How the let, const, and var Keywords Work in JavaScript

freecodecamp.org/news/understanding-let-const-and-var-keywords

11 January 2022

TAPAS ADHIKARY

As a JavaScript beginner, you probably learned how to declare variables and assign values.

In the old, pre-ES6 era of JavaScript, developers used to declare variables using the keyword `var` or without any keywords. But times have changed!

With ES6 (EcmaScript 2015), the beginning of the modern era in JavaScript, the language got two new keywords to help us declare variables. These are `let` and `const`.

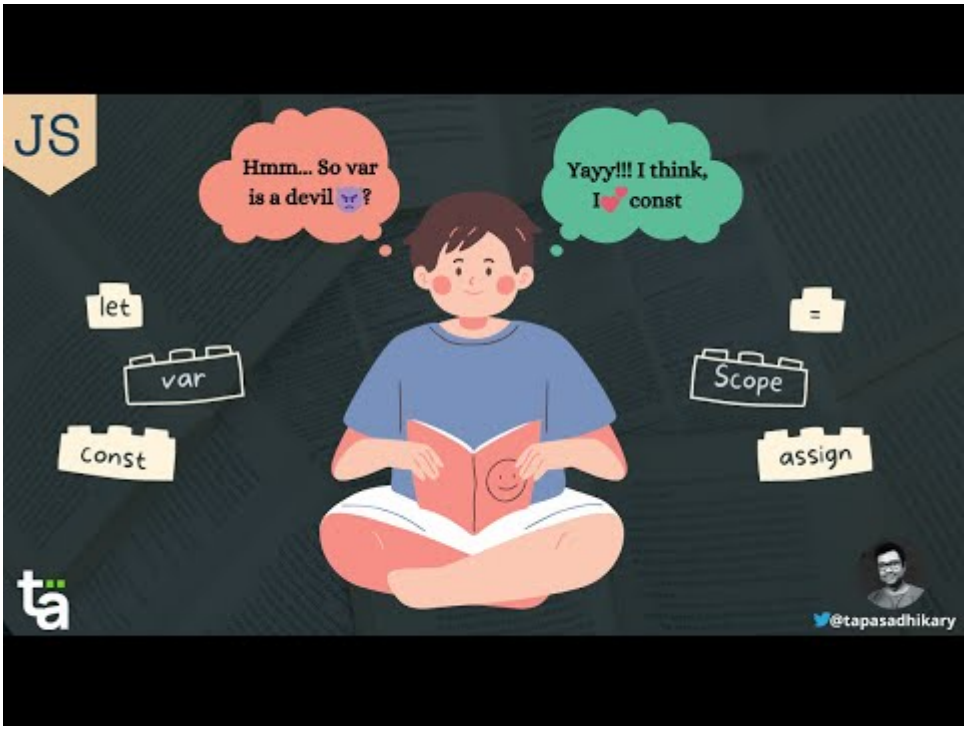
In this article, we will learn about all of these keywords (yes, including `var`) with examples, and we'll see when to use them, and when not to use them.

If you like to learn from video content as well, this article is also available as a YouTube video tutorial here: 🙂

With ES6 (EcmaScript 2015), the beginning of the modern era in JavaScript, the language got two new keywords to help us declare variables. These are `let` and `const`.

In this article, we will learn about all of these keywords (yes, including `var`) with examples, and we'll see when to use them, and when not to use them.

If you like to learn from video content as well, this article is also available as a YouTube video tutorial here: 🙂



Watch Video At: https://youtu.be/ehn_BcusA_c

Btw, this is a widely discussed topic. Then, why write about it again? Well, these keywords can be tough to learn because:

1. Many devs try using them interchangeably(especially `let` with the other two).
2. At times, you may get confused about the relationship of these keywords to a fundamental JavaScript concept called `Scope`.

So, this article aims to teach these keywords in the context of three essential concepts. I hope you enjoy reading it.

