Immutable JavaScript

Fernando Daciuk

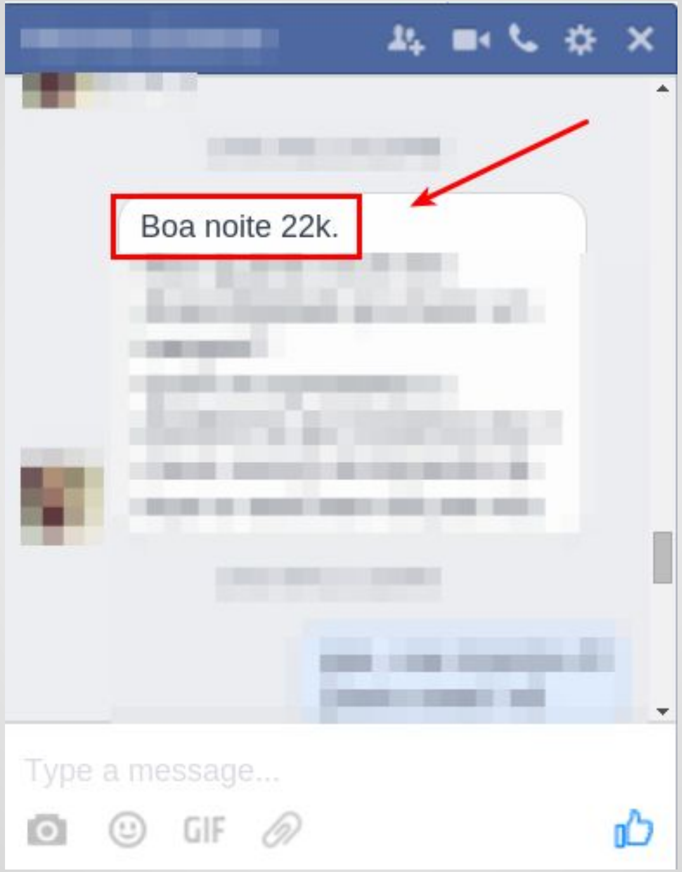
```
$ npm install fdaciuk
```

Fernando Daciuk

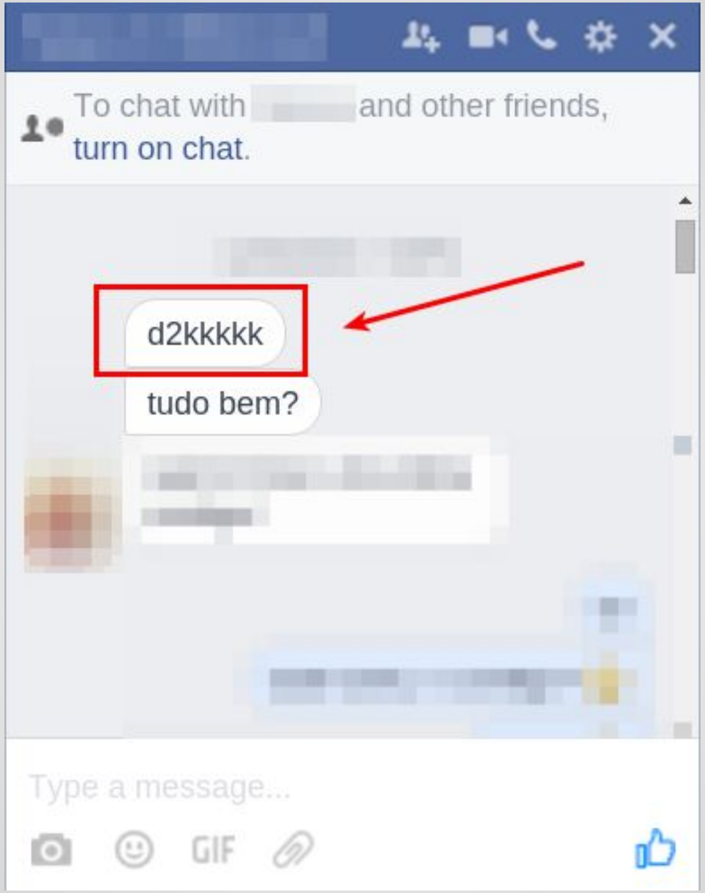
```
$ npm install fdaciuk
```

Daciuk = Da”Two”k

Daciuk = Da2k



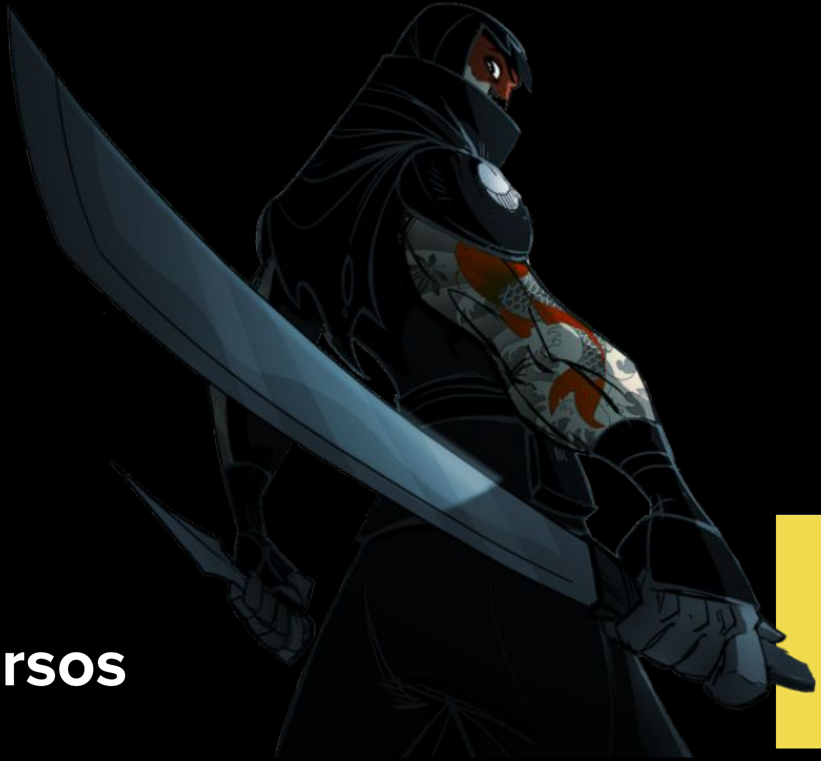
Boa noite 22k.


To chat with and other friends, turn on chat.

d2kkkkk

tudo bem?

JAVASCRIPT NINJA



https://blog.da2k.com.br/cursos

JS

React Ninja

https://blog.da2k.com.br/cursos

https://queroser.ninja/promocoes

Immutable JavaScript

Immutability

is really **easy**

to understand

Think in **something** that **NEVER** changes

That is

Immutability!

Okay, but **why** should I use **Immutability?**

#1

concise code

Variables must have

same value

from

start

to **finish**

#2

avoid bugs

Immutable **code** prevents **side-effects**

Bugs

usually

live in

mutable state

#3

thread safe

Immutable code

doesn't change

Therefore,

it doesn’t have

race

conditions

So, **how is that**

related to

JavaScript?

two

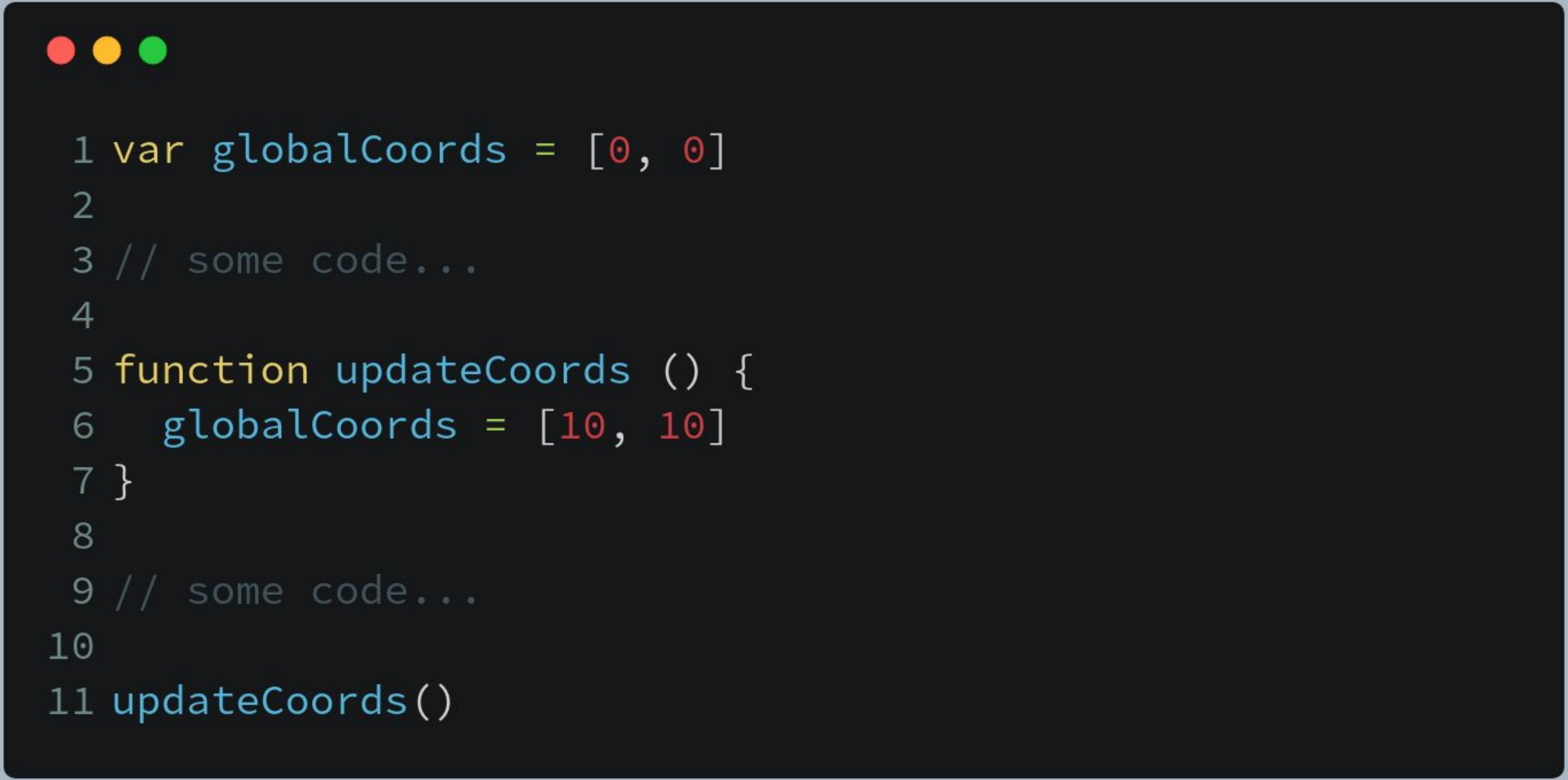
ways:

#1
assignment

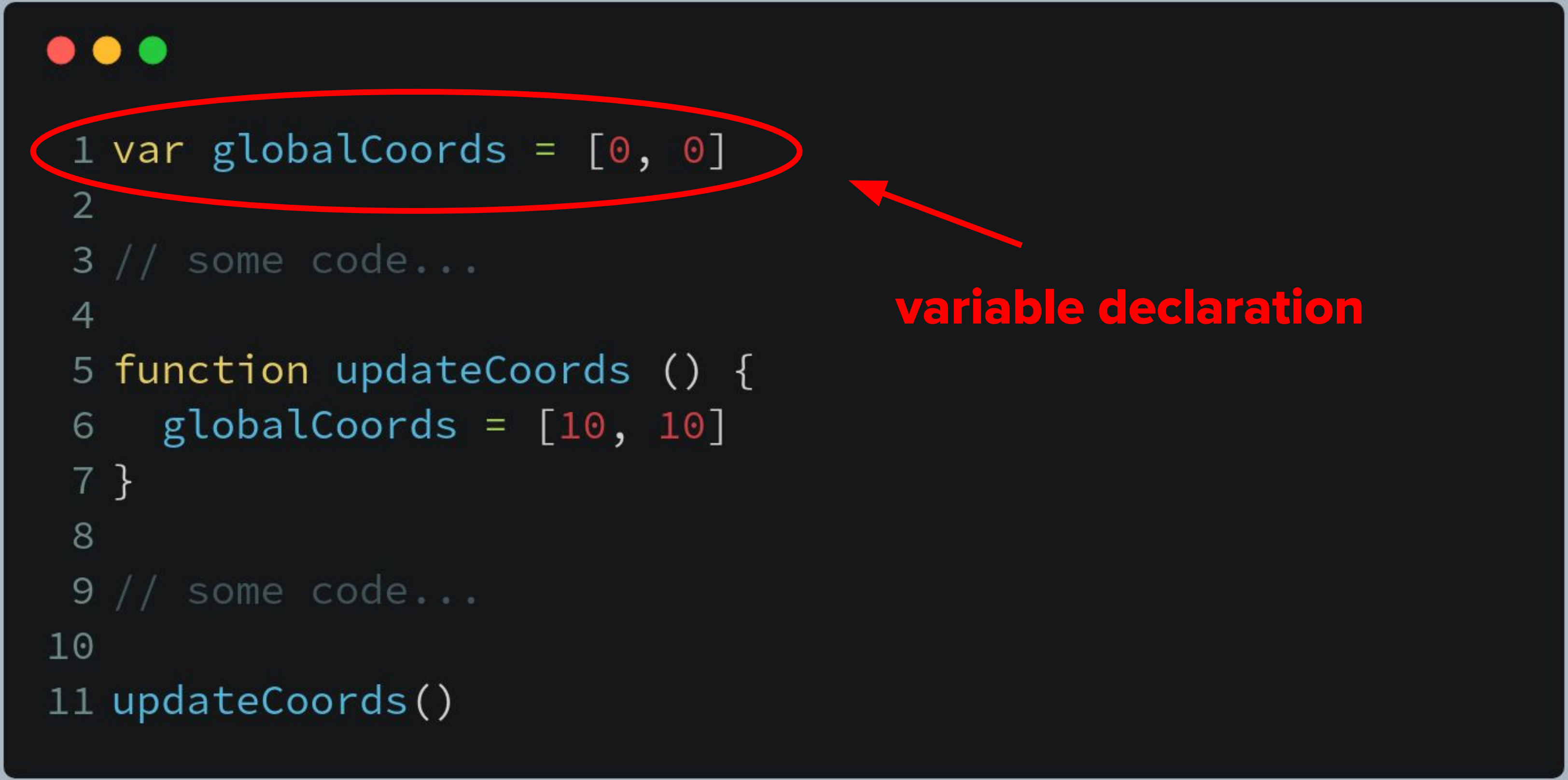
#2

objects

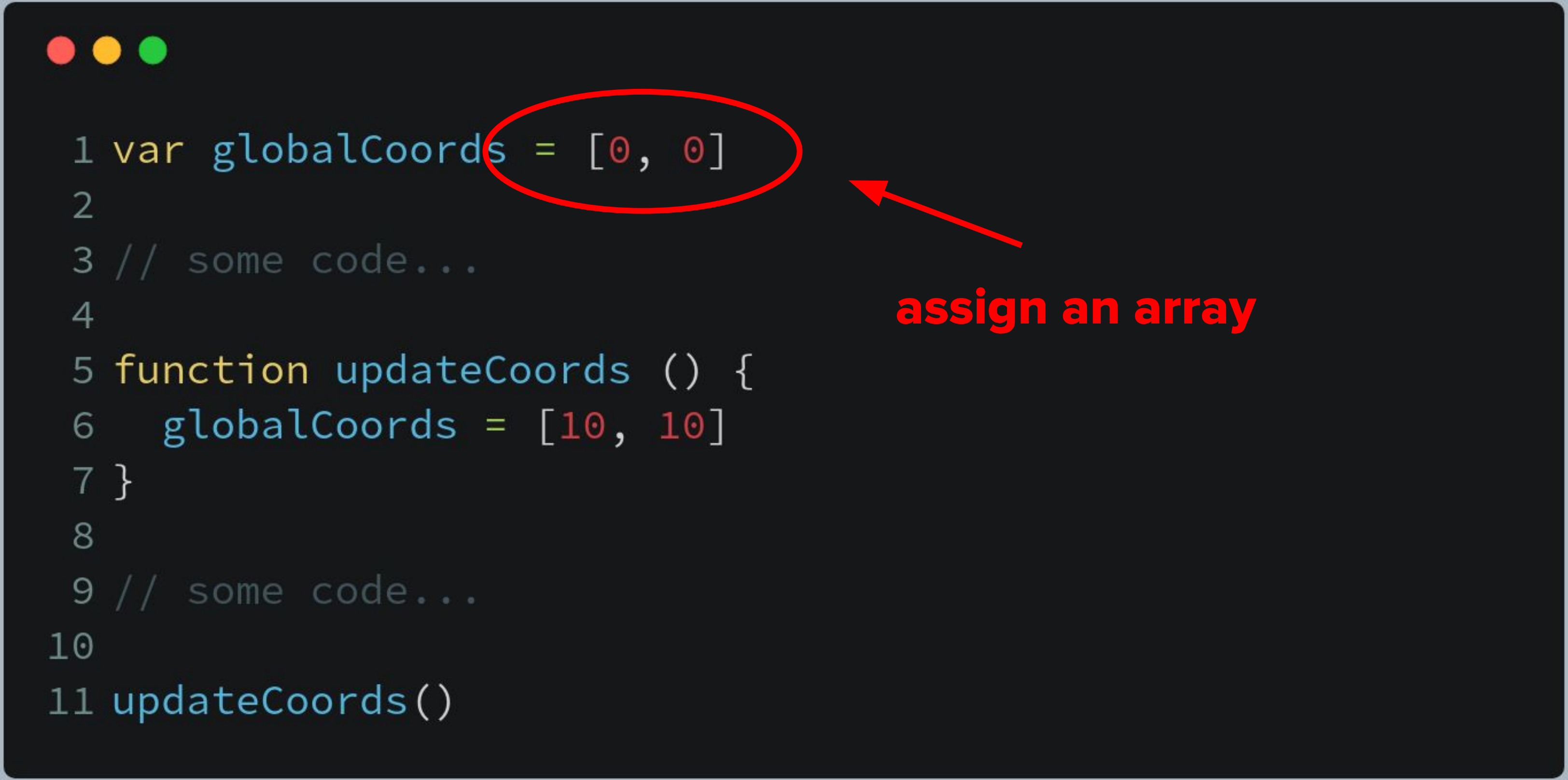
Let's see some code!



