ES2015 Scope & Hoisting - let & const

We'll learn how the new keywords 'let' and 'const' eliminate the problems of variable hoisting, scope limitations, and unintended reassignments. We'll show why 'var' should never be used again.

ES2015 introduces two new ways to declare variables. The keywords `let` and `const`. If you're not familiar with these yet, here are the basics of these two keywords.

`let`

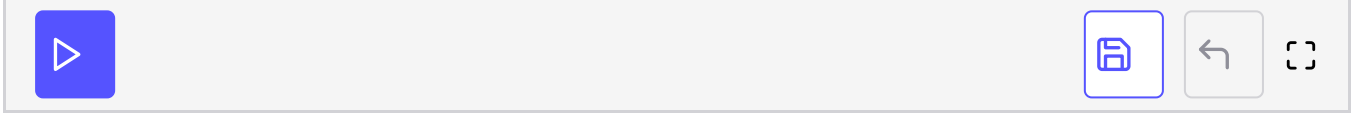
`let` is similar to `var`. It allows us to declare variables in our local scope. It's used the same way, `let x = 4;`. The differences are that `let` is:

- not hoisted
- block-scoped

Not Hoisted

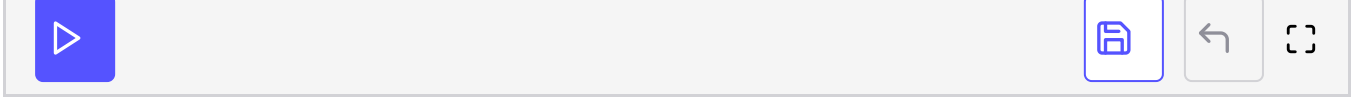
We've seen that variable declarations using `var` get hoisted to the top of their scope.

```
console.log(x); // -> undefined
var x = 10;
console.log(x); // -> 10
```



Variables declared with `let`, however, are not hoisted. Attempting to do the same with `let` will cause a reference error. `x` exists only at and after the line on which it was written.

```
console.log(x); // -> Uncaught ReferenceError: x is not defined
let x = 10;
console.log(x);
```



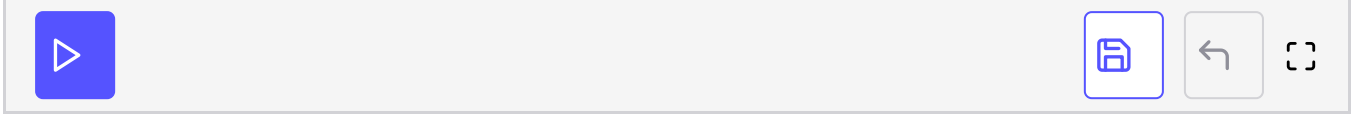
Simple as that. No more unexpected behavior or buggy code due to variable hoisting. It's still important to know the concept, however, as you'll likely deal with pre-2015 code often in the near future. Hoisting even appears in the occasional job interview exercise.

Block-Scoped

`var` is *function scoped*. The only way for it be locally scoped is inside a function.

`let` is *block-scoped*. This is a brand new scoping mechanism in JavaScript. Scope is no longer limited to functions and global. Any pair of arbitrary curly brackets `{}` creates a new scope and variables declared with `let` are available only in those curly brackets.

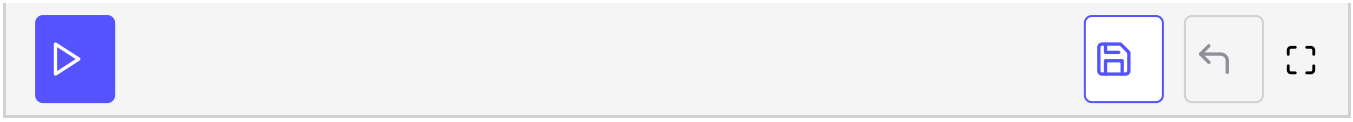
```
{
  var x = 10;
  let y = 20;

  console.log(x); // -> 10
  console.log(y); // -> 20
}

console.log(x); // -> 10
console.log(y); // -> Uncaught ReferenceError: y is not defined
```



This extends to loops as well. A variable declared with `let` in a `for` or `while` loop is unavailable outside the loop, while `var`-declared variables are available.