How to Declare Variables in JavaScript

In JavaScript, we can declare variables in three different ways like this:

```
// Without keywords. It is essentially the same as var
// and not allowed in 'strict' mode.
name = 'Jack';

// Using var
var price = 100;

// Using let
let isPermanent = false;

// Using const
const PUBLICATION = 'freeCodeCamp'
```

It is best when you understand var, let, and const with these three concepts:

- Scope
- Reassigning a new value
- When you access a variable before declaring it

These keywords differ in usage in respect to these concepts. Let's see how.

Variable Scope in JavaScript

In JavaScript, we use scope as a way to identify where and whether we can use a variable. The variables may exist within a block, inside a function, or outside a function and block.

So, what is a block? A block (that is, a code block) is a section of the code we define using a pair of curly brace s({...}). Something like this:

```
{
  let name = "alex";
}
```

On the other hand, a function is a bunch of code instructions you want to place logically together.

Usually, you define a function using the `function` keyword and a name. Just be aware that you can define a function without a name, which we call an `anonymous function`. But we won't discuss that in today's article for simplicity.

Here is a function with the name `test`.

```
function test() {
  let name = "alex";
}
```

Anything and everything outside of a block or a function we'll call `Global`. So, when we declare variables, they can exist within a block, inside a function, or outside of a block/function – that is, they have global scope.

There are mainly three types of scope:

- Block Scope
- Functional Scope
- Global Scope

The three keywords `var`, `let`, and `const` work around these scopes. So let's understand how things fit together.

How to Use JavaScript Variables in Block Scope

If you **do not want** a variable declared inside a `{ }` block to be accessed outside of the block, you need to declare them using the `let` or `const` keywords. Variables declared with the `var` keyword inside the `{ }` block **are** accessible outside of the block too. So, be careful.

Let's take an example:

```
{
    let f_name  = 'Alex';
    const ZIP = 500067;
    var age = 25;
}

console.log(f_name); // Uncaught ReferenceError: f_name is not defined
console.log(ZIP);  // Uncaught ReferenceError: ZIP is not defined
console.log(age);  // 25
```

As you see, the value of the age variable may get overridden unknowingly and eventually introduce a bug. So, the moral of the story is,

> Do not use the `var` keyword inside a block (block scope). Always use `let` and `const` instead.

How to Use JavaScript Variables in Functional Scope

A variable declared inside a function using these keywords is **not** accessible outside the function. That's the applied functional scope.

It is true irrespective of whether you use var, let, or const. Inside the function, they are pretty similar in managing a variable's scope.

Let's take an example again:

```
// f1() is a function

function f1() {
 let f_name = "Alex";
 const ZIP = 560089;
 var age = 25;
}

f1();

console.log(f_name); // Uncaught ReferenceError: f_name is not defined
console.log(ZIP);  // Uncaught ReferenceError: ZIP is not defined
console.log(age);  // Uncaught ReferenceError: age is not defined
```

As you see above, none of the variables are accessible outside of the function, not even `age` which is declared using `var`. So, the conclusion is,

> The variable declared with `var` inside a function is not accessible outside of it. The keyword `var` has function-scope.

How to Use JavaScript Variables in Global Scope

Variables declared outside of any functions and blocks are `global` and are said to have `Global Scope`. This means you can access them from any part of the current JavaScript program.

You can use `var`, `let`, and `const` to declare global variables. But you shouldn't do it too often.