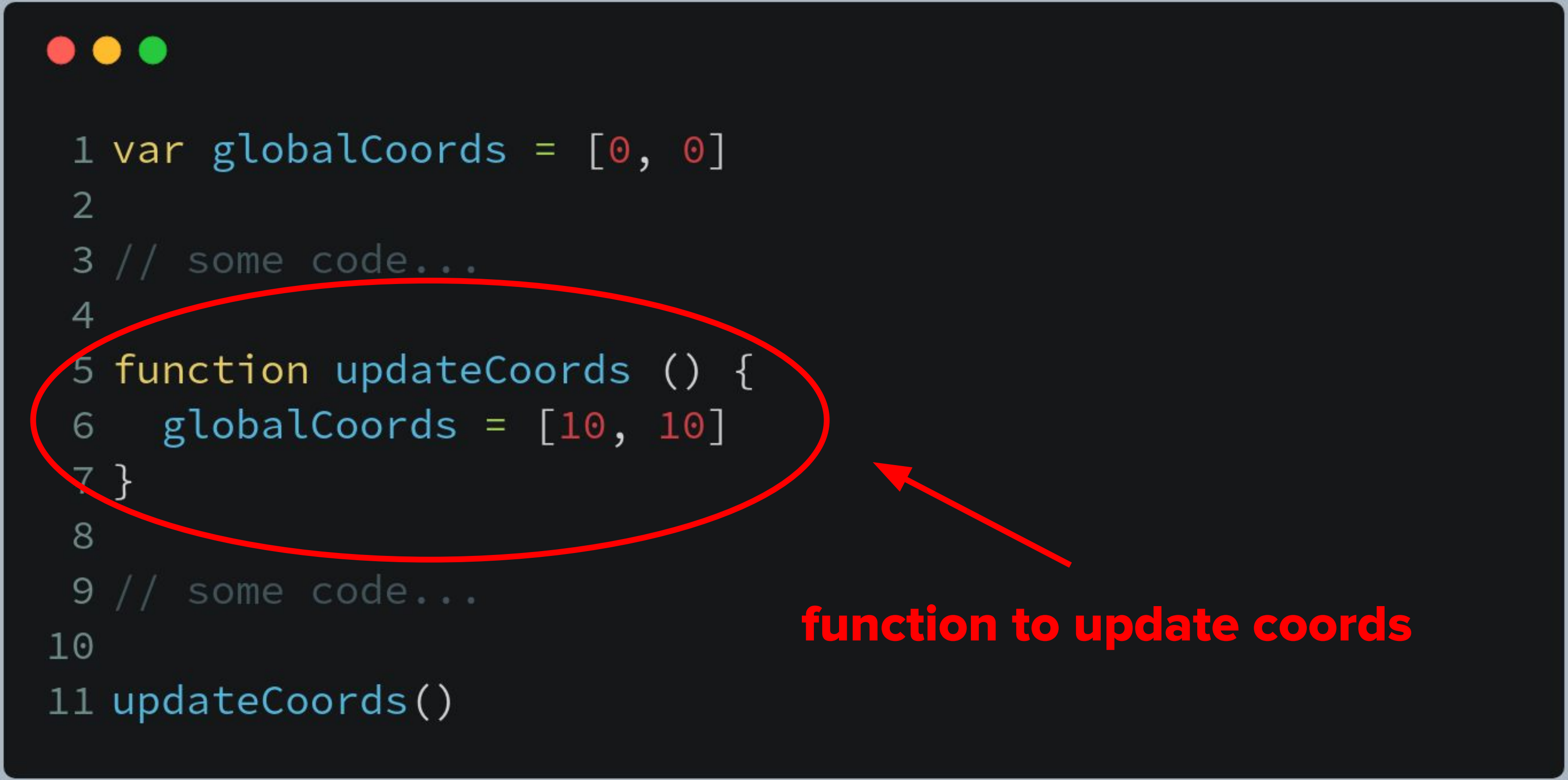
```javascript
var globalCoords = [0, 0]

// some code...

function updateCoords () {
  globalCoords = [10, 10]
}

// some code...

updateCoords()
```



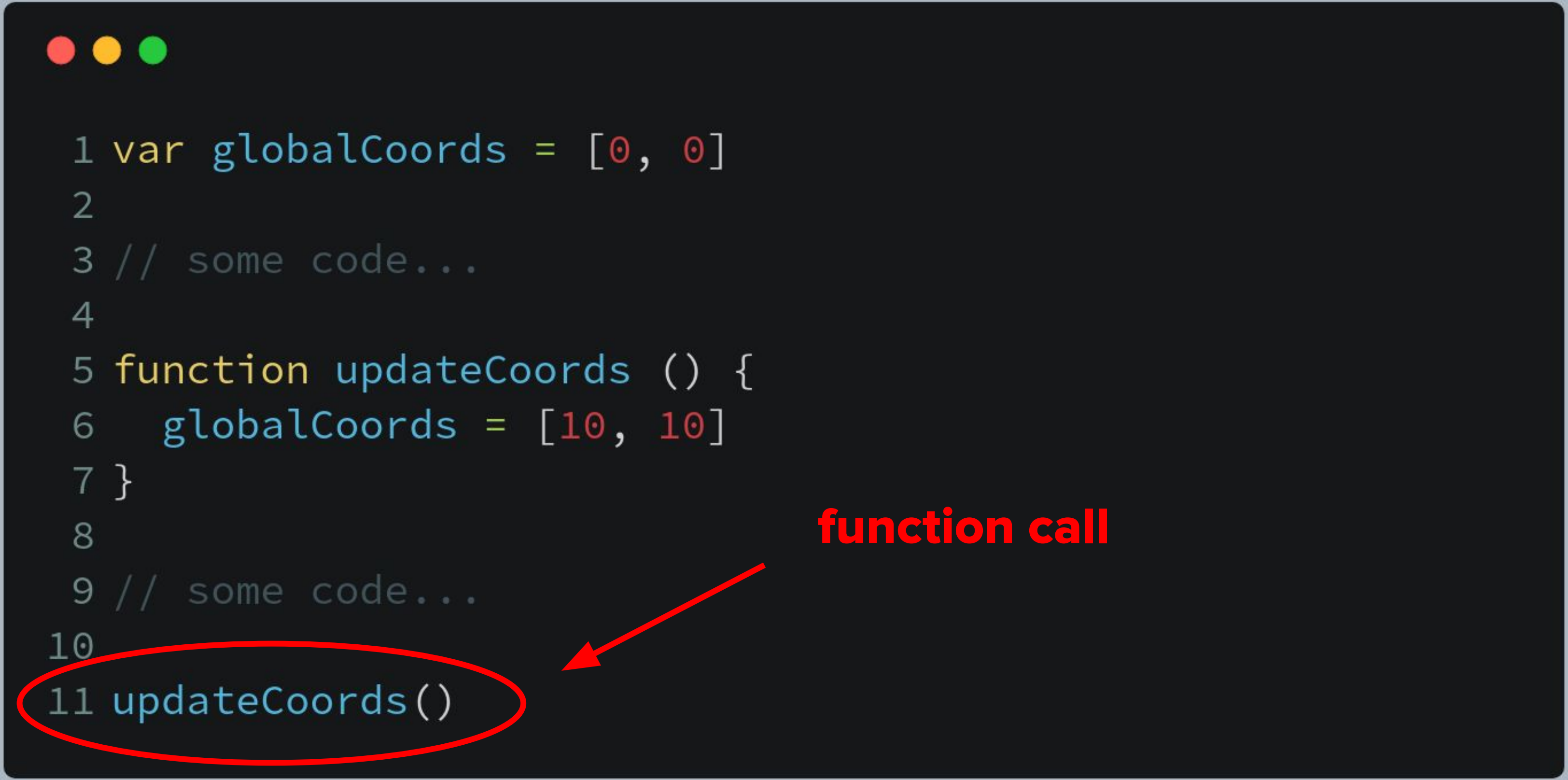
```javascript
var globalCoords = [0, 0]

// some code...

function updateCoords () {
  globalCoords = [10, 10]
}

// some code...

updateCoords()
```



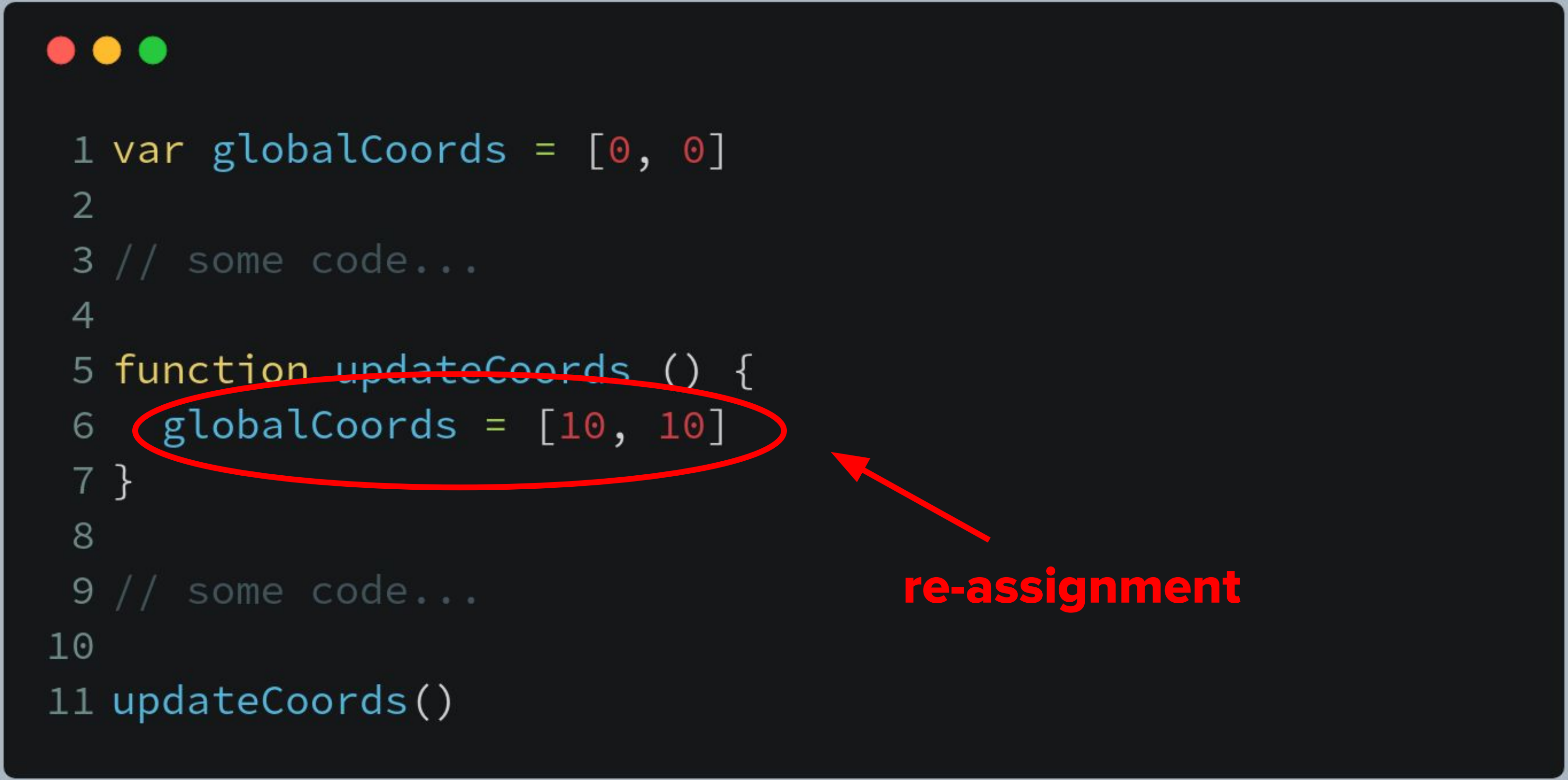
```
var globalCoords = [0, 0]

// some code...

function updateCoords () {
  globalCoords = [10, 10]
}

// some code...

updateCoords()
```



```
var globalCoords = [0, 0]

// some code...

function updateCoords () {
  globalCoords = [10, 10]
}

// some code...

updateCoords()
```



```
var globalCoords = [0, 0]

// some code...

function updateCoords () {
  globalCoords = [10, 10]
}

// some code...

updateCoords()
```

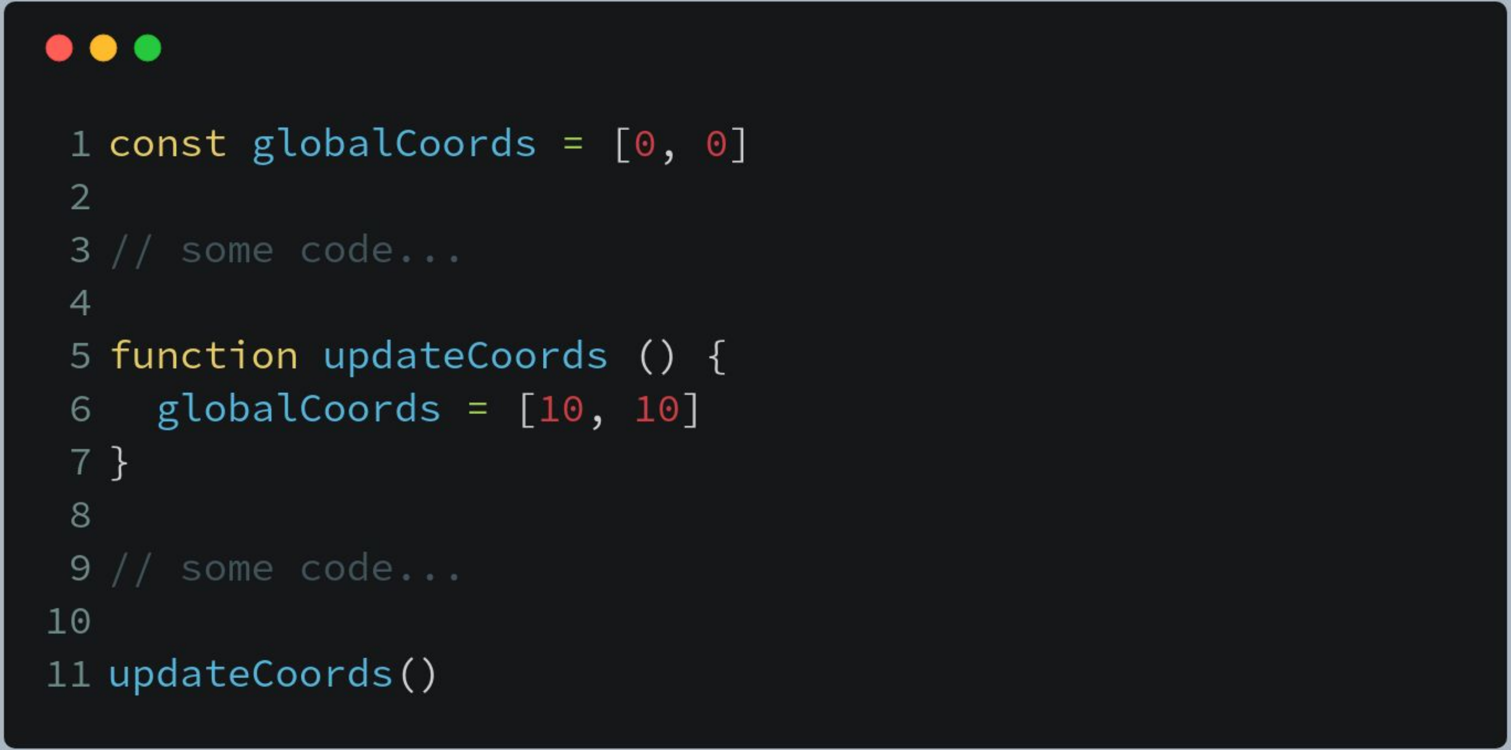


```
var globalCoords = [0, 0]

// some code...

function updateCoords () {
  globalCoords = [10, 10]
}

// some code...

updateCoords()
```

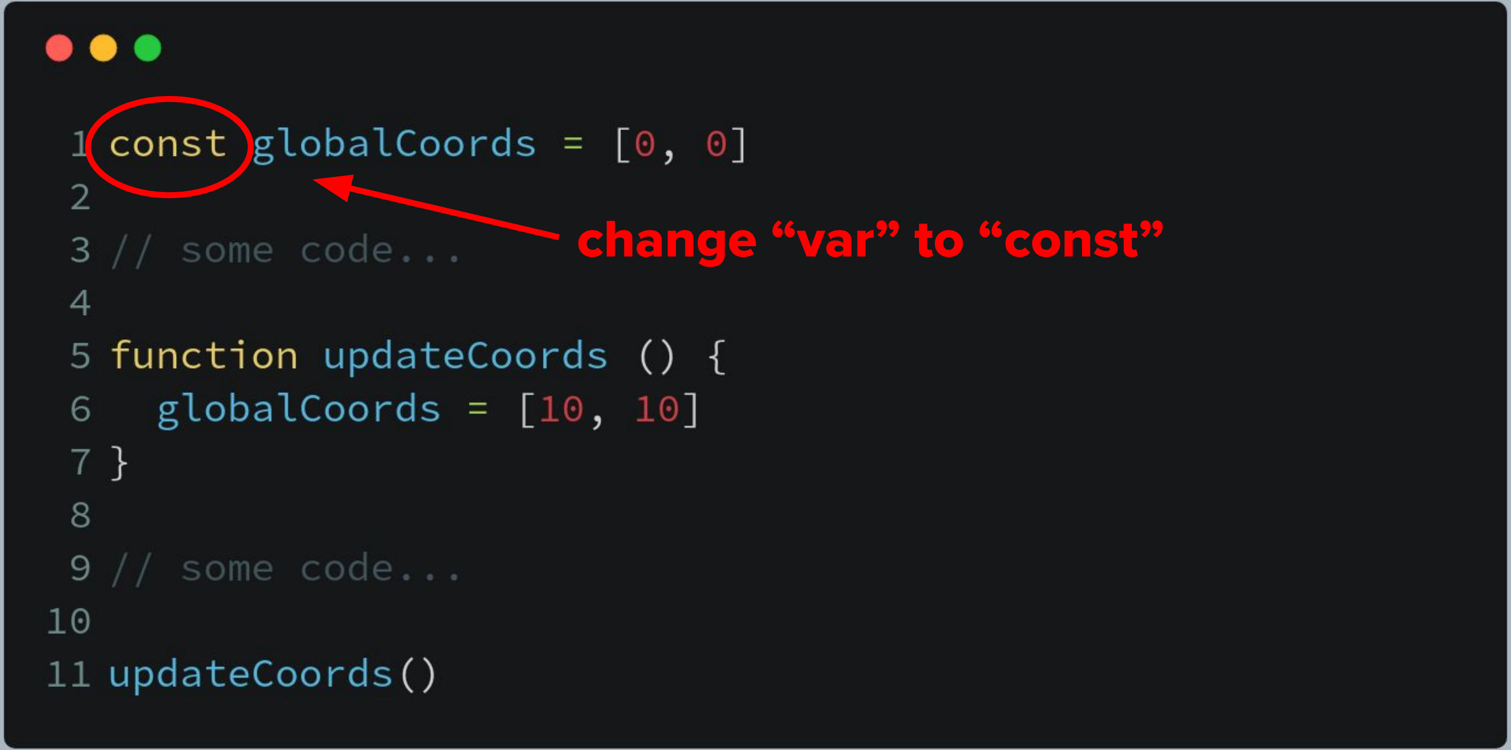
That is
mutability

by
assignment

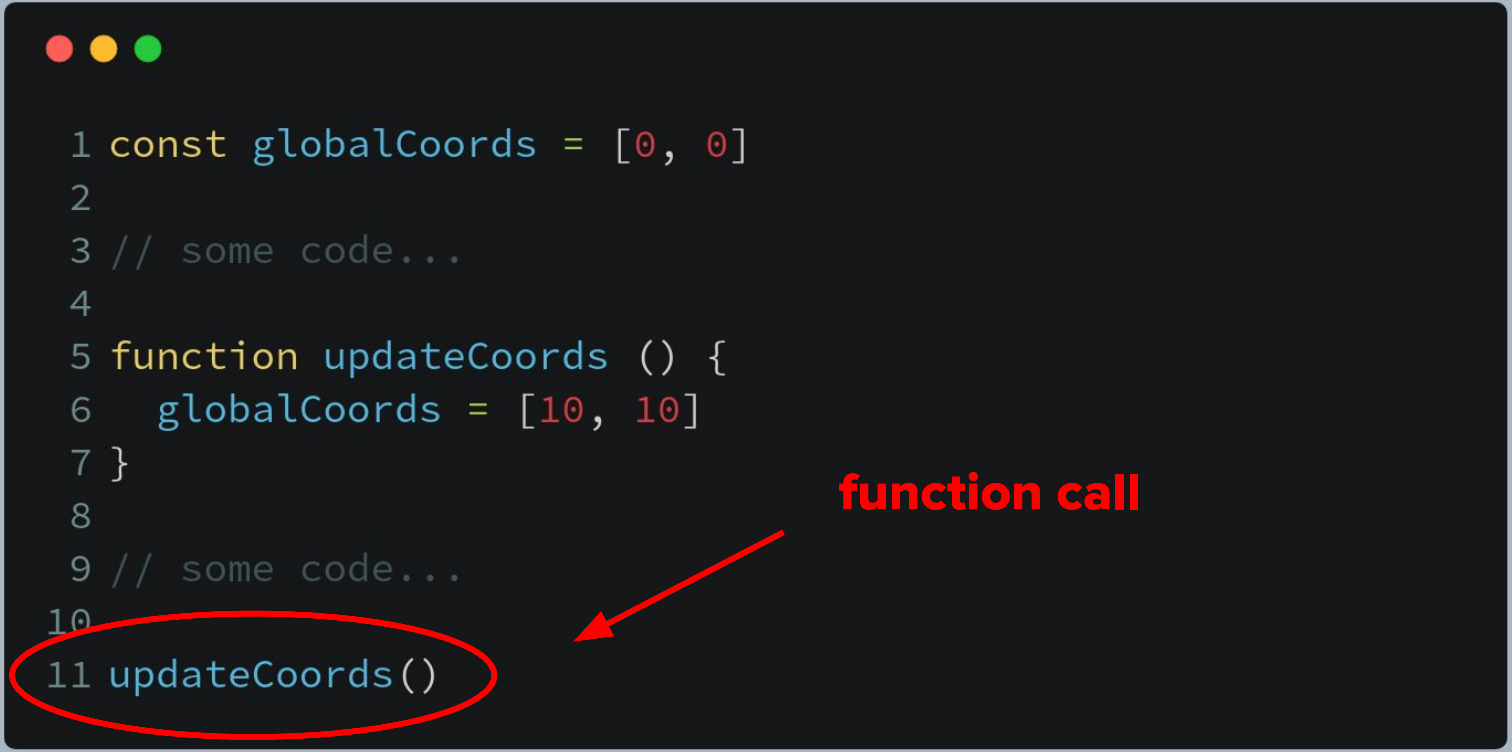
How to **make** that code **immutable?**



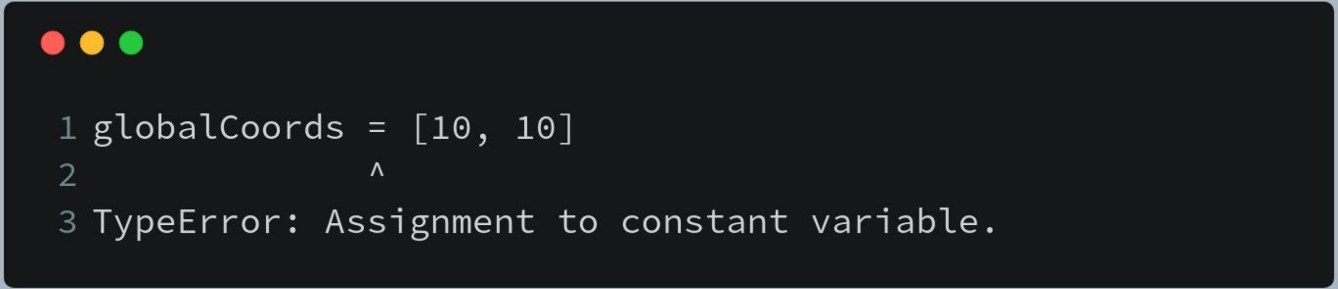
```
const globalCoords = [0, 0]

// some code...

function updateCoords () {
  globalCoords = [10, 10]
}

// some code...

updateCoords()
```



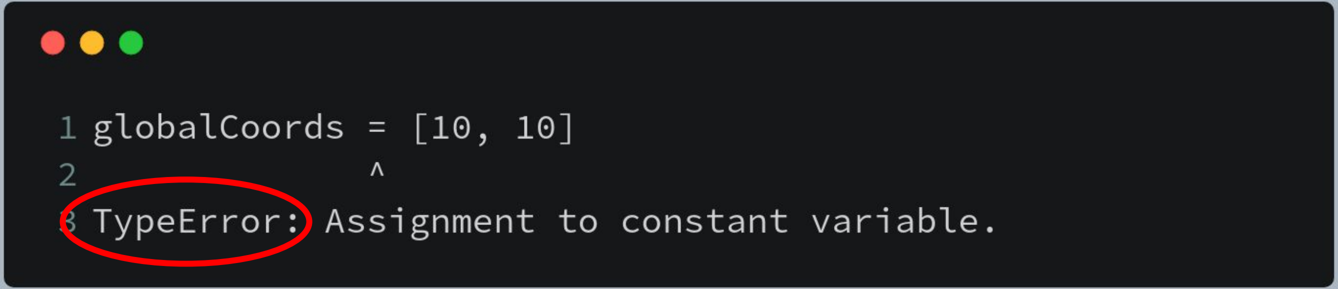
```
const globalCoords = [0, 0]

// some code...

function updateCoords () {
  globalCoords = [10, 10]
}

// some code...

updateCoords()
```



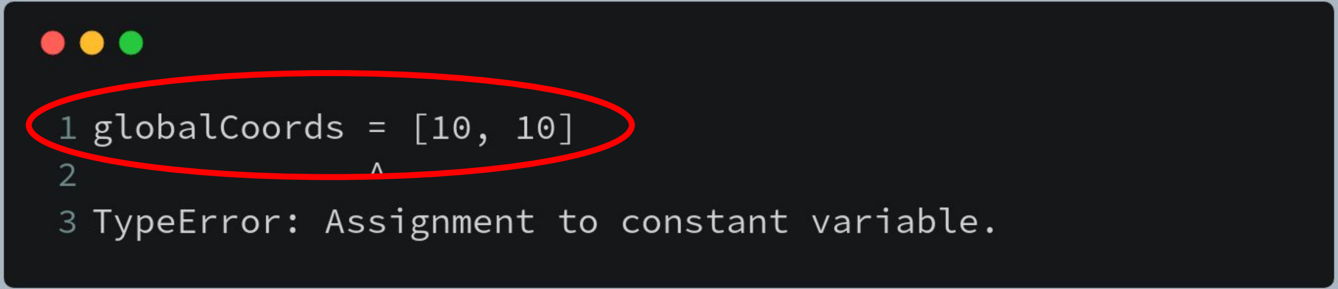
```
const globalCoords = [0, 0]

// some code...

function updateCoords () {
  globalCoords = [10, 10]
}

// some code...

updateCoords()
```



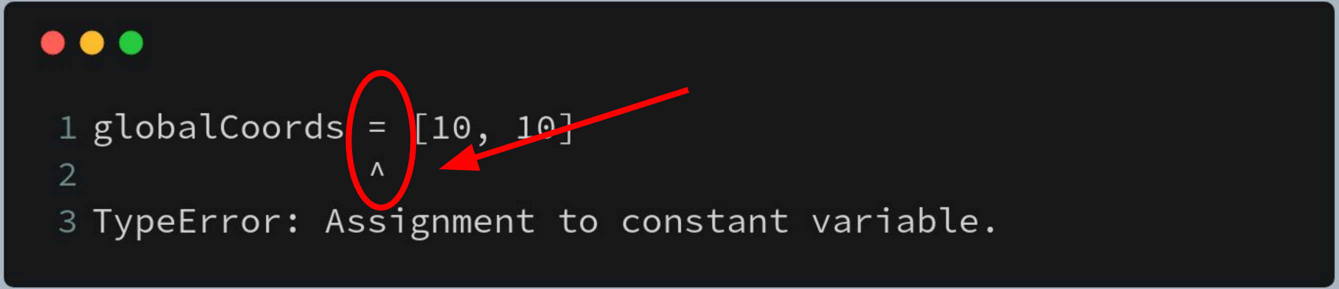
```
globalCoords = [10, 10]
             ^
TypeError: Assignment to constant variable.
```



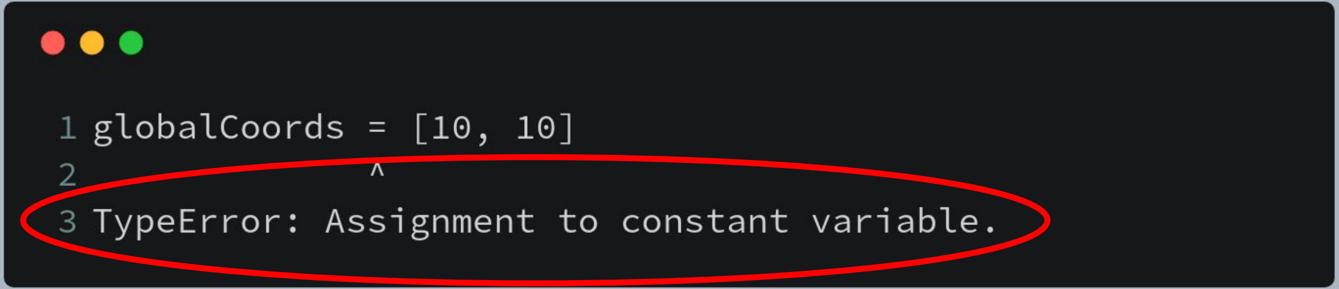
```
globalCoords = [10, 10]
             ^
TypeError: Assignment to constant variable.
```



```
globalCoords = [10, 10]
                  ^
TypeError: Assignment to constant variable.
```



```
globalCoords = [10, 10]
             ^
TypeError: Assignment to constant variable.
```



```
globalCoords = [10, 10]
                ^
TypeError: Assignment to constant variable.
```

const

prevents

reassignments

Lesson #01:

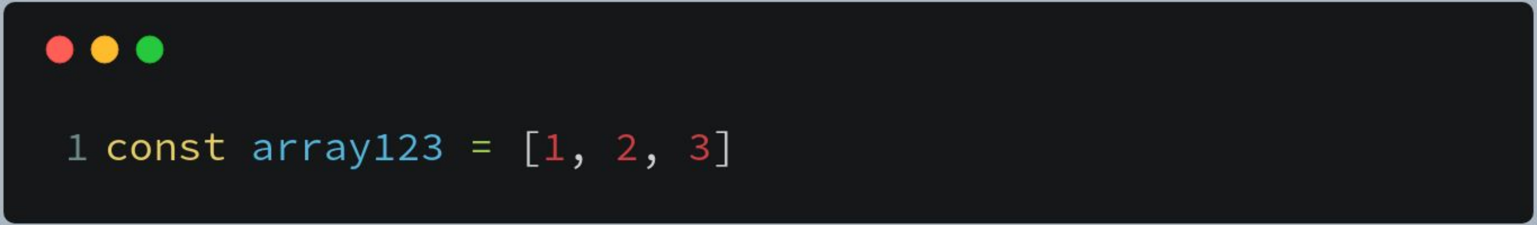
use **const** instead **var** or **let** reassignment

problem:

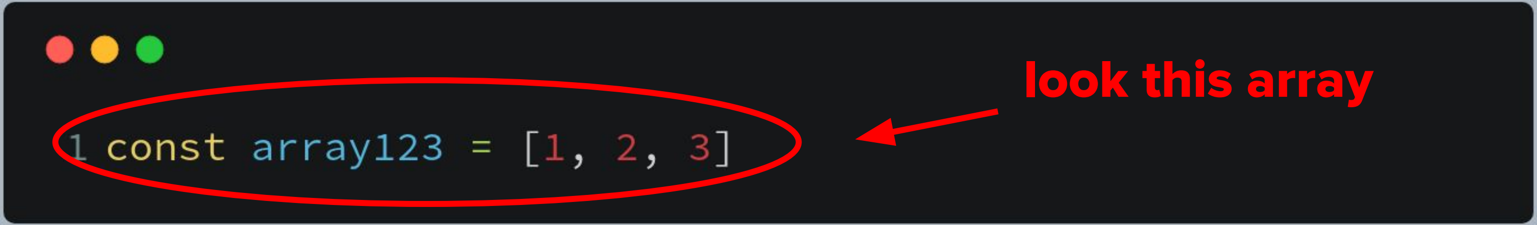
solved! ✅

Let’s **see** another

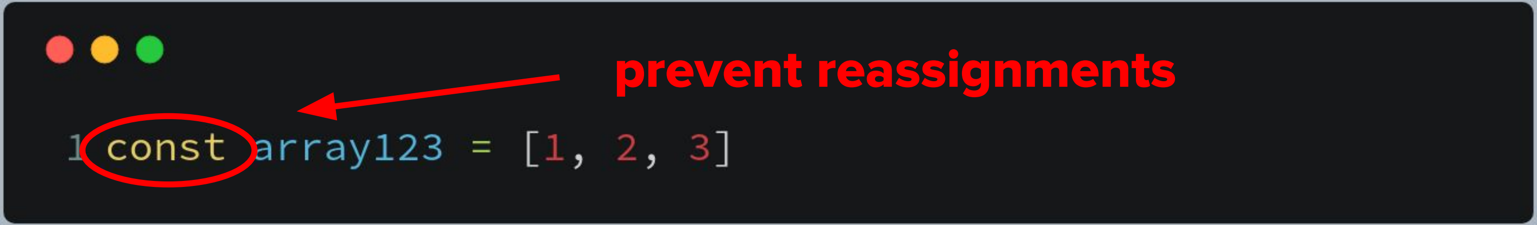
example



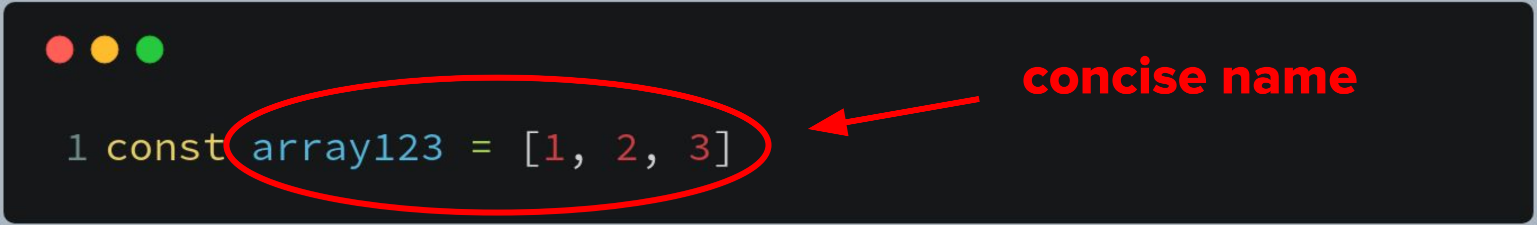
```
const array123 = [1, 2, 3]
```



```
const array123 = [1, 2, 3]
```

