

Var, let and const- what's the difference?



Sarah Chima  Oct 25 '17 Updated on Nov 07, 2017 · 5 min read

#es6 #javascript #newbie

A lot of shiny new features came with ES2015 (ES6) and since it's 2017, it's assumed that a lot of JavaScript developers have become familiar with and have started using these features. While this assumption might be true, it's still possible that some of these features remain a mystery to some.

One of the features that came with ES6 is addition of `let` and `const` which can be used for variable declaration. The question now is, what makes them different from our good ol' `var` which has been in use? If you are still not clear about this, this article is for you.

In this article, `var`, `let` and `const` will be discussed with respect to their scope, use and hoisting. As you read, take note of the differences between them I'll point out.

VAR

Before the advent of ES6, `var` declarations ruled as King. There are issues associated with variables declared with `var` though. That is why it was necessary for new ways to declare variables

to emerge. First though, let us get to understand `var` more before we discuss one of such issues.

Scope of `var`

Scope essentially means where these variables are available for use. `var` declarations are globally scoped or function/locally scoped. It is globally scoped when a `var` variable is declared outside a function. This means that any variable that is declared with `var` outside a function block is available for use in the whole window. `var` is function scoped when it is declared within a function. This means that it is available and can be accessed only within that function.

To understand further, look at the example below.

```
var greeter = "hey hi";

function newFunction() {
    var hello = "hello";
}
```

Here, `greeter` is globally scoped because it exists outside a function while `hello` is function scoped. So we cannot access the variable `hello` outside of a function. So if we do this:

```
var tester = "hey hi";

function newFunction() {
    var hello = "hello";
}
```

```
}  
console.log(hello); // error: hello is not defined
```

We'll get an error which is as a result of `hello` not being available outside the function.

var variables can be re-declared and updated

That means that we can do this within the same scope and won't get an error.

```
var greeter = "hey hi";  
var greeter = "say Hello instead";
```

and this also

```
var greeter = "hey hi";  
greeter = "say Hello instead";
```

Hoisting of var

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution. What this means is that if we do this:

```
console.log (greeter);  
var greeter = "say hello"
```

it is interpreted as this

```
var greeter;  
console.log(greeter); //greeter is undefined  
greeter = "say hello"
```

So `var` variables are hoisted to the top of its scope and initialized with a value of undefined.

Problem with `var`

There's a weakness that comes with `var` . I'll use the example below to explain this.

```
var greeter = "hey hi";  
var times = 4;  
  
if (times > 3) {  
    var greeter = "say Hello instead";  
}  
  
console.log(greeter) //"say Hello instead"
```

So, since `times > 3` returns true, `greeter` is redefined to `"say Hello instead"` . While this is not a problem if you knowingly

want `greeter` to be redefined, it becomes a problem when you do not realize that a variable `greeter` has already been defined before.

If you have use `greeter` in other parts of your code, you might be surprised at the output you might get. This might cause a lot of bugs in your code. This is why the `let` and `const` is necessary.

LET

`let` is preferred for variable declaration now. It's no surprise as it comes as an improvement to the `var` declarations. It also solves this problem that was raised in the last subheading. Let's consider why this is so.

`let` is block scoped

A block is chunk of code bounded by `{}`. A block lives in curly braces. Anything within curly braces is a block. So a variable declared in a block with the `let` is only available for use within that block. Let me explain this with an example.

```
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
```

We see that using `hello` outside its block(the curly braces where it was defined) returns an error. This is because `let` variables are block scoped .

`let` can be updated but not re-declared.

Just like `var` , a variable declared with `let` can be updated within its scope. Unlike `var` , a `let` variable cannot be re-declared within its scope. So while this will work,

```
let greeting = "say Hi";  
greeting = "say Hello instead";
```

this will return an error.

```
let greeting = "say Hi";  
let greeting = "say Hello instead";//error: Identifier 'greeting' has already been declared
```



However, if the same variable is defined in different scopes, there will be no error.

```
let greeting = "say Hi";  
if (true) {  
  let greeting = "say Hello instead";  
  console.log(greeting);//"say Hello instead"  
}  
console.log(greeting);//"say Hi"
```

Why is there no error? This is because both instances are treated as different variables since they have different scopes.

This fact makes `let` a better choice than `var`. When using `let`, you don't have to bother if you have used a name for a variable before as a variable exists only within its scope. Also, since a variable cannot be declared more than once within a scope, then the problem discussed earlier that occurs with `var` does not occur.

Hoisting of `let`

Just like `var`, `let` declarations are hoisted to the top. Unlike `var` which is initialized as `undefined`, the `let` keyword is not initialized. So if you try to use a `let` variable before declaration, you'll get a `Reference Error`.

CONST

Variables declared with the `const` maintain constant values. `const` declarations share some similarities with `let` declarations.

`const` declarations are block scoped

Like `let` declarations, `const` declarations can only be accessed within the block it was declared.

`const` cannot be updated or re-declared

This means that the value of a variable declared with `const` remains the same within its scope. It cannot be updated or re-declared. So if we declare a variable with `const`, we can neither do this

```
const greeting = "say Hi";  
greeting = "say Hello instead";//error : Assignment to constant variable.
```



nor this

```
const greeting = "say Hi";  
const greeting = "say Hello instead";//error : Identifier 'greeting' has a
```



Every `const` declaration therefore, must be initialized at the time of declaration.

This behavior is somehow different when it comes to objects declared with `const`. While a `const` object cannot be updated, the properties of this objects can be updated. Therefore, if we declare a `const` object as this

```
const flow = {  
  message : "say Hi",  
  times : 4  
}
```


while we cannot do this

```
const greeting = {  
  words : "Hello",  
  number : "five"  
} //error : Assignment to constant variable.
```

we can do this

```
greeting.message = "say Hello instead";
```

This will update the value of `greeting.message` without returning errors.

Hoisting of `const`

Just like `let`, `const` declarations are hoisted to the top but are not initialized.

So just in case, you missed the differences, here they are :

1. `var` declarations are globally scoped or function scoped while `let` and `const` are block scoped.
2. `var` variables can be updated and re-declared within its scope; `let` variables can be updated but not re-declared; `const` variables can neither be updated nor re-declared.

3. They are all hoisted to the top of their scope but while `var` variables are initialized with `undefined`, `let` and `const` variables are not initialized.
4. While `var` and `let` can be declared without being initialized, `const` must be initialized during declaration.

Got any question or addition? please leave a comment.

Thank you for reading :)



Sarah Chima + FOLLOW

A Frontend web developer interested in making the web accessible for everyone.

@sarah_chima sarah_chima sarahchima.com

Add to the discussion



PREVIEW

SUBMIT



Dominic Sears

Oct 25 '17

For a noob like me, this explained "let" declarations for me.



12

REPLY



Sarah Chima

Oct 25 '17

I'm glad it did.



2

REPLY