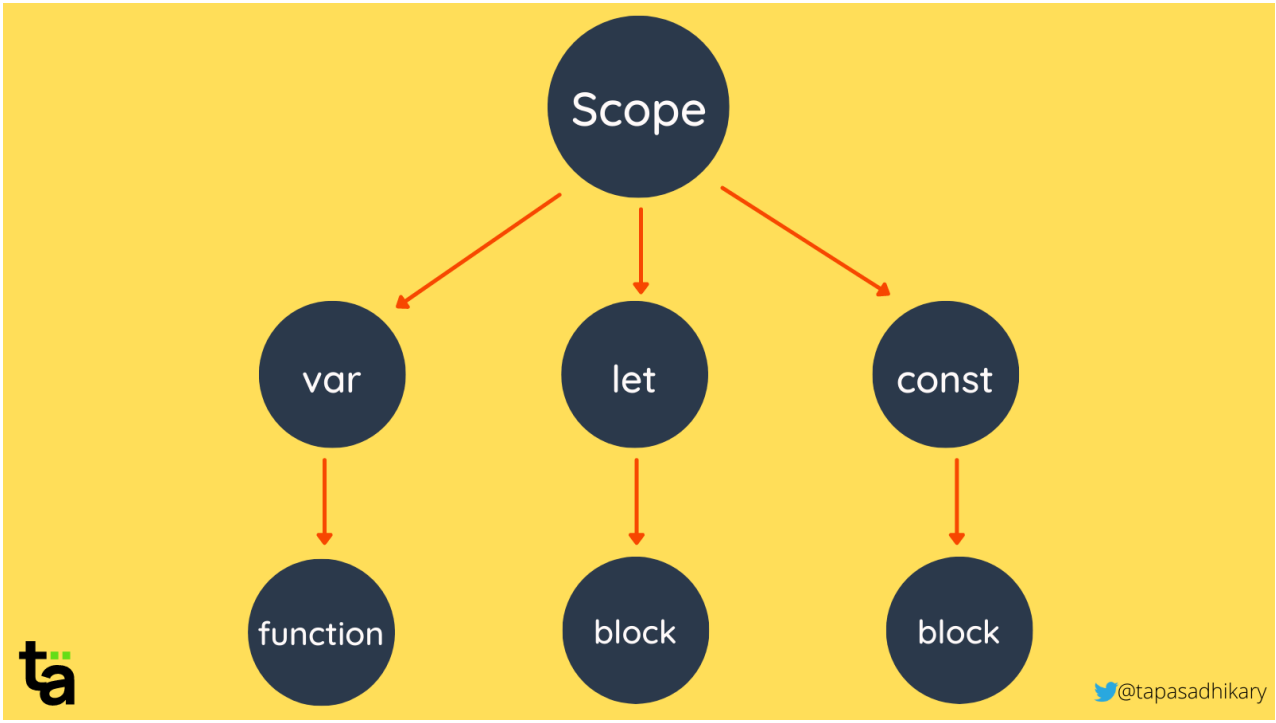
```
let f_name = "Alex";
 const ZIP = 560089;
 var age = 25;

// f1() is a function
function f1() {
  console.log(f_name); // Alex
  console.log(ZIP);  // 560089
  console.log(age);  // 25
}

f1();

console.log(f_name); // Alex
console.log(ZIP);  // 560089
console.log(age);  // 25
```

As you see, the variables are accessible everywhere.

So, to restrict the scope of a variable using the `var`, `let`, and `const` keywords, here's the order of accessibility in scope starting with the lowest:

- `var`: The functional scope level
- `let`: The block scope level
- `const`: The block scope level

The image below shows a mindmap of these three keywords with reference to different scopes.



Let's move on to the next concept to understand how these three keywords influence the code's behavior when we reassign a new value to a variable.

How to Reassign a New Value to a Variable in JavaScript

Once you've declared a variable with `var` or `let`, you **can** reassign a new value to the variable in your programming flow. It is possible if the variable is accessible to assign a value. But with `const`, you **can't** reassign a new value at all.

