JavaScript Variables

Variables are Containers for Storing Data. JavaScript Variables can be declared in 4 ways:

- Automatically
- Using `var`
- Using `let`
- Using `const`

In this first example, `x`, `y`, and `z` are undeclared variables. They are automatically declared when first used:

Example

```
x = 5;
y = 6;
z = x + y;
```

Note: It is considered good programming practice to always declare variables before use.

From the examples you can guess:

- x stores the value 5
- y stores the value 6
- z stores the value 11

Example using var

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

Note: The `var` keyword was used in all JavaScript code from 1995 to 2015.
The `let` and `const` keywords were added to JavaScript in 2015.
The `var` keyword should only be used in code written for older browsers.

Example using let

```
let x = 5;
let y = 6;
let z = x + y;
```

Example using const

```
const x = 5;
const y = 6;
const z = x + y;
```

Mixed Example

```
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
```

The two variables `price1` and `price2` are declared with the `const` keyword.

These are constant values and cannot be changed. The variable `total` is declared with the `let` keyword. The value `total` can be changed.

When to Use var, let, or const?

1. Always declare variables
2. Always use `const` if the value should not be changed
3. Always use `const` if the type should not be changed (Arrays and Objects)
4. Only use `let` if you can't use `const`
5. Only use `var` if you MUST support old browsers.

Just Like Algebra

Just like in algebra, variables hold values:

```
let x = 5;
let y = 6;
```

Just like in algebra, variables are used in expressions:

```
let z = x + y;
```

From the example above, you can guess that the total is calculated to be 11.

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**. These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter.
- Names can also begin with $ and _ (but we will not use it in this tutorial).
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

Note: JavaScript identifiers are case-sensitive.

The Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

```
x = x + 5
```

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

Note

The "equal to" operator is written like == in JavaScript.

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe". In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes. If you put a number in quotes, it will be treated as a text string.

Example

```
const pi = 3.14;
let person = "John Doe";
let answer = 'Yes I am!';
```

Declaring a JavaScript Variable

Creating a variable in JavaScript is called "declaring" a variable. You declare a JavaScript variable with the `var` or the `let` keyword:

```
var carName;
```

or:

```
let carName;
```

After the declaration, the variable has no value (technically it is `undefined`). To **assign** a value to the variable, use the equal sign:

```
carName = "Volvo";
```

You can also assign a value to the variable when you declare it:

```
let carName = "Volvo";
```

In the example below, we create a variable called `carName` and assign the value "Volvo" to it. Then we "output" the value inside an HTML paragraph with id="demo":

Example

```
<p id="demo"></p>
<script>
let carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
```

Note: It's a good programming practice to declare all variables at the beginning of a script.

One Statement, Many Variables

You can declare many variables in one statement. Start the statement with `let` and separate the variables by **comma**:

Example

```
let person = "John Doe", carName = "Volvo", price = 200;
```

A declaration can span multiple lines:

Example

```
let person = "John Doe",
carName = "Volvo",
price = 200;
```

Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value `undefined`. The variable carName will have the value `undefined` after the execution of this statement:

Example

```
let carName;
```

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable declared with `var`, it will not lose its value.

The variable `carName` will still have the value "Volvo" after the execution of these statements:

Example

```
var carName = "Volvo";
var carName;
```

Note: You cannot re-declare a variable declared with `let` or `const`. This will not work:

```
let carName = "Volvo";
let carName;
```

JavaScript Arithmetic

As with algebra, you can do arithmetic with JavaScript variables, using operators like `=` and `+`:

```
let x = 5 + 2 + 3;
```

You can also add strings, but strings will be concatenated:

Example

```
let x = "John" + " " + "Doe";
```

Also try this:

Example

```
let x = "5" + 2 + 3;
```

Note: If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

Now try this:

Example

```
let x = 2 + 3 + "5";
```

JavaScript Dollar Sign $

Since JavaScript treats a dollar sign as a letter, identifiers containing $ are valid variable names:

Example

```
let $ = "Hello World";
let $$$ = 2;
let $myMoney = 5;
```

Using the dollar sign is not very common in JavaScript, but professional programmers often use it as an alias for the main function in a JavaScript library.

JavaScript Underscore (_)

Since JavaScript treats underscore as a letter, identifiers containing _ are valid variable names:

Example

```
let _lastName = "Johnson";
let _x = 2;
let _100 = 5;
```

Using the underscore is not very common in JavaScript, but a convention among professional programmers is to use it as an alias for "private (hidden)" variables.

JavaScript Let

The `let` keyword was introduced in ES6 (2015).

- Variables declared with `let` have **Block Scope**
- Variables declared with `let` must be **Declared** before use
- Variables declared with `let` cannot be **Redeclared** in the same scope

Block Scope

Before ES6 (2015), JavaScript did not have **Block Scope**. JavaScript only had **Global Scope** and **Function Scope**. ES6 introduced the two new JavaScript keywords: `let` and `const`.

These two keywords provided **Block Scope** in JavaScript:

Example

Variables declared inside a { } block cannot be accessed from outside the block:

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

Global Scope

Variables declared with the `var` always have **Global Scope**. Variables declared with the `var` keyword can NOT have block scope:

Example

Variables declared with `var`inside a { } block can be accessed from outside the block:

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

Cannot be Redeclared

- Variables defined with `let` **cannot** be redeclared.
- You cannot accidentally redeclare a variable declared with `let`.