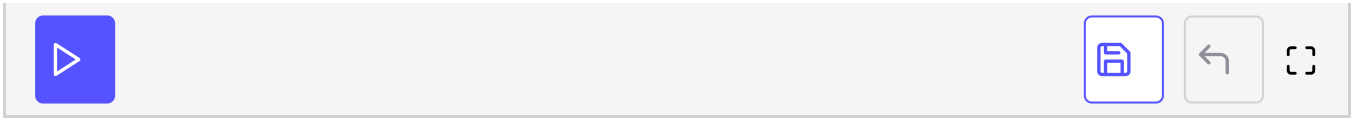
Variables declared in the conditions of a loop are also scoped only to the loop, even though they're not technically inside the brackets.

```
for (var i = 0; i < 10; i++) {
}

for (let j = 0; j < 10; j++) {
}

console.log(i); // -> 10
console.log(j); // -> Uncaught ReferenceError: j is not defined
```



Duplicate declaration also fails with `let`, unlike with `var`.

```
var x = 4;
var x = 5;

let y = 6;
let y = 7; // -> Uncaught SyntaxError: Identifier 'y' has already been declared
```



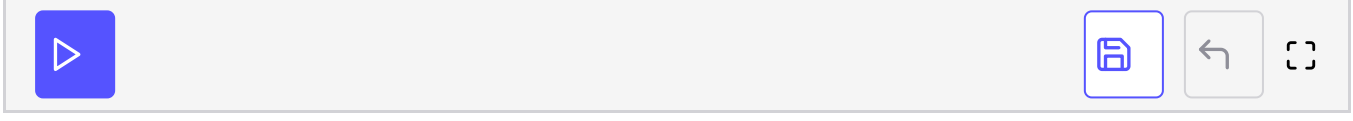
The engine re-declares a variable declared with `var` just fine. Attempting to do the same with `let` causes an error.

`const`

No reassignment

All of the new rules applying to `let` also apply to `const`. `const` has the added effect that inside its scope, a variable declared with `const` cannot be reassigned.

```
var x = 3;
let y = 4;
const z = 5;

x = 30;
y = 40;
z = 50; // -> Uncaught TypeError: Assignment to constant variable.
```

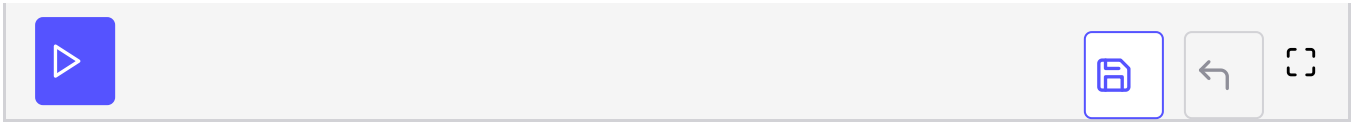


No empty initializations

An empty initialization is not allowed with `const`. Whereas using `var x;` or `let x;` would put the value `undefined` in `x`, `const x;` throws an error. It must be given a value on declaration.

```
var a;
let b;
```

```
const c; // -> Uncaught SyntaxError: Missing initializer in const declaration
```



Usage

The fact is that `var` is obsolete. There is absolutely no good reason to use it ever again. Use `let` or `const` entirely from here on out.

Which one?

Preferring `const`

There are two schools of thought about `let` and `const`. The first is to use `const` wherever possible. You'll find that variables that need to be reassigned are much less common than constant variables.

Using `const` wherever we can indicates that when we see a `let`, we know the variable is meant to be reassigned somewhere. We would strive to only use `const`, falling back to `let` when necessary.

Preferring `let`

Others, including Kyle Simpson, author of the "You Don't Know JS" series, prefer using `let` over `const`. The idea is that `const` should be reserved for true constants, mathematical or otherwise. Writing `const PI = 3.14159;` would be appropriate, as pi is a pure mathematical constant that should never change.

I personally prefer the first strategy, using `const` wherever possible and demoting a variable to a `let` declaration when reassignment is necessary. I believe it leads to code that is easier to read and it's what I'll use in code blocks in the rest of the course. The decision's up to you.