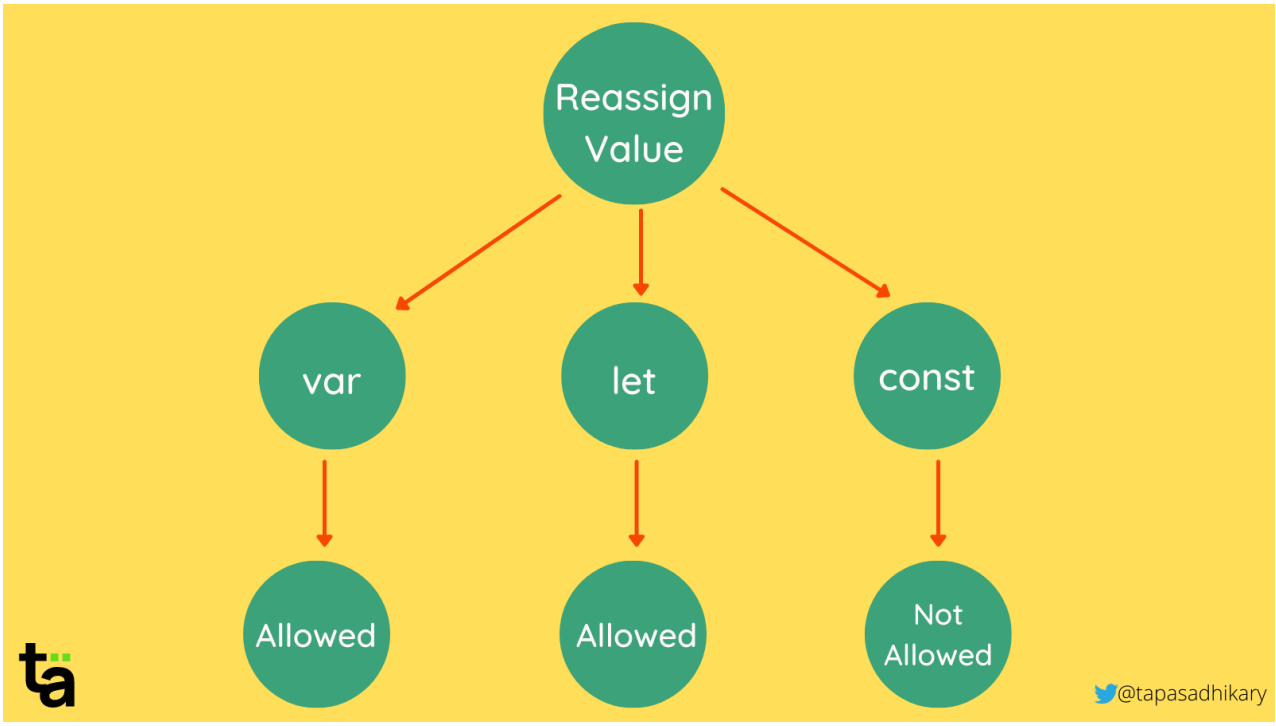
```
// Declare variables with initial values
let f_name = "Alex";
const ZIP = 560089;
var age = 25;

// Reassign values
f_name = "Bob"; // the f_name value is 'Bob"
ZIP = 65457; // Uncaught TypeError: Assignment to constant variable.
age = 78; // the age value is 78
```

There is a tricky part with `cost` that you must be aware of. When an object is declared and assigned a value with `const`, you can still change the value of its `properties`. But you can not reassign another object value to the same variable. This is a common mistake many devs make.

Check out the example here:

```
const blog = {
    'url': 'https://greenroots.info'
}

blog.url = 'https://blog.greenroots.info"; //Allowed

blog = {}; // Uncaught TypeError: Assignment to constant variable.
```

Here is a mindmap to help you grasp how reassigning works for variables declared with these three keywords.



