One thing I like about JavaScript is the const declaration method,  
which allows you to declare a variable one time, and that variable can’t  
be reassigned after that. I.e, this piece of code will throw an error:

node -e "const x = 12; x = 14"

## [eval]:1

## const x = 12; x = 14

## ^

##

## TypeError: Assignment to constant variable.

## at [eval]:1:18

## at Script.runInThisContext (vm.js:124:20)

## at Object.runInThisContext (vm.js:314:38)

## at Object. ([eval]-wrapper:9:26)

## at Module.\_compile (internal/modules/cjs/loader.js:805:30)

## at evalScript (internal/process/execution.js:60:25)

## at internal/main/eval\_string.js:16:1

The cool thing about this is that you can’t override the variable by  
mistake: once it’s set, it’s set. On the other hand, R allows you to  
override almost any variable (well, except some reserved variables).

I asked Twitter if there was any implementation of that concept in R.  
The use case, for example, would arise when you have a value that takes  
some time to compute. If I do my computation, I can accidentally  
override it later on. Event more if you’re using notebook, where you  
create symbols and values all along your document.

a <- some\_very\_complex\_computation()

# [...] Going on the weekend

a <- "Hello there!"

Here, I have no way to prevent myself from erasing the value in a. Of  
course, there are always rigor, explicit variable name, and  
not-assigning-things-without-thinking but you know how it is in the real  
world, and there is no Cmd + Z there.

pointed out that ?lockBinding existed, and that it was what I was  
looking for. And that does.

Here’s how it works: it takes a character string referring to a symbol,  
and an environment, and prevents from assigning any new value to this  
symbol in the given environment.

x <- 12

lockBinding("x", .GlobalEnv)

x <- 13

## Error in eval(expr, envir, enclos): cannot change value of locked binding for 'x'

And here’s a small wrapper to do that:

lock <- function(x){

lockBinding(

deparse(

substitute(x)),

env = parent.frame()

)

}

plop <- 12

lock(plop)

plop <- 13

## Error in eval(expr, envir, enclos): cannot change value of locked binding for 'plop'

pouet <- function(){

plop <- 14

print(plop)

lock(plop)

plop <- 13

}

pouet()

## [1] 14

## Error in pouet(): cannot change value of locked binding for 'plop'

So there I could do

a <- some\_very\_complex\_computation()

lock(a)

# [...] Going on the weekend

a <- "Hello there!"

And there, I have prevented myself from erasing my a variable. Of  
course, it’s not the same as JavaScript const, as there is always a  
way to unlock the  
symbol.

x <- 12

## Error in eval(expr, envir, enclos): cannot change value of locked binding for 'x'

lock(x)

x <- 13

## Error in eval(expr, envir, enclos): cannot change value of locked binding for 'x'

unlockBinding("x", .GlobalEnv)

x <- 13

x

## [1] 13

But I think it’s a rather elegant solution for preventing yourself from  
unwanted variable overwriting.