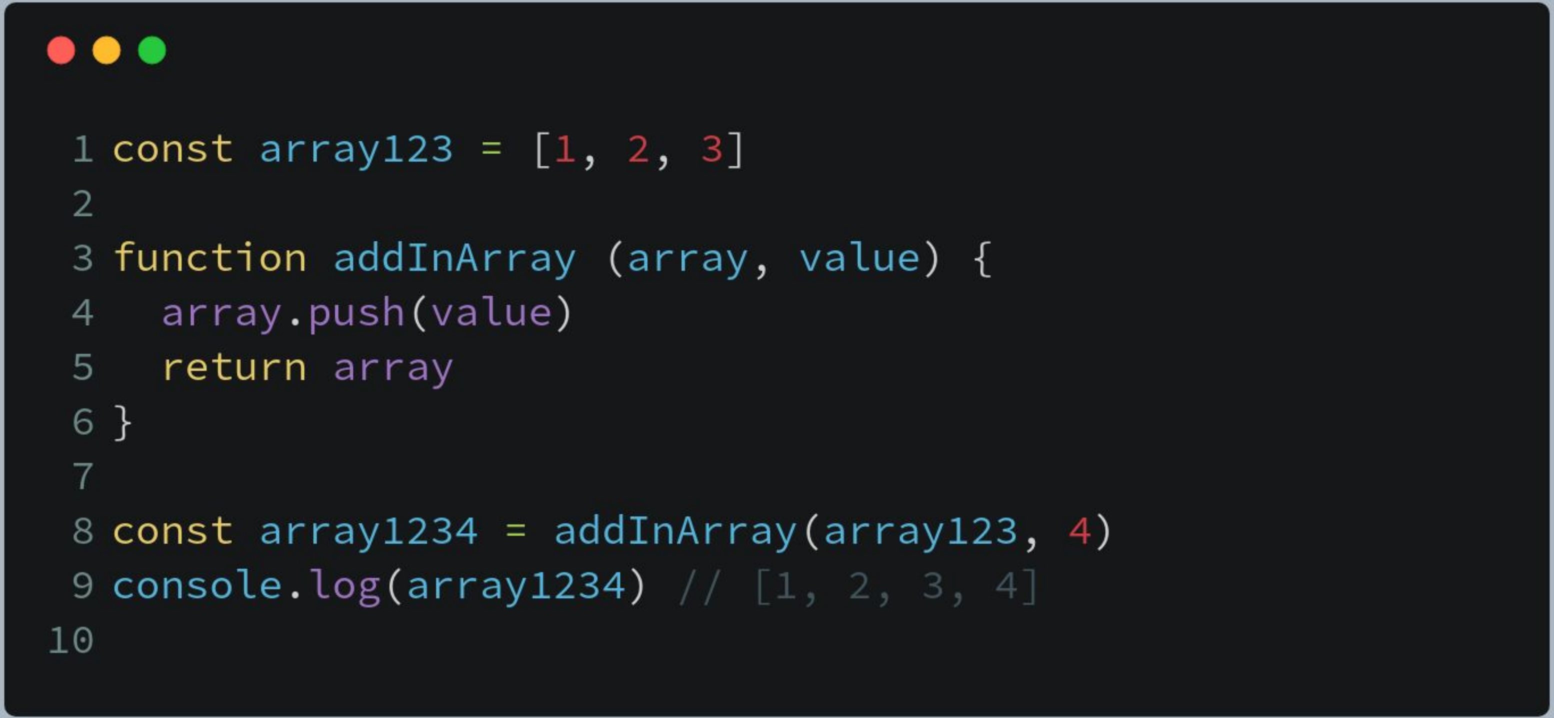


```
const array123 = [1, 2, 3]
```

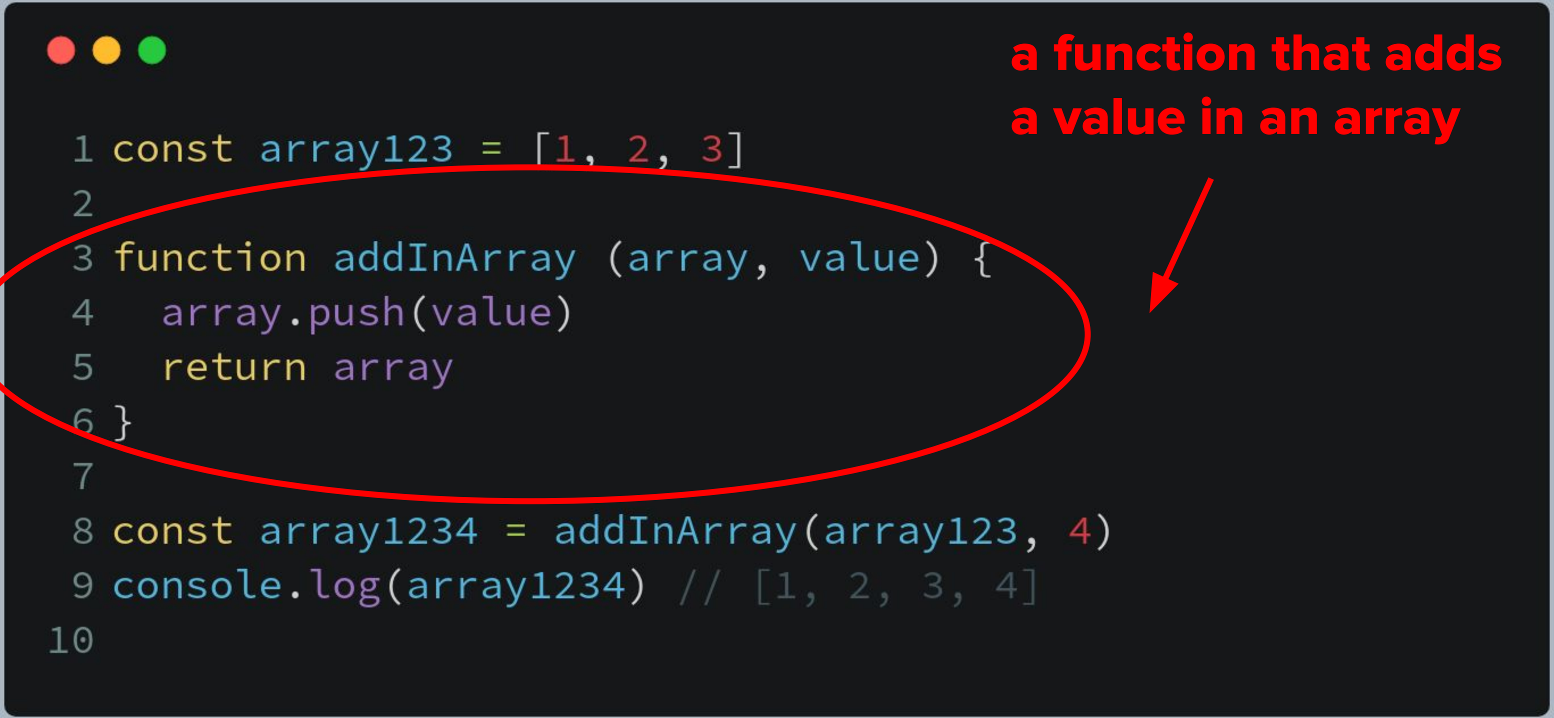


```
const array123 = [1, 2, 3]
```

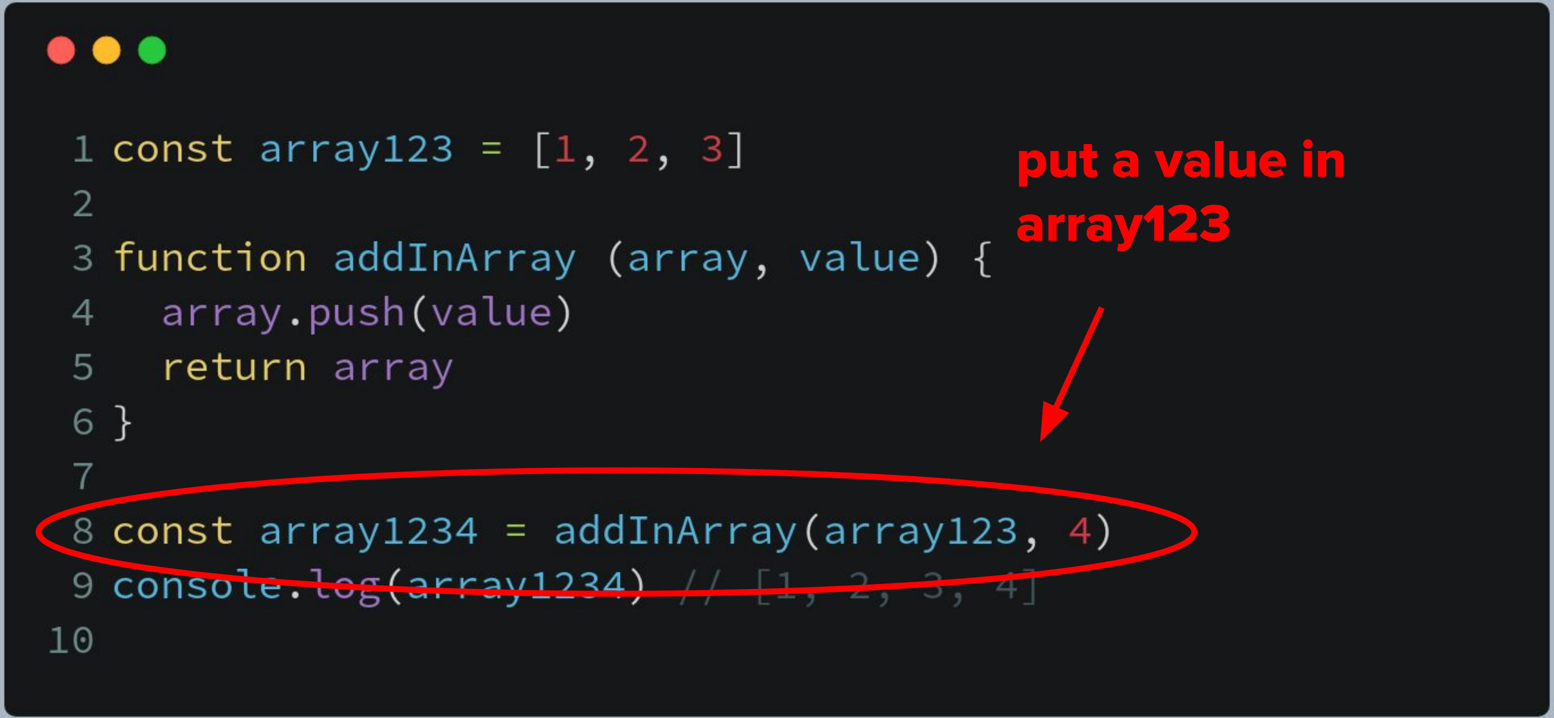
Let’s **do** something with that **array**



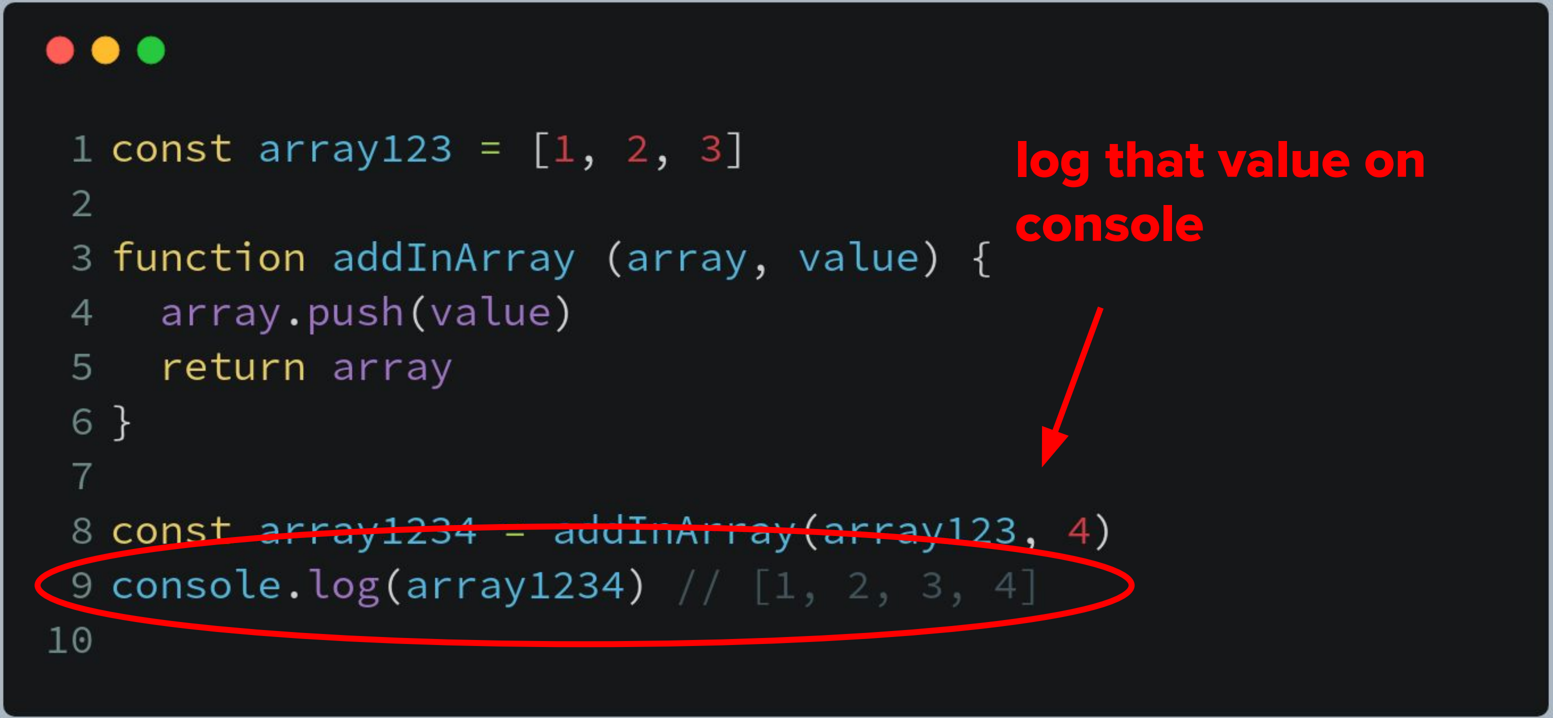
```javascript
const array123 = [1, 2, 3]

function addInArray (array, value) {
  array.push(value)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]

```



```
const array123 = [1, 2, 3]

function addInArray (array, value) {
  array.push(value)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]
```



```
const array123 = [1, 2, 3]

function addInArray (array, value) {
  array.push(value)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]

```



```
const array123 = [1, 2, 3]

function addInArray (array, value) {
  array.push(value)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]

```

Nothing

new

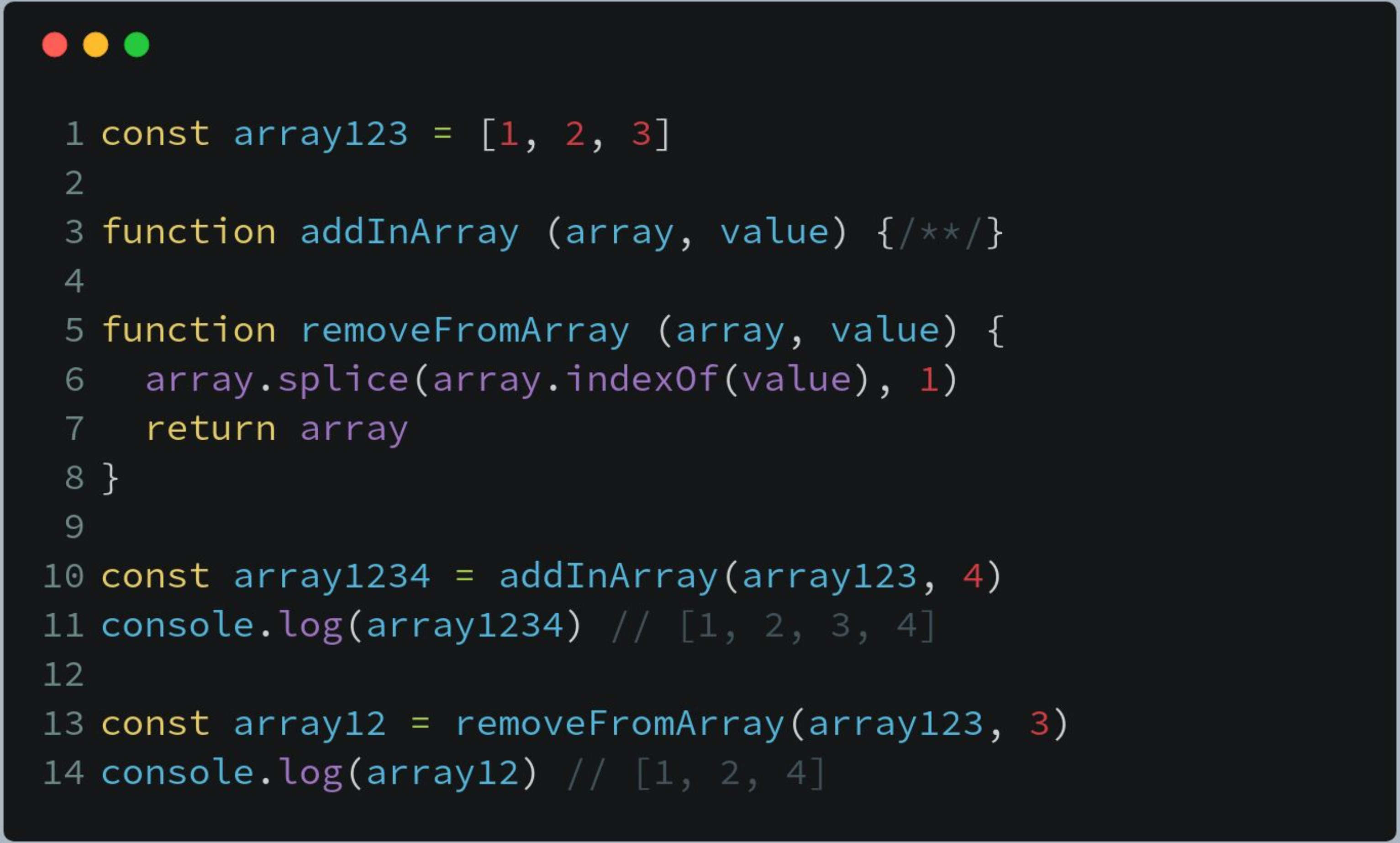
so far

Let's **do**

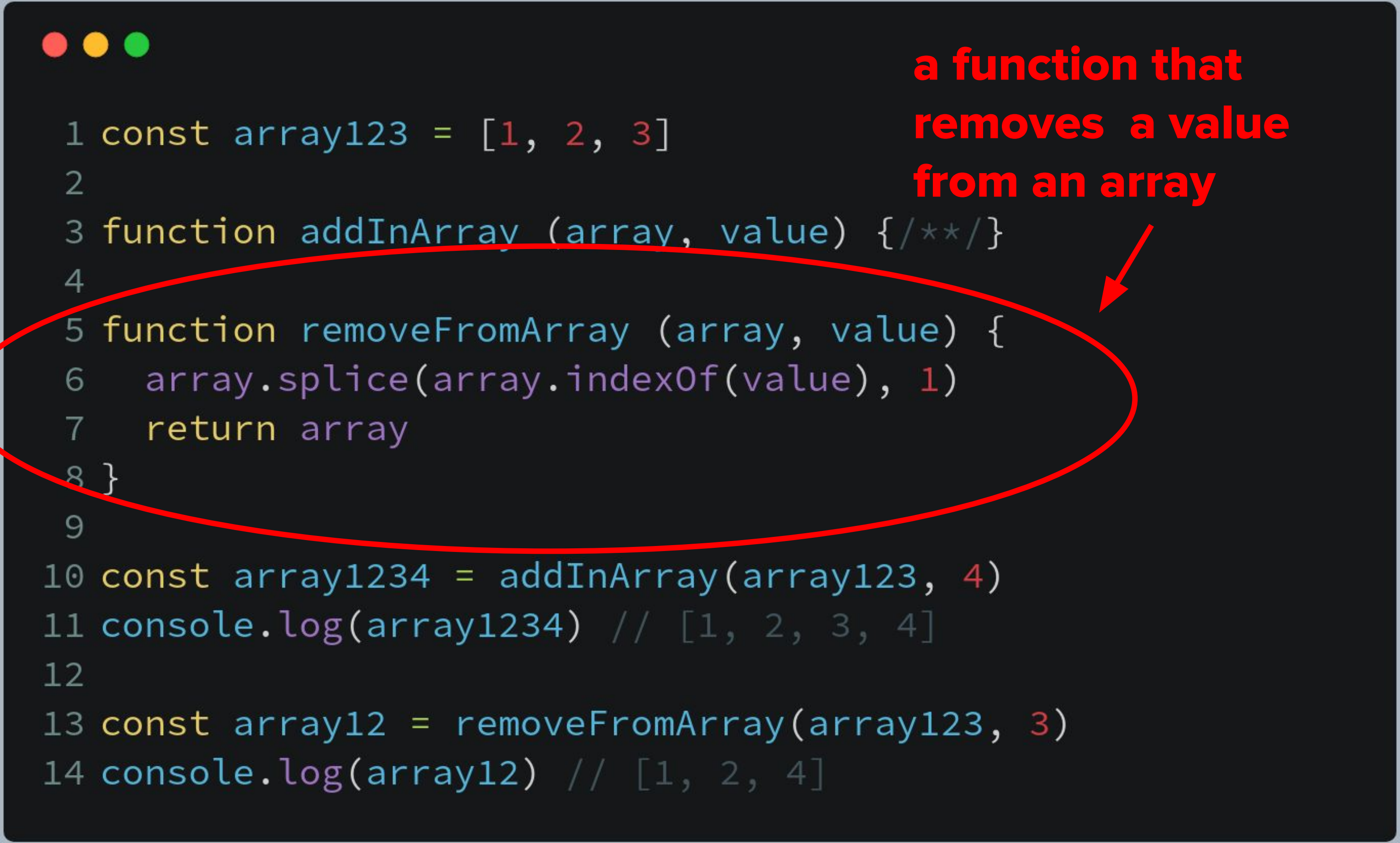
another action

with

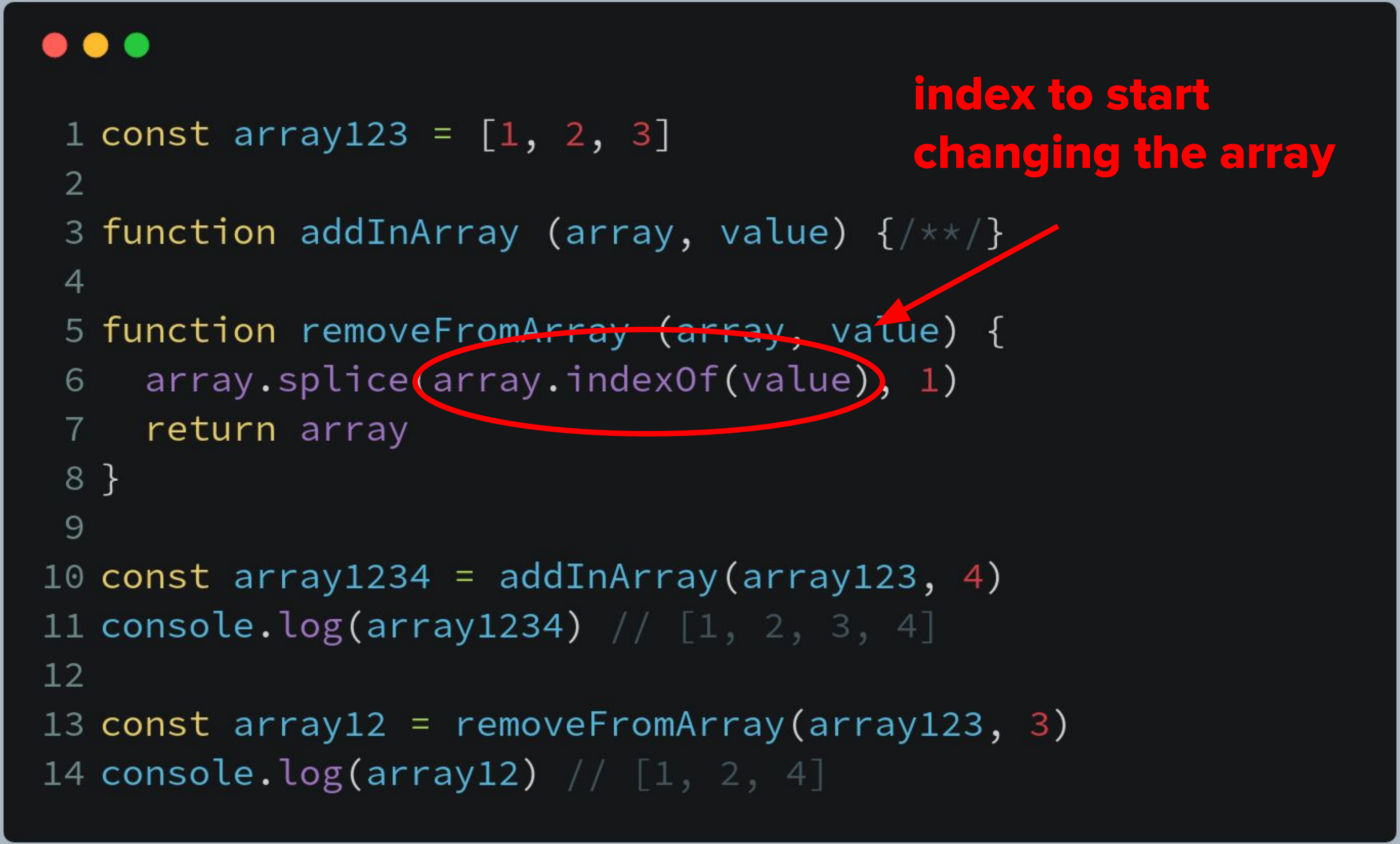
same array



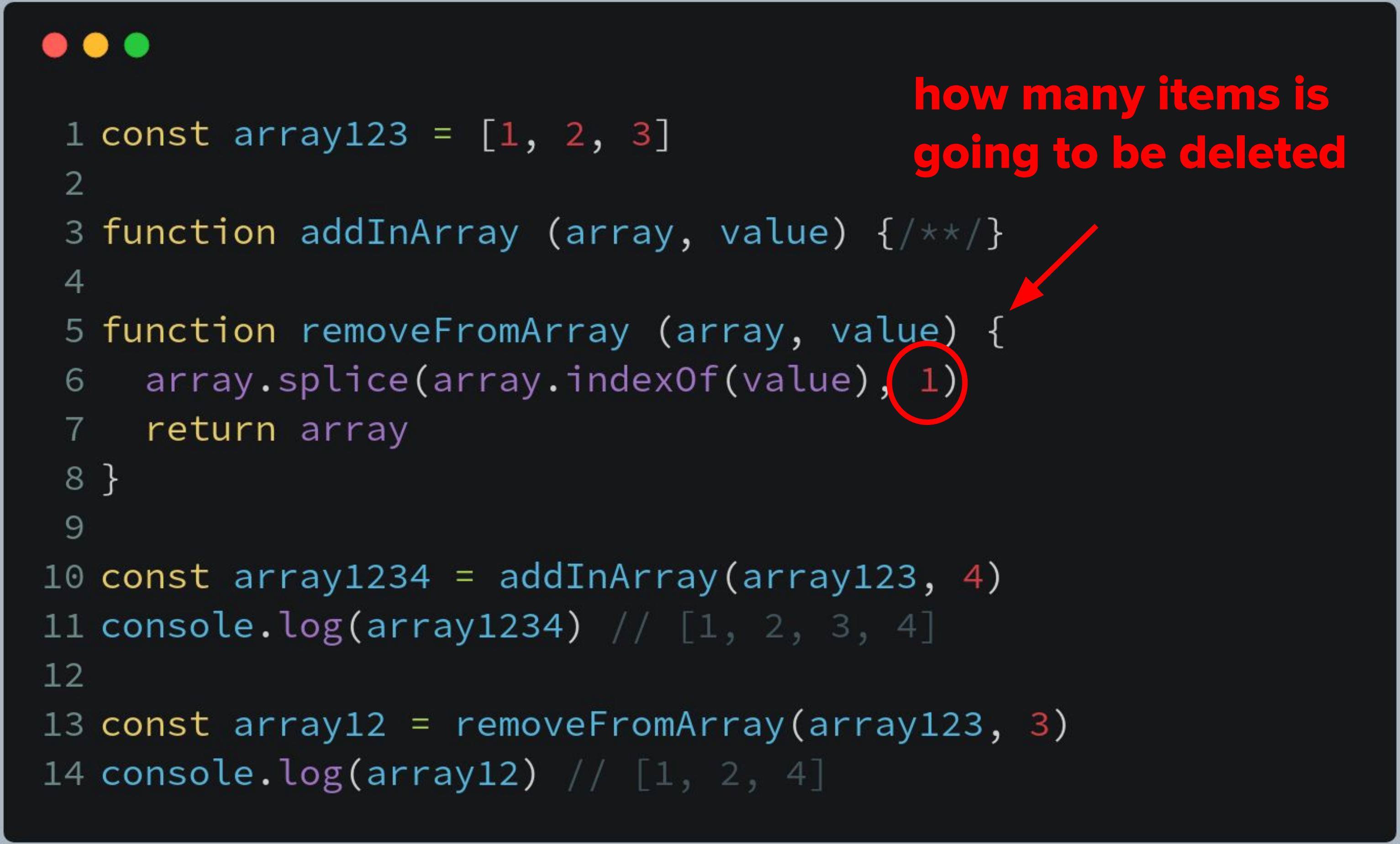
```javascript
const array123 = [1, 2, 3]

function addInArray (array, value) {/**/}

function removeFromArray (array, value) {
  array.splice(array.indexOf(value), 1)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]

const array12 = removeFromArray(array123, 3)
console.log(array12) // [1, 2, 4]
```



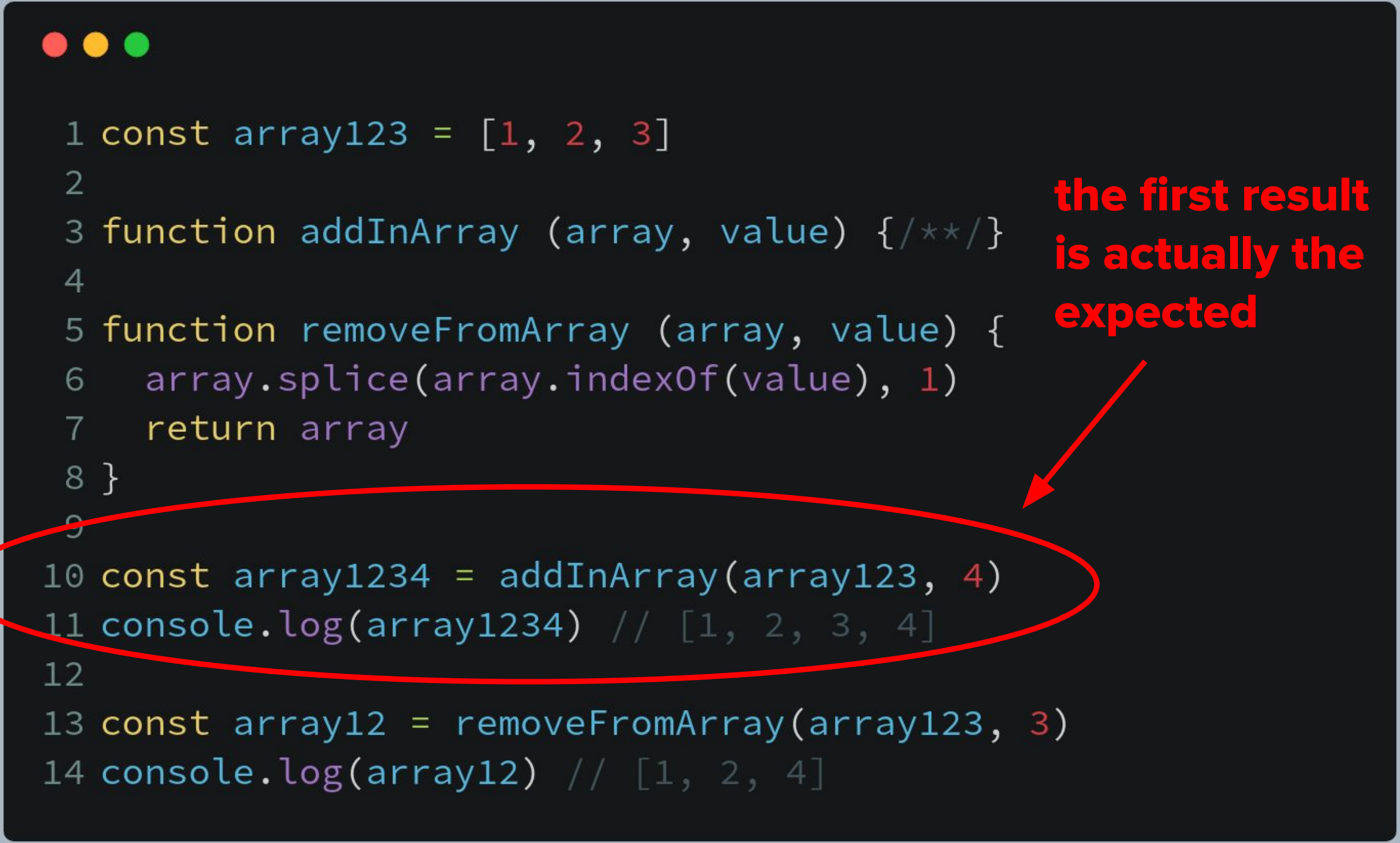
```
const array123 = [1, 2, 3]

function addInArray (array, value) {/**/}

function removeFromArray (array, value) {
  array.splice(array.indexOf(value), 1)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]

const array12 = removeFromArray(array123, 3)
console.log(array12) // [1, 2, 4]
```



```
const array123 = [1, 2, 3]

function addInArray (array, value) {/**/}

function removeFromArray (array, value) {
  array.splice(array.indexOf(value), 1)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]

const array12 = removeFromArray(array123, 3)
console.log(array12) // [1, 2, 4]
```

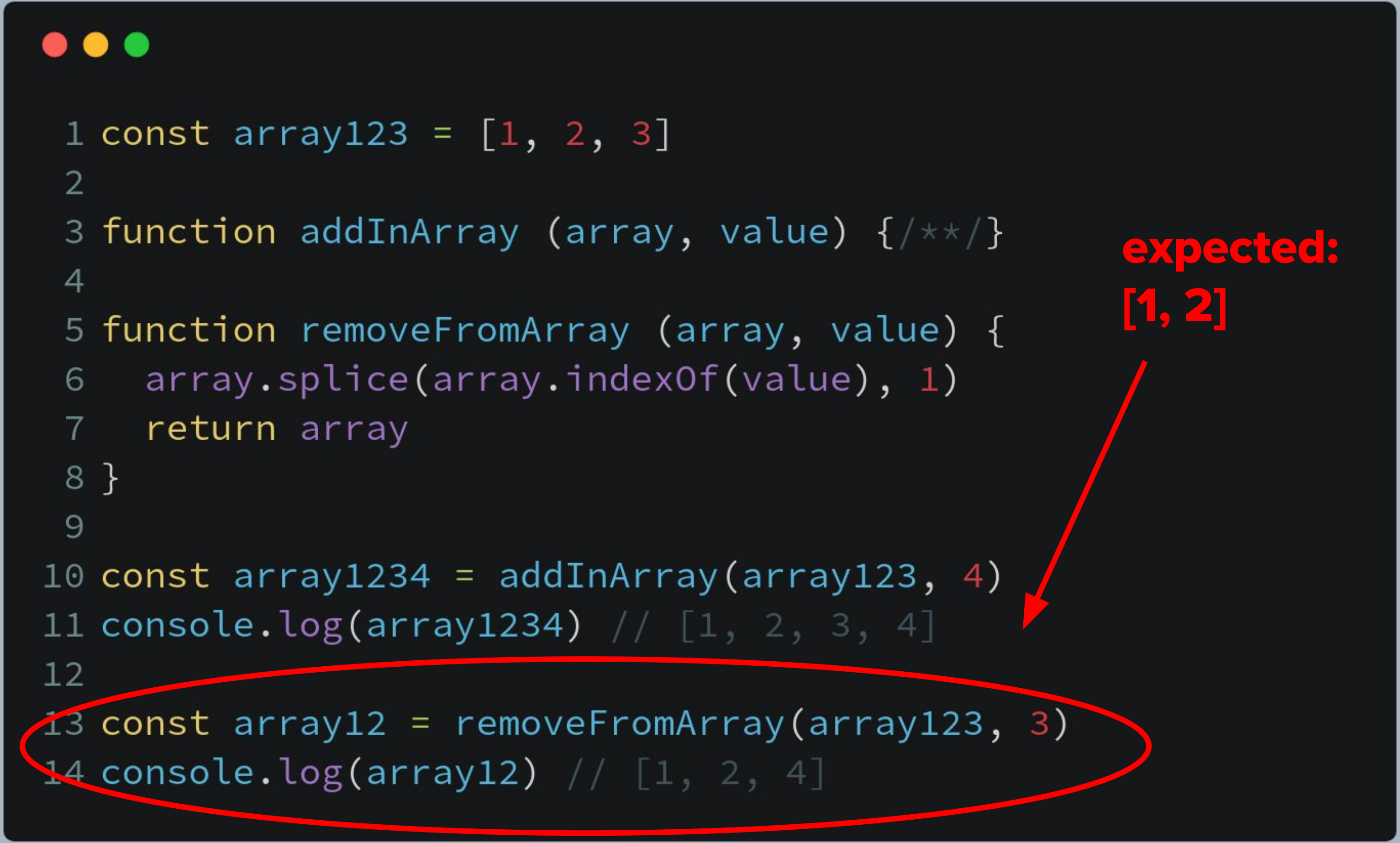


```
const array123 = [1, 2, 3]

function addInArray (array, value) {/**/}

function removeFromArray (array, value) {
  array.splice(array.indexOf(value), 1)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]

const array12 = removeFromArray(array123, 3)
console.log(array12) // [1, 2, 4]
```

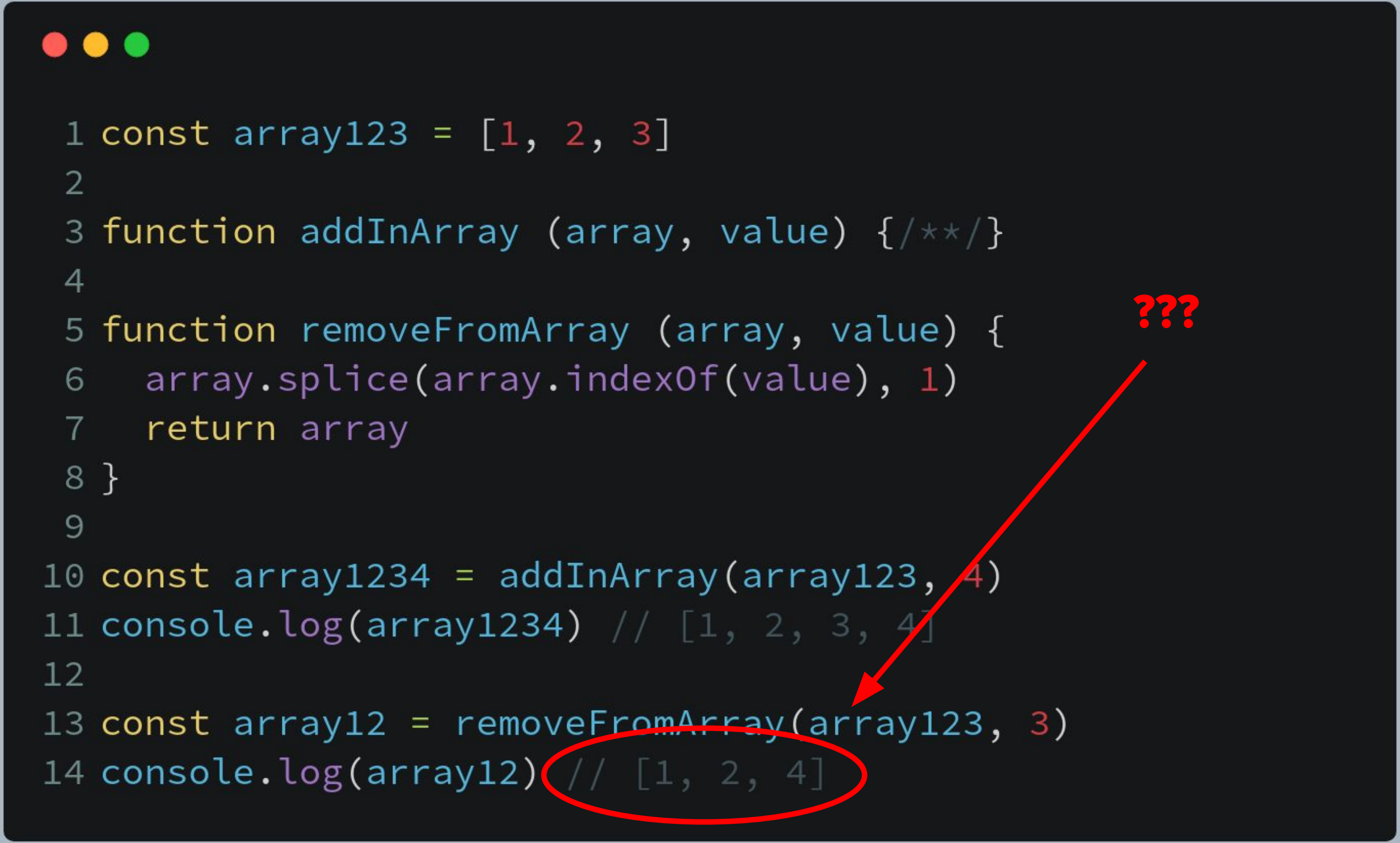


```
const array123 = [1, 2, 3]

function addInArray (array, value) {/**/}

function removeFromArray (array, value) {
  array.splice(array.indexOf(value), 1)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]

const array12 = removeFromArray(array123, 3)
console.log(array12) // [1, 2, 4]
```

the first result is actually the expected

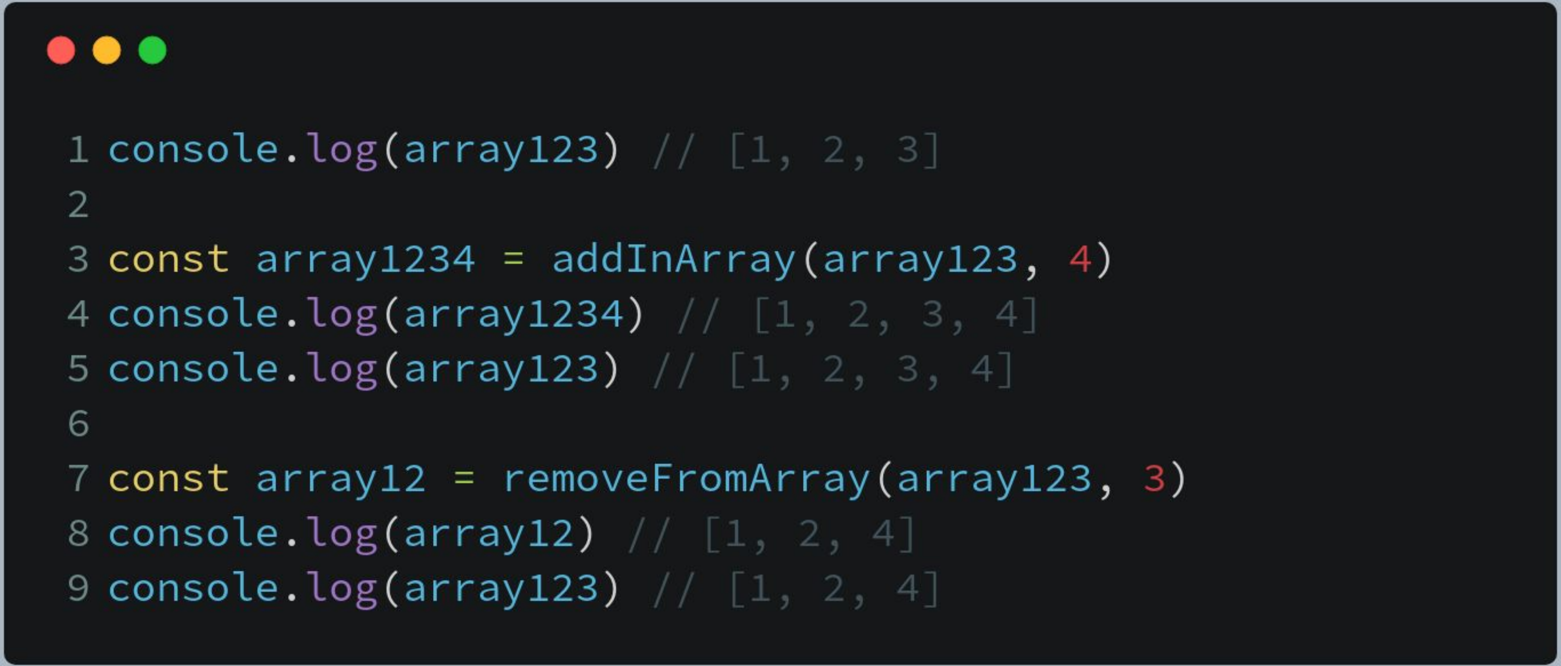


```
const array123 = [1, 2, 3]

function addInArray (array, value) {/**/}

function removeFromArray (array, value) {
  array.splice(array.indexOf(value), 1)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]

const array12 = removeFromArray(array123, 3)
console.log(array12) // [1, 2, 4]
```

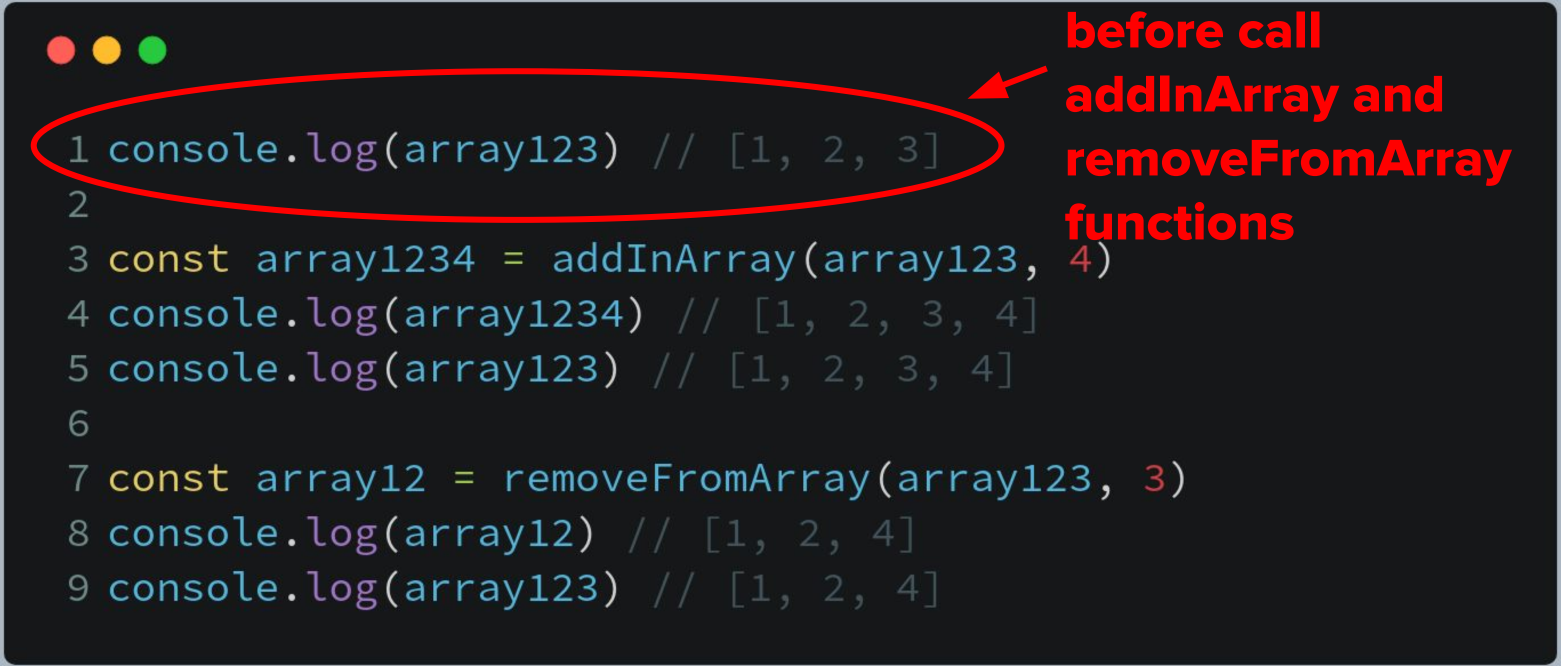


```
const array123 = [1, 2, 3]

function addInArray (array, value) {/**/}

function removeFromArray (array, value) {
  array.splice(array.indexOf(value), 1)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]

const array12 = removeFromArray(array123, 3)
console.log(array12) // [1, 2, 4]
```

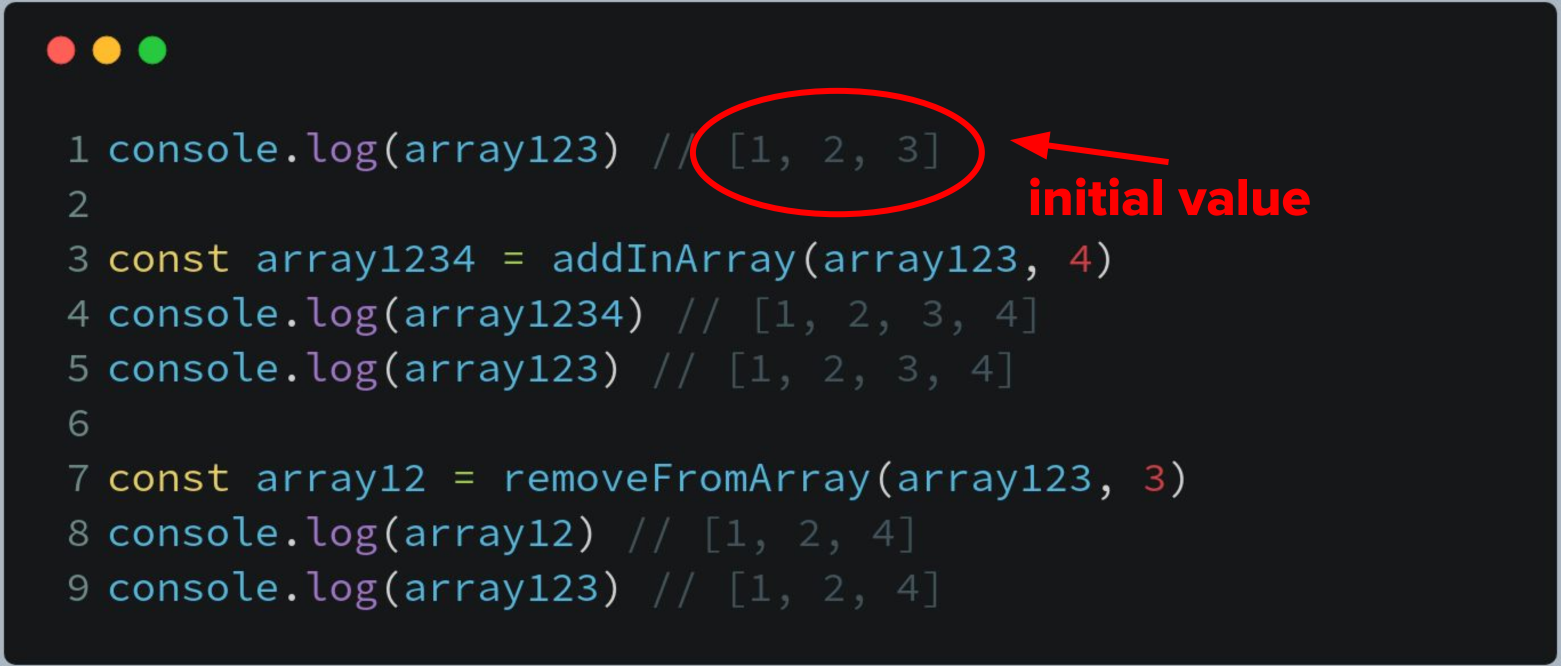
What happened?



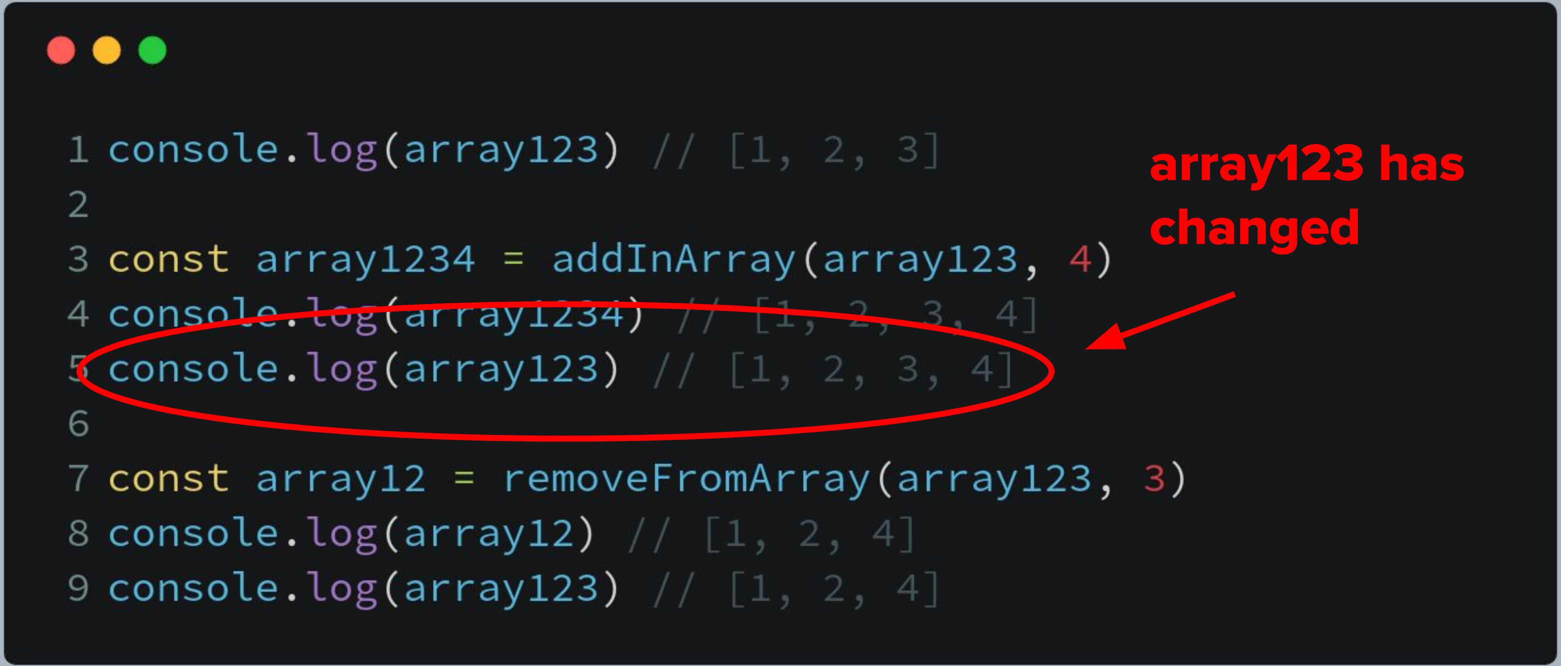
```javascript
console.log(array123) // [1, 2, 3]

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]
console.log(array123) // [1, 2, 3, 4]

const array12 = removeFromArray(array123, 3)
console.log(array12) // [1, 2, 4]
console.log(array123) // [1, 2, 4]
```



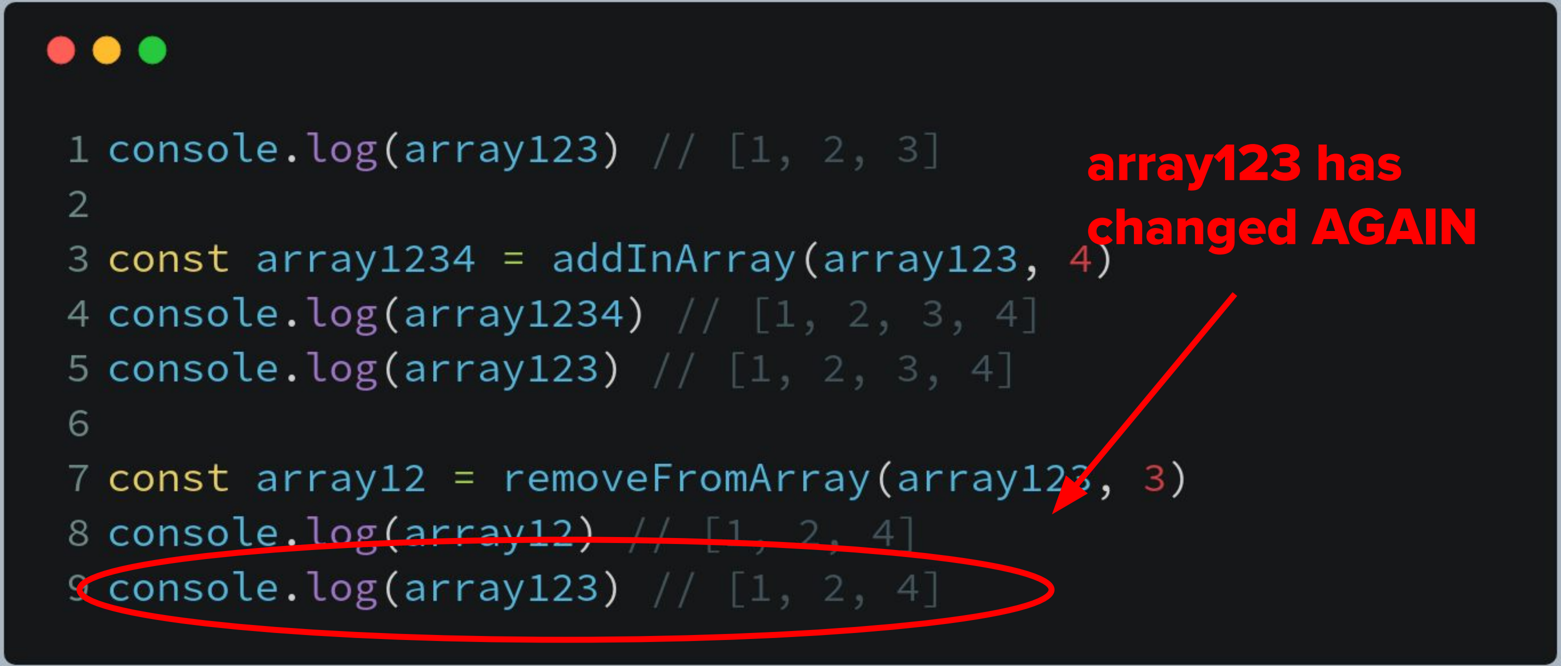
```
console.log(array123) // [1, 2, 3]

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]
console.log(array123) // [1, 2, 3, 4]

const array12 = removeFromArray(array123, 3)
console.log(array12) // [1, 2, 4]
console.log(array123) // [1, 2, 4]
```



```javascript
console.log(array123) // [1, 2, 3]

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]
console.log(array123) // [1, 2, 3, 4]

const array12 = removeFromArray(array123, 3)
console.log(array12) // [1, 2, 4]
console.log(array123) // [1, 2, 4]
```



```
console.log(array123) // [1, 2, 3]

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]
console.log(array123) // [1, 2, 3, 4]

const array12 = removeFromArray(array123, 3)
console.log(array12) // [1, 2, 4]
console.log(array123) // [1, 2, 4]
```



```
console.log(array123) // [1, 2, 3]

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]
console.log(array123) // [1, 2, 3, 4]

const array12 = removeFromArray(array123, 3)
console.log(array12) // [1, 2, 4]
console.log(array123) // [1, 2, 4]
```

Why is that happened?

For a

complete

understanding...

Let's **talk** about

objects

But

I do NOT

mean { }

I mean

object

data type

JavaScript has two groups of data types:

Primitives

and

objects

Primitives are:

String

Number

Boolean

Undefined

Null

Symbol (ES6+)

Objects are all others:

Object

Array

Function

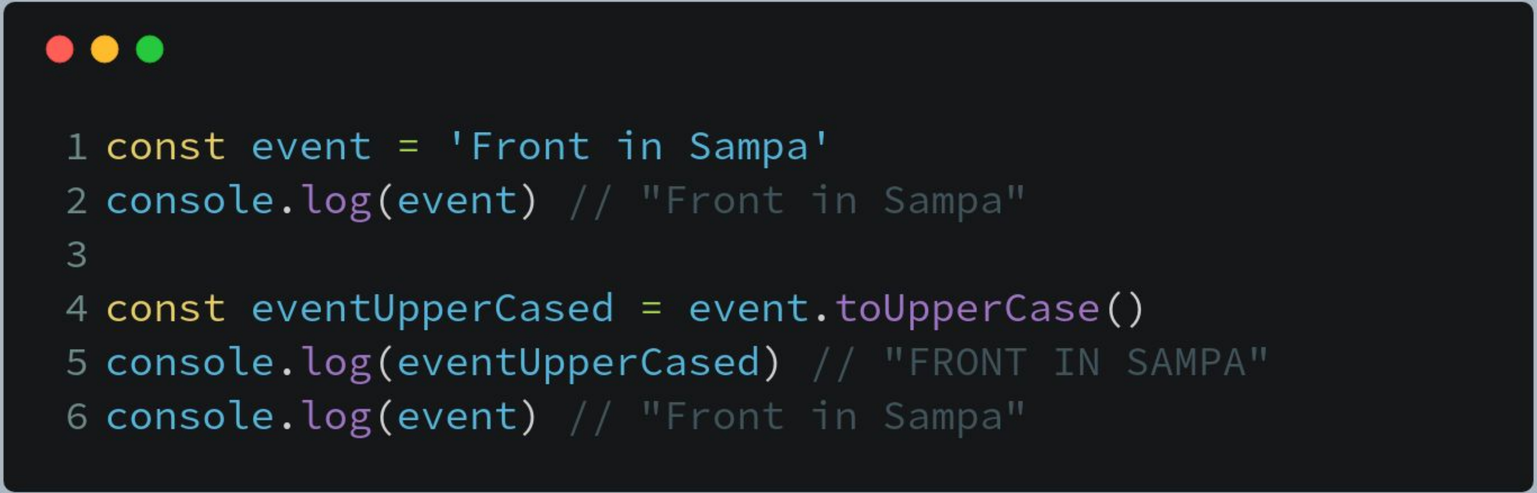
RegExp

etc...

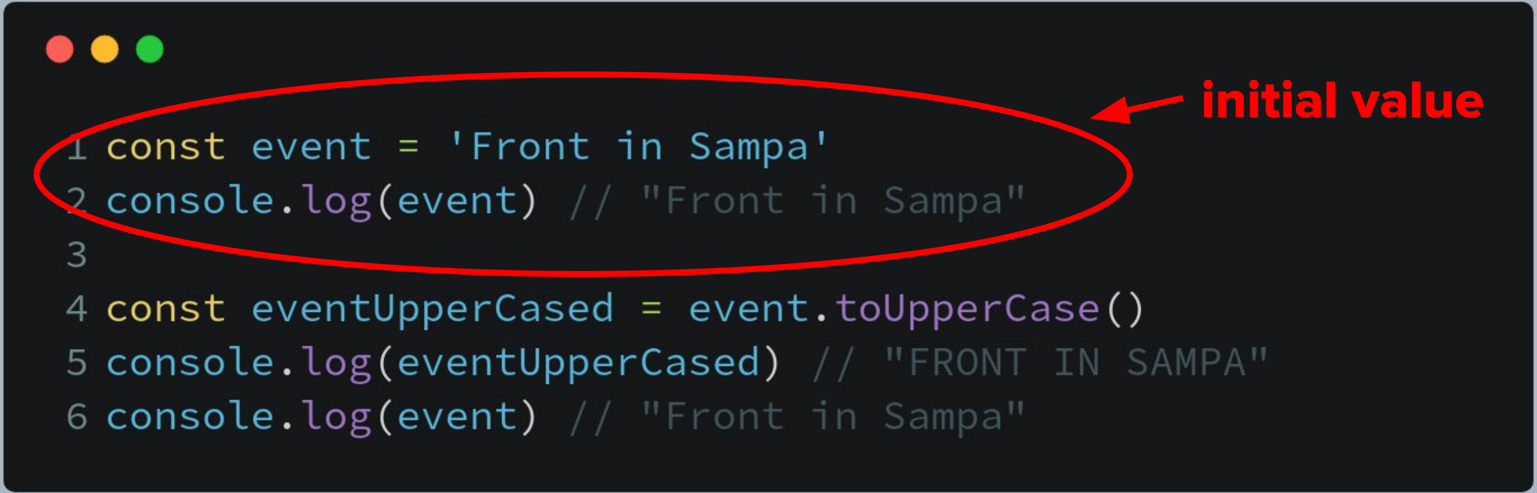
Primitive values are immutable

Objects are always mutable

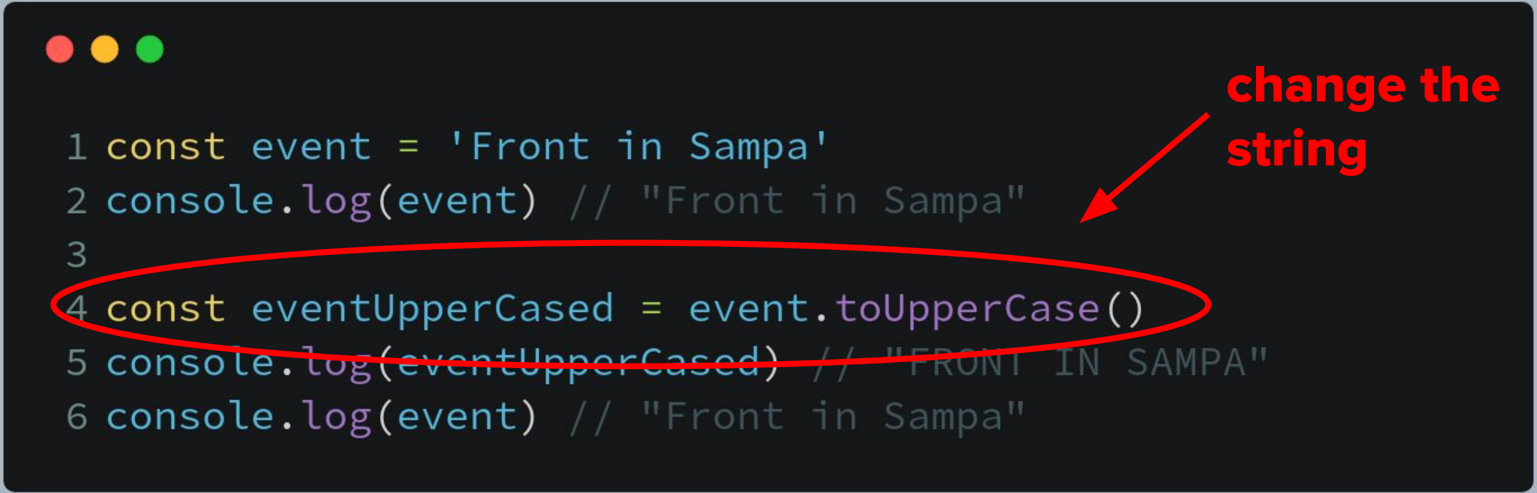
What is that mean?



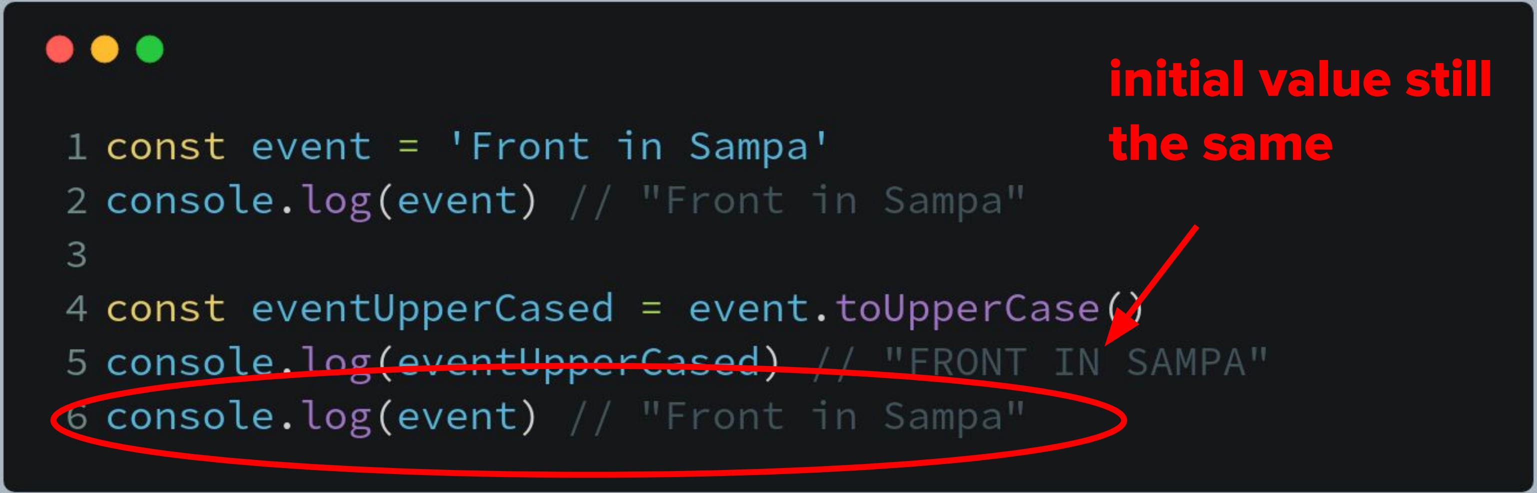
```javascript
const event = 'Front in Sampa'
console.log(event) // "Front in Sampa"

const eventUpperCased = event.toUpperCase()
console.log(eventUpperCased) // "FRONT IN SAMPA"
console.log(event) // "Front in Sampa"
```



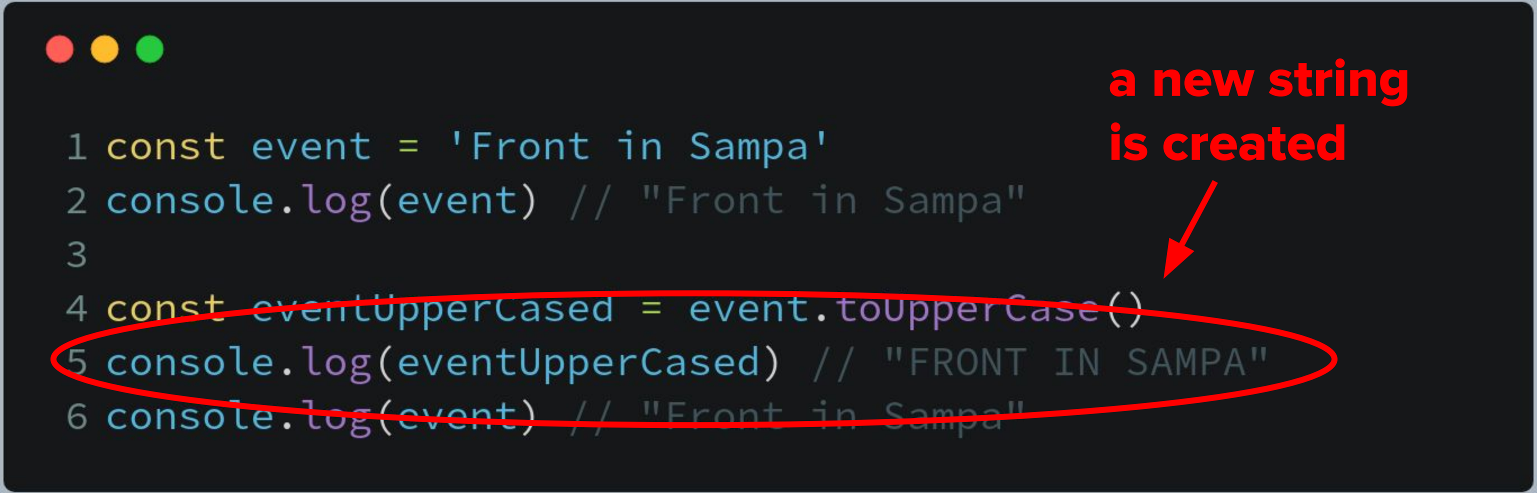
```
const event = 'Front in Sampa'
console.log(event) // "Front in Sampa"

const eventUpperCased = event.toUpperCase()
console.log(eventUpperCased) // "FRONT IN SAMPA"
console.log(event) // "Front in Sampa"
```



```
const event = 'Front in Sampa'
console.log(event) // "Front in Sampa"

const eventUpperCased = event.toUpperCase()
console.log(eventUpperCased) // "FRONT IN SAMPA"
console.log(event) // "Front in Sampa"
```



```
const event = 'Front in Sampa'
console.log(event) // "Front in Sampa"

const eventUpperCased = event.toUpperCase()
console.log(eventUpperCased) // "FRONT IN SAMPA"
console.log(event) // "Front in Sampa"
```



```
const event = 'Front in Sampa'
console.log(event) // "Front in Sampa"

const eventUpperCased = event.toUpperCase()
console.log(eventUpperCased) // "FRONT IN SAMPA"
console.log(event) // "Front in Sampa"
```

That’s

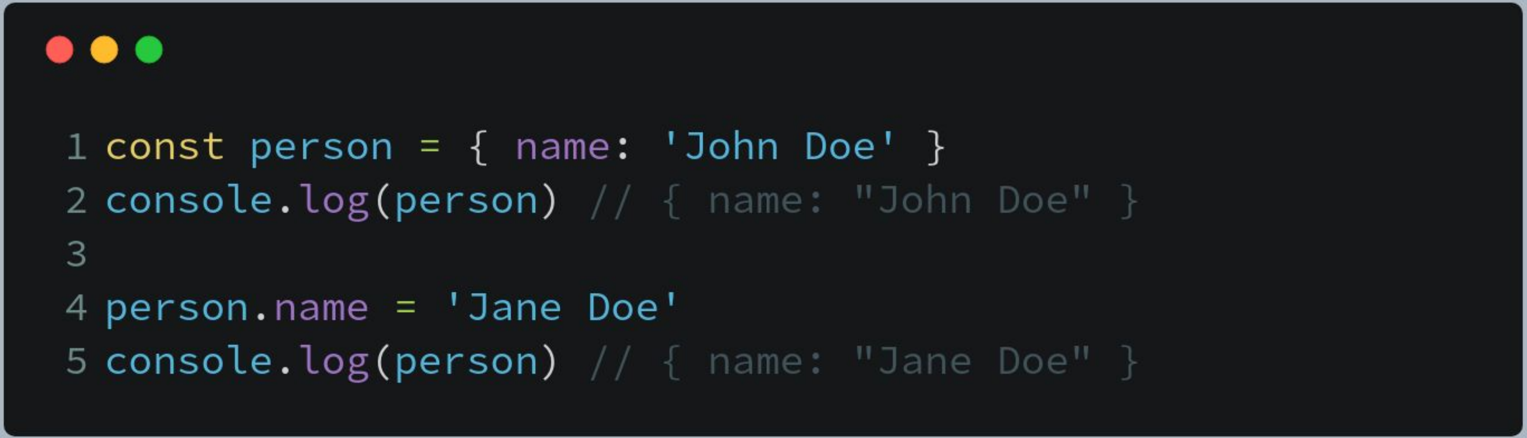
what

Immutability

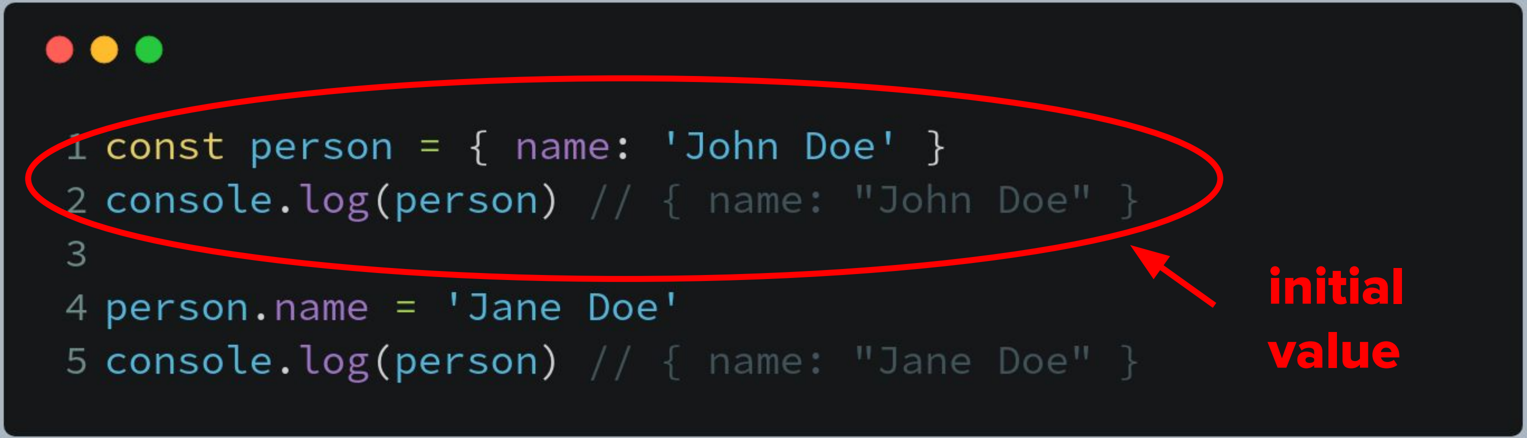
means!

And **what** about

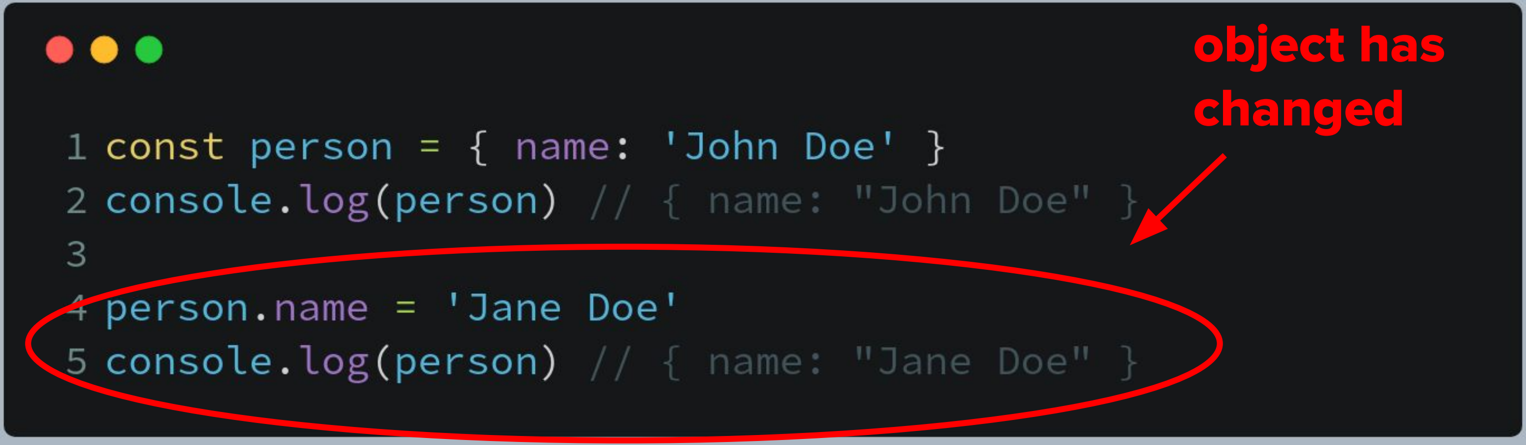
objects?



```javascript
const person = { name: 'John Doe' }
console.log(person) // { name: "John Doe" }

person.name = 'Jane Doe'
console.log(person) // { name: "Jane Doe" }
```



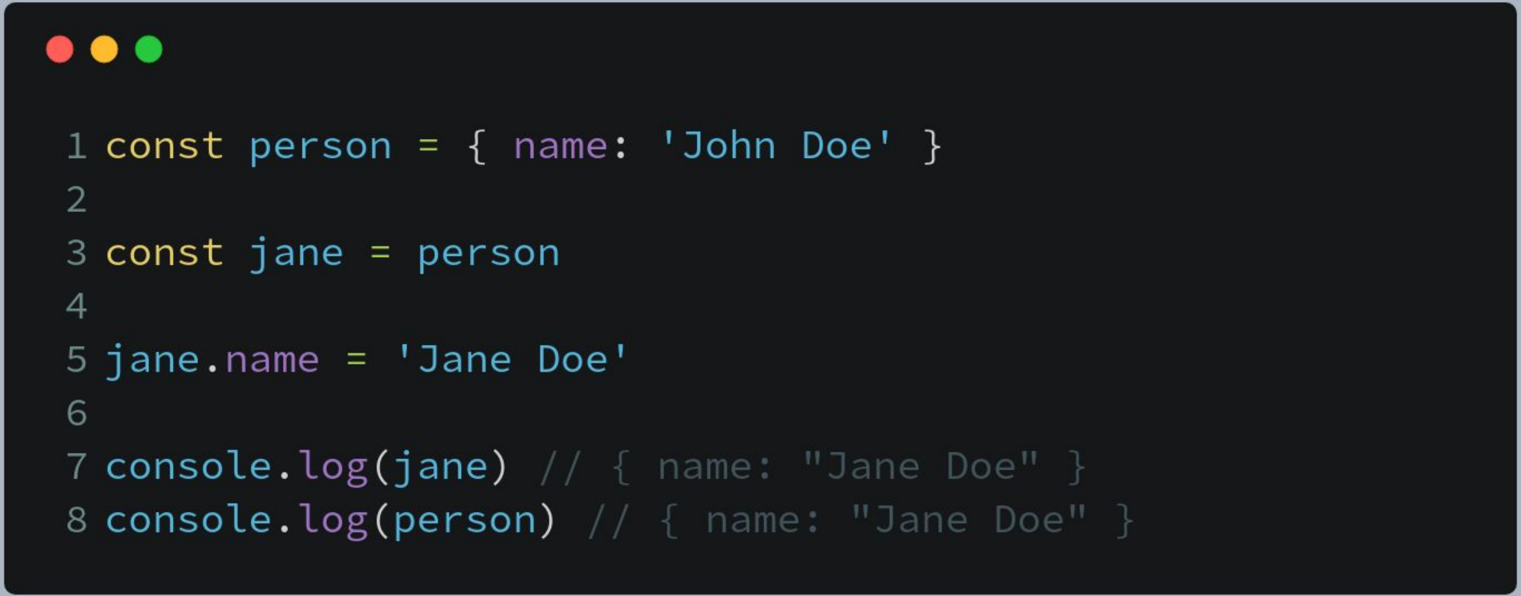
```
const person = { name: 'John Doe' }
console.log(person) // { name: "John Doe" }

person.name = 'Jane Doe'
console.log(person) // { name: "Jane Doe" }
```



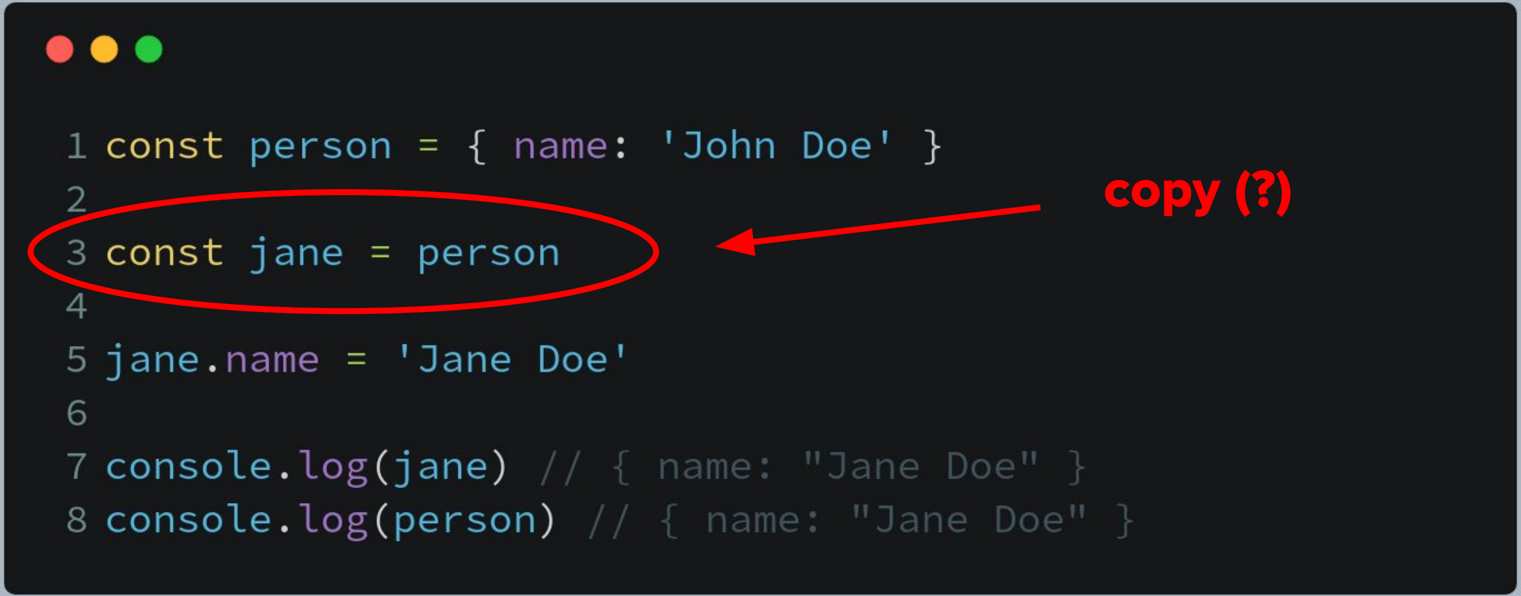
```
const person = { name: 'John Doe' }
console.log(person) // { name: "John Doe" }

person.name = 'Jane Doe'
console.log(person) // { name: "Jane Doe" }
```

Remember:

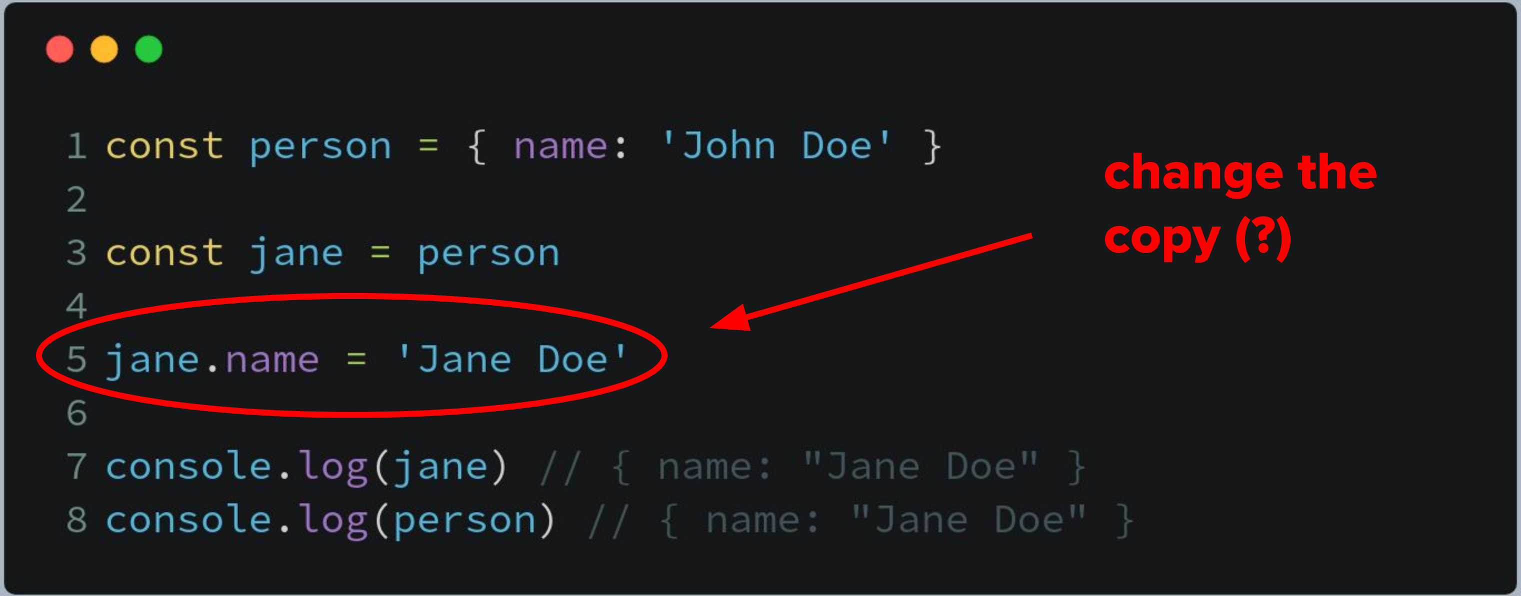
Objects are always mutable



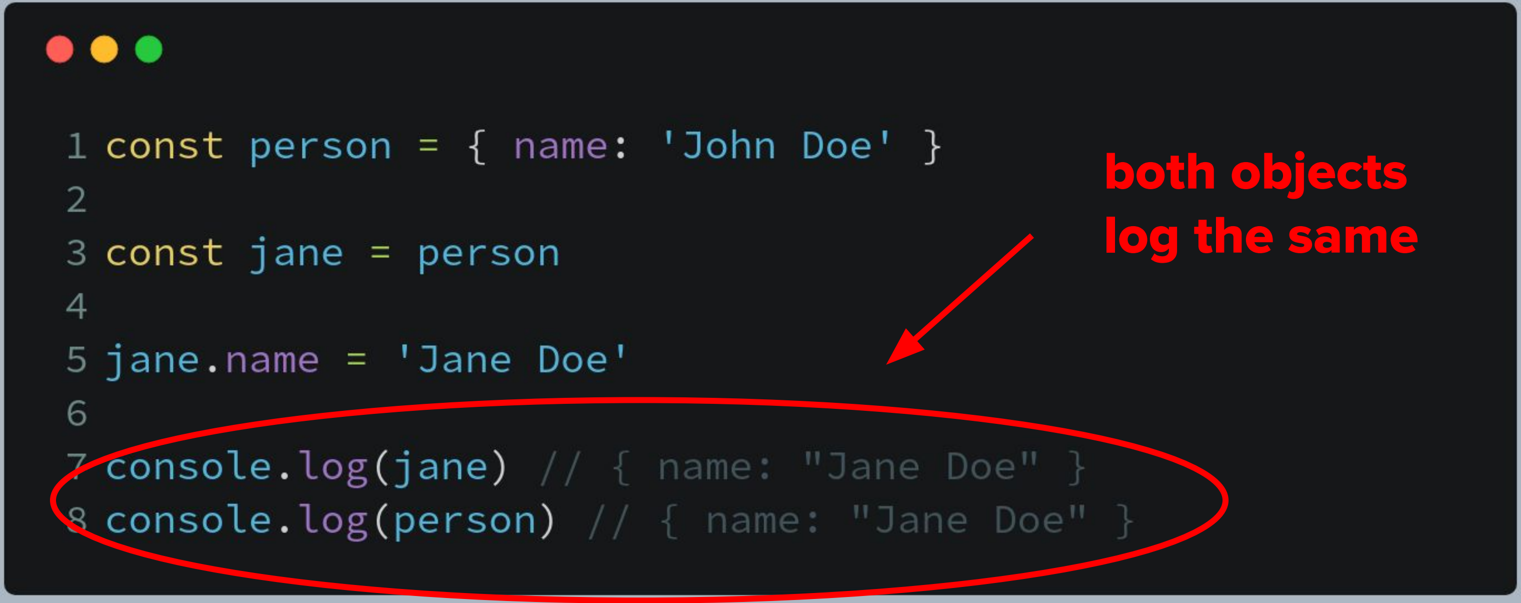
```javascript
const person = { name: 'John Doe' }

const jane = person

jane.name = 'Jane Doe'

console.log(jane) // { name: "Jane Doe" }
console.log(person) // { name: "Jane Doe" }
```



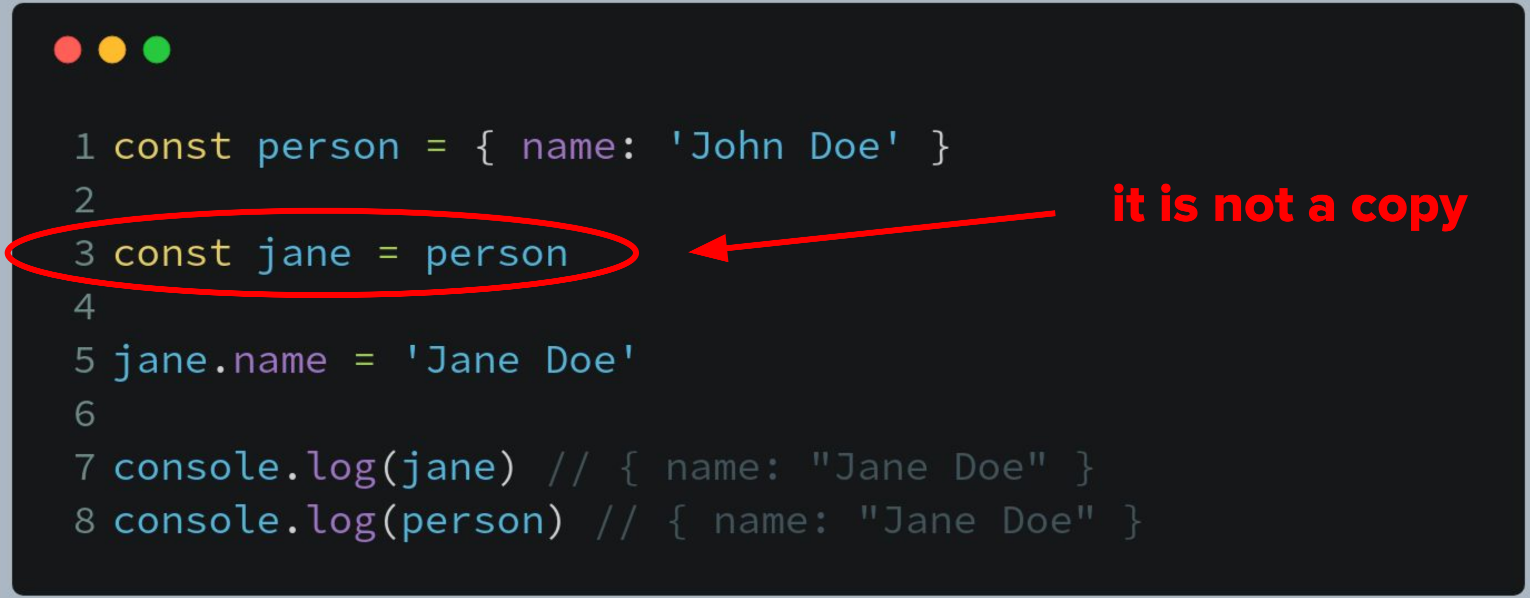
```
const person = { name: 'John Doe' }

const jane = person

jane.name = 'Jane Doe'

console.log(jane) // { name: "Jane Doe" }
console.log(person) // { name: "Jane Doe" }
```



```
const person = { name: 'John Doe' }

const jane = person

jane.name = 'Jane Doe'

console.log(jane) // { name: "Jane Doe" }
console.log(person) // { name: "Jane Doe" }
```



```
const person = { name: 'John Doe' }

const jane = person

jane.name = 'Jane Doe'

console.log(jane) // { name: "Jane Doe" }
console.log(person) // { name: "Jane Doe" }
```



```
const person = { name: 'John Doe' }

const jane = person

jane.name = 'Jane Doe'

console.log(jane) // { name: "Jane Doe" }
console.log(person) // { name: "Jane Doe" }
```

Objects are

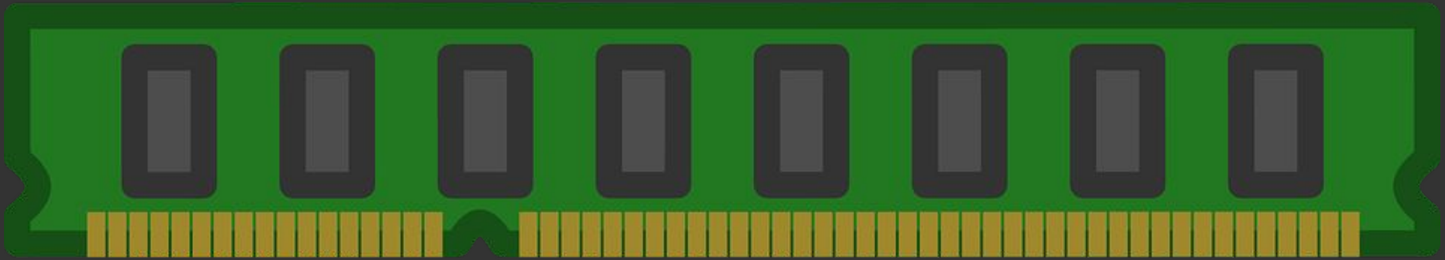
passed

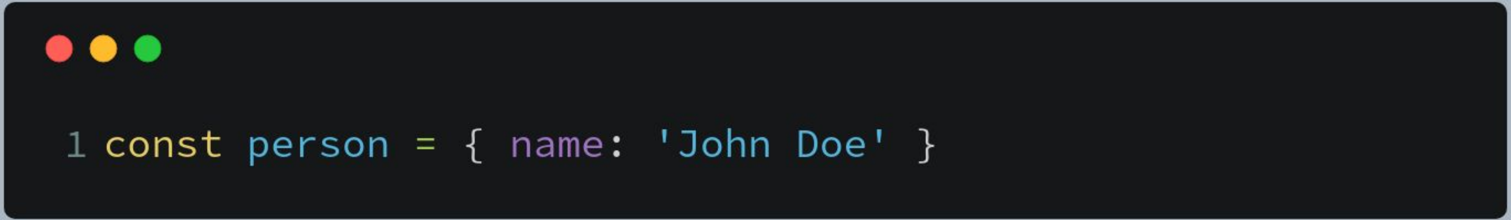
by

reference

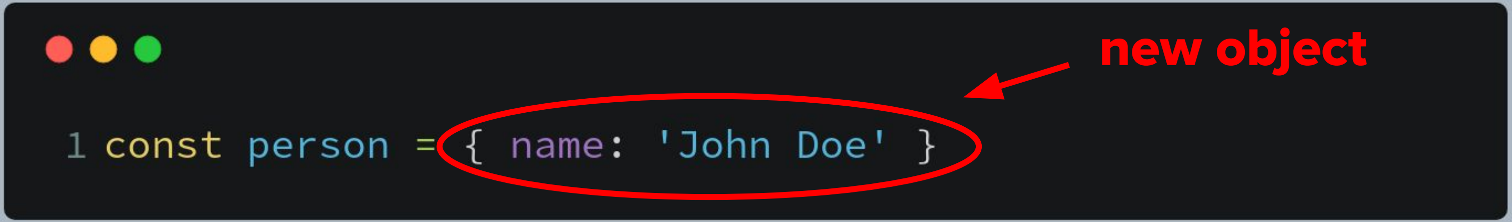
What is that mean?

memory chip

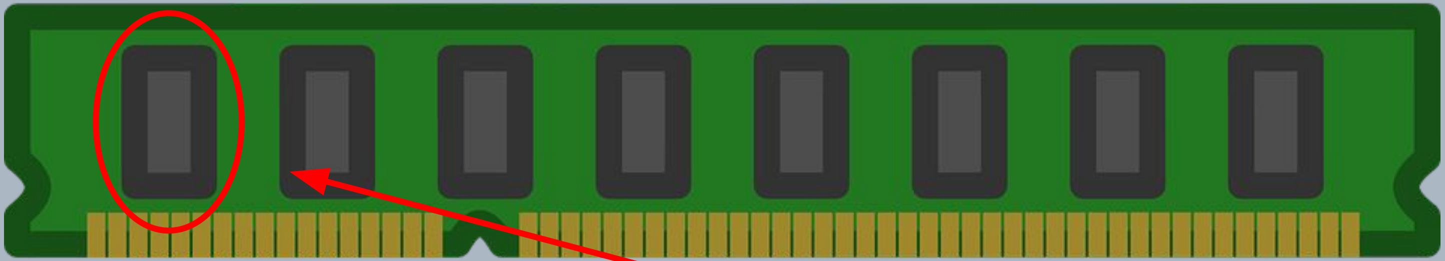


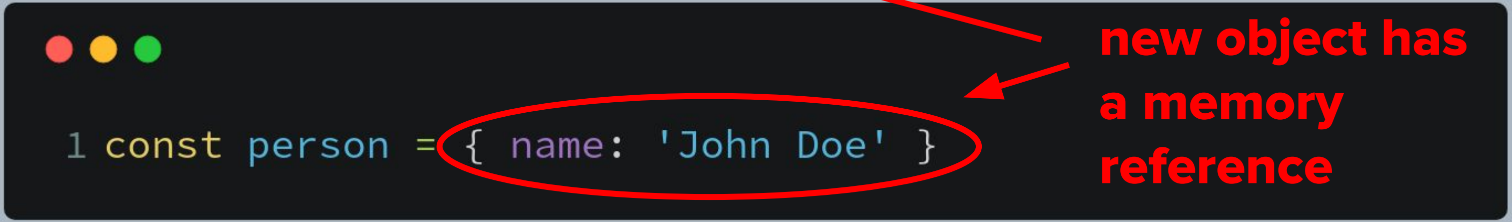


```javascript
const person = { name: 'John Doe' }
```



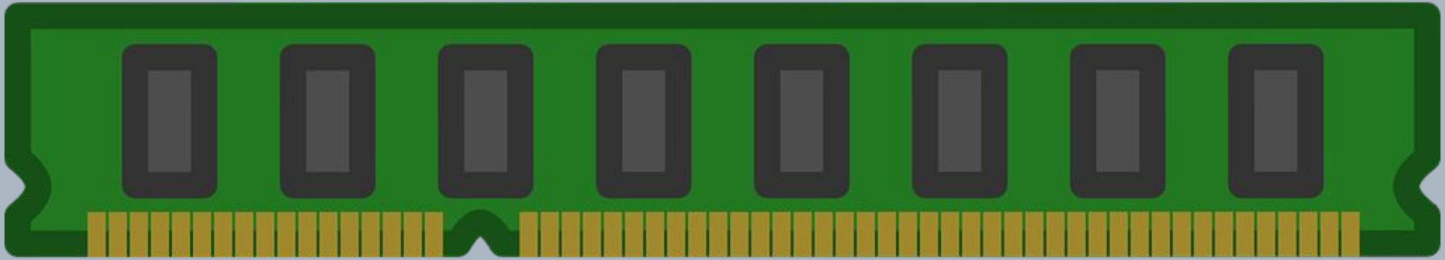
```
const person = { name: 'John Doe' }
```