What Happens When You Access a Variable Before Declaring it in JavaScript

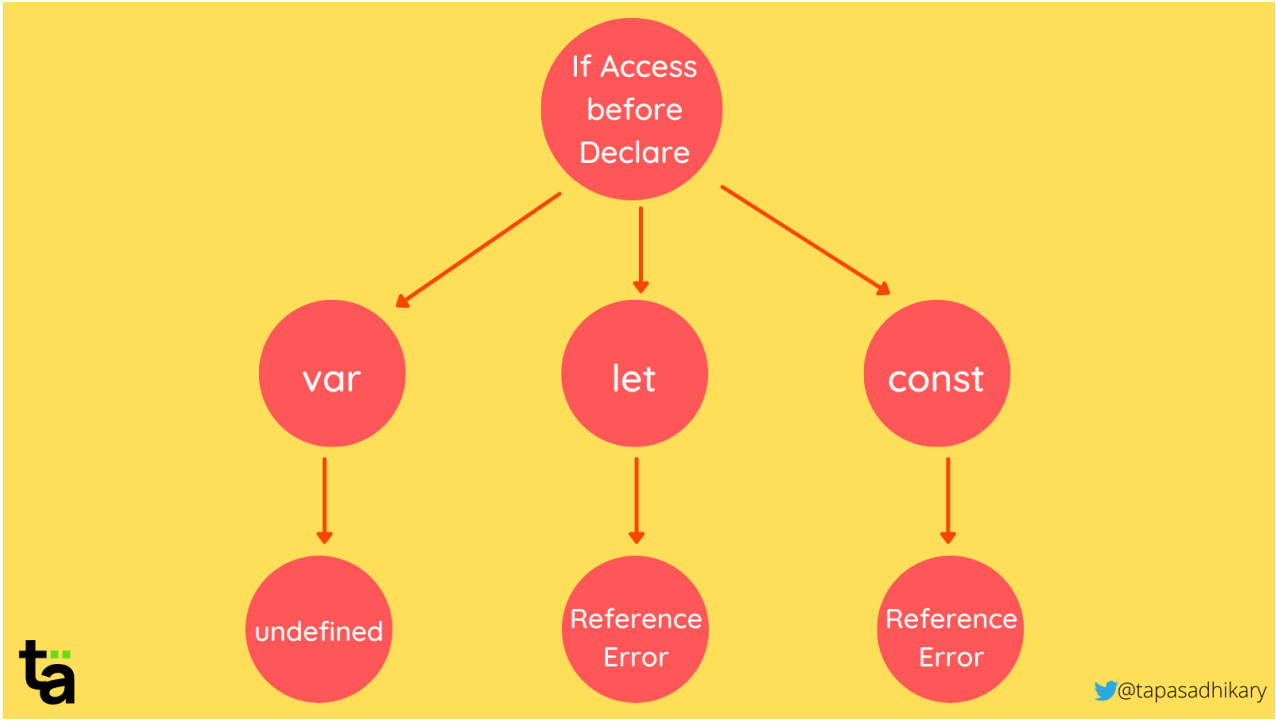
As a pragmatic programmer, you should never try accessing a variable without declaring it. But in case it happens, let's see how the variable may behave.

With `var` in non-strict mode, the variable will have an `undefined` value. This means that a variable has been declared but has no value assigned.

In strict mode, you will get a `ReferenceError` that the variable is not declared.

With `let` and `const`, if you try to access a variable before declaring, you will always get a `ReferenceError`.

Here is a mindmap again to help you understand it visually. In the mindmap, `var` is depicted for non-strict mode.



That's all, my friends. You need to consider these circumstances and concepts to evaluate how `var`, `let`, and `const` behave. So, the rule goes:

- Don't use `var` anymore.
- Use `let` or `const`.
- Use `const` more often. Use `let` when you need to reassign another value to a variable.
- Don't try to access a variable without declaring it.

Before We End...

That's the story behind `let`, `const`, and `var`. I hope you found the article insightful and informative. My DMs are open on `Twitter` if you want to discuss further.

Let's connect. I share my learnings on JavaScript, Web Development, and Blogging on these platforms as well:

See you soon with my next article. Until then, please take care of yourself, and stay happy.

TAPAS ADHIKARY
Writer . YouTuber . Creator . Mentor

If you read this far, tweet to the author to show them you care. Tweet a thanks