There are now two new ways to declare variables in JavaScript: **let** and **const**.

Up until now, the only way to declare a variable in JavaScript was to use the keyword var. To understand why let and const were added, it’s probably best to look at an example of when using var can get us into trouble.

Take a look at the following code.

**QUESTION 1 OF 3**

What do you expect to be the output from running getClothing(false)?

**function** **getClothing**(isCold) {

**if** (isCold) {

**var** freezing = 'Grab a jacket!';

} **else** {

**var** hot = 'It’s a shorts kind of day.';

console.log(freezing);

}

}

* 

ReferenceError: freezing is not defined

* 

Grab a jacket!

* undefined
* 

It's a shorts kind of day.

SUBMIT

**Hoisting**

Hoisting is a result of how JavaScript is interpreted by your browser. Essentially, before any JavaScript code is executed, all variables declared with var are "hoisted", which means they're raised to the top of the function scope. So at run-time, the getClothing() function actually looks more like this…

Before the function is executed, all variables are hoisted to the top of the function scope. So what’s our solution?

**let and const**

Variables declared with let and const eliminate this specific issue of hoisting because they’re scoped **to the block**, not to the function. Previously, when you used var, variables were either scoped *globally* or *locally* to an entire function scope.

If a variable is declared using let or const inside a block of code (denoted by curly braces { }), then the variable is stuck in what is known as the **temporal dead zone** until the variable’s declaration is processed. This behavior prevents variables from being accessed only until after they’ve been declared.

Variables declared with let and const are only available within the block they're declared.

**QUESTION 2 OF 3**

What do you expect to be the output from running getClothing(false)?

**function** **getClothing**(isCold) {

**if** (isCold) {

**const** freezing = 'Grab a jacket!';

} **else** {

**const** hot = 'It’s a shorts kind of day.';

console.log(freezing);

}

}

* ReferenceError: freezing is not defined
* 

Grab a jacket!

* 

undefined

* 

It's a shorts kind of day.

SUBMIT

**Rules for using let and const**

let and const also have some other interesting properties.

* Variables declared with let can be reassigned, but can’t be redeclared in the same scope.
* Variables declared with const must be assigned an initial value, but can’t be redeclared in the same scope, and can’t be reassigned.

**QUESTION 3 OF 3**

What do you expect to be output from running the following code?

**let** instructor = 'James';

instructor = 'Richard';

console.log(instructor);

* 

James

* Richard
* 

undefined

* 

SyntaxError: Identifier 'instructor' has already been declared

SUBMIT

**Use cases**

The big question is when should you use let and const? The general rule of thumb is as follows:

* use let when you plan to reassign new values to a variable, and
* use const when you don’t plan on reassigning new values to a variable.

Since const is the strictest way to declare a variable, we suggest that you always declare variables with const because it'll make your code easier to reason about since you know the identifiers won't change throughout the lifetime of your program. If you find that you need to update a variable or change it, then go back and switch it from const to let.

That’s pretty straightforward, right? But what about var?

**What about var?**

Is there any reason to use var anymore? *Not really*.

There are some arguments that can be made for using var in situations where you want to globally define variables, but this is often considered bad practice and should be avoided. From now on, we suggest ditching var in place of using let and const.