript can be **local** or **global variable** based on its scope. Variables declared outside function are **global variables** and variable declared inside function are **local variables**.

Global Variables

Variables declared outside function are global variable. **Global variables have global scope**. This means, a variable in `<script>` tag is by default **global**. **Global Variable** once declared can be used anywhere, even inside a function.

Global Variables Example

```
    var x="hello js";        // global variable
    var y=88;                // another global variable

    console.log(x);          // returns "hello string"
    console.log(y);          // returns 88

    function myFunction(){
        console.log(x)           // returns "hello string"
        console.log(y);          // returns 88
    }
```

Local Variables

Variables declared inside functions are local variables. **Local Variables** have **local scope**. This means they are accessible only within their function block, not outside.

Local Variables Example

```
function myFunction(){

    var x="local variable";
    var y="local variable";

    console.log(x);    // returns "local variable"
    console.log(y);    // returns "local variable"
}

    myFunction()        // call the function

    console.log(x);    // x is not defined
    console.log(y);    // y is not defined
```

JavaScript global Variables

```
function myFunction(){

    var x="local variable";
    y="global variable";

    console.log(x);    // returns "local variable"
    console.log(y);    // returns "global variable"
}

    myFunction()        // call the function

    console.log(x);    // x is not defined
    console.log(y);    // y is defined
```

let and Const block scope

let and **const** were introduced in ECMA 6. **let** is used to declare temporary values with block scope, inside {} . Whereas **const** is used to declare permanent values, which can't be change. Like value of pi, e, G and other constants.

Scope of var is global or local.

JavaScript let

JavaScript let is used to declare variable in **block scope**. The value of let is only accessible inside block. Also let naming is not hoisted like var. So let can be used only after declaration.

```
<script>
{
    let x="user";        // to store values in block scope
}
</script>
```

```
<script>
var i=2;                    // global scope

if(i==2){
    let i=3;                // block scope
    console.log(i);        // 3
}

console.log(i);            // 2
</script>
```

JavaScript const

JavaScript const is used to declared immutable values, i.e fixed values which are not changeable. Const can't be undefined.

```
<script>
const pi=Math.PI;    // pi is a constant, and can't be changed
</script>
```

```
<script>
const user="avi";

user="xyz";  //Assignment to constant variable.

</script>
```

var vs let vs const

var is **function scoped**, but let and const are **block scoped**. *let* and **const** are replacement of **var** in JS ES6 and onwards. Here is table of comparison of var, let and const.

Property	**var**	**let**	**const**
hoisting	declaration	temporal dead zone	temporal dead zone
Scope	Function	block	block
Create Global property	yes	no	no

1. Although it is not compulsory, but its recommended to use strict mode.
2. **let** and **const** are supported in IE 11 and above, edge, firefox 36 and above, chrome 21 and above, safari 5.1 and above.
3. For IE 10 and below, use var.

← Javascript Tutorial