Chapter 9 of AR’s book Exploring ES6

let and const behave more strictly and throw more exceptions (e.g. when you access their variables inside their scope before they are declared). Block-scoping helps with keeping the effects of code fragments more local (see the next section for a demonstration). And it’s more mainstream than function-scoping, which eases moving between JavaScript and other programming languages.

Because behavior differs between var and let, you can’t blindly replace var with let or const in existing code; you have to be careful during refactoring

Standard seems to be to avoid var, preferably use const and, when needed, let

let works similarly to var, but the variable it declares is block-scoped

const works like let, but the variable you declare must be immediately initialized and the value cannot be changed afterwards – values are read-only

it has to do with the binding of consts in memory

in for-of loops, each iteration creates a new storage space for a const, you can const declare a loop variable

for (const x of ['a', 'b']) {

console.log(x);

}

The lack of hoisting for let and const causes temporal dead zone

In ECMAScript 6, accessing a let or const variable before its declaration (within its scope) causes a ReferenceError . The time span when that happens, between the creation of a variable's binding and its declaration, is called the temporal dead zone.