With `let` you **cannot** do this:

```
let x = "John Doe";

let x = 0;
```

Variables defined with `var` **can** be redeclared. With `var` you **can** do this:

```
var x = "John Doe";

var x = 0;
```

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
```

Redeclaring a variable using the `let` keyword can solve this problem.
Redeclaring a variable inside a block will not redeclare the variable outside the block:

Example

```
let x = 10;
// Here x is 10
{
let x = 2;
// Here x is 2
}
// Here x is 10
```

Difference Between var, let and const

	Scope	Redeclare	Reassign	Hoisted	Binds this
var	No	Yes	Yes	Yes	Yes
let	Yes	No	Yes	No	No
const	Yes	No	No	No	No

What is Good?

- `let` and `const` have **block scope**.
- `let` and `const` cannot be **redeclared**.
- `let` and `const` must be **declared** before use.
- `let` and `const` does **not bind** to `this`.
- `let` and `const` are **not hoisted**.

What is Not Good?

- `var` does not have to be declared.
- `var` is hoisted.
- `var` binds to this.

Redeclaring

Redeclaring a JavaScript variable with `var` is allowed anywhere in a program:

Example

```
var x = 2;
// Now x is 2

var x = 3;
// Now x is 3
```

With `let`, redeclaring a variable in the same block is NOT allowed:

Example

```
var x = 2;   // Allowed
let x = 3;   // Not allowed
```

```
{
let x = 2;   // Allowed
let x = 3;   // Not allowed
}

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

Redeclaring a variable with `let`, in another block, IS allowed:

Example

```
let x = 2;   // Allowed

{
let x = 3;   // Allowed
}

{
let x = 4;    // Allowed
}
```

Let Hoisting

Variables defined with `var` are **hoisted** to the top and can be initialized at any time. Meaning: You can use the variable before it is declared:

Example

This is OK:

```
carName = "Volvo";
var carName;
```

Variables defined with `let` are also hoisted to the top of the block, but not initialized. Meaning: Using a `let` variable before it is declared will result in a `ReferenceError`:

Example

```
carName = "Saab";
let carName = "Volvo";
```

JavaScript Const

- The `const` keyword was introduced in ES6 (2015)
- Variables defined with `const` cannot be **Redeclared**
- Variables defined with `const` cannot be **Reassigned**
- Variables defined with `const` have **Block Scope**

Cannot be Reassigned

A variable defined with the `const` keyword cannot be reassigned:

Example

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

Must be Assigned

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

Correct

```
const PI = 3.14159265359;
```

Incorrect

```
const PI;
PI = 3.14159265359;
```

When to use JavaScript const?

Always declare a variable with `const` when you know that the value should not be changed.

Use `const` when you declare:

- A new Array
- A new Object
- A new Function
- A new RegExp

Constant Objects and Arrays

The keyword `const` is a little misleading. It does not define a constant value. It defines a constant reference to a value.

Because of this you can NOT:

- Reassign a constant value
- Reassign a constant array
- Reassign a constant object

But you CAN:

- Change the elements of constant array
- Change the properties of constant object

Constant Arrays

You can change the elements of a constant array:

Example

```
// 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");
```

But you can NOT reassign the array:

Example

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

cars = ["Toyota", "Volvo", "Audi"];    // ERROR
```

Constant Objects

You can change the properties of a constant object:

Example

```
// 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";
```

But you can NOT reassign the object:

Example

```
const car = {type:"Fiat", model:"500", color:"white"};

car = {type:"Volvo", model:"EX60", color:"red"};    // ERROR
```

What is Good?

- `let` and `const` have **block scope**.
- `let` and `const` can not be **redeclared**.
- `let` and `const` must be **declared** before use.
- `let` and `const` does **not bind** to `this`.
- `let` and `const` are **not hoisted**.

What is Not Good?

- `var` does not have to be declared.
- `var` is hoisted.
- `var` binds to this.

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:

Example

```
const x = 10;
// Here x is 10

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

// Here x is 10
```

Redeclaring

Redeclaring a JavaScript `var` variable is allowed anywhere in a program:

Example

```
var x = 2;      // Allowed
var x = 3;      // Allowed
x = 4;          // Allowed
```

Redeclaring an existing `var` or `let` variable to `const`, in the same scope, is not allowed:

Example

```
var x = 2;      // Allowed
const x = 2;    // Not allowed

{
let x = 2;      // Allowed
const x = 2;    // Not allowed
}

{
const x = 2;    // Allowed
const x = 2;    // Not allowed
}
```

Reassigning an existing `const` variable, in the same scope, is not allowed:

Example

```
const x = 2;      // Allowed
x = 2;            // Not allowed
var x = 2;        // Not allowed
let x = 2;        // Not allowed
const x = 2;      // Not allowed

{
  const x = 2;    // Allowed
  x = 2;          // Not allowed
  var x = 2;      // Not allowed
  let x = 2;      // Not allowed
  const x = 2;    // Not allowed
}
```

Redeclaring a variable with `const`, in another scope, or in another block, is allowed:

Example

```
const x = 2;       // Allowed

{
  const x = 3;   // Allowed
}

{
  const x = 4;   // Allowed
}
```

Hoisting

Variables defined with `var` are **hoisted** to the top and can be initialized at any time. Meaning: You can use the variable before it is declared:

Example

This is OK:

```
carName = "Volvo";
var carName;
```

Variables defined with `const` are also hoisted to the top, but not initialized. Meaning: Using a `const` variable before it is declared will result in a `ReferenceError`:

Example

```
alert (carName);
const carName = "Volvo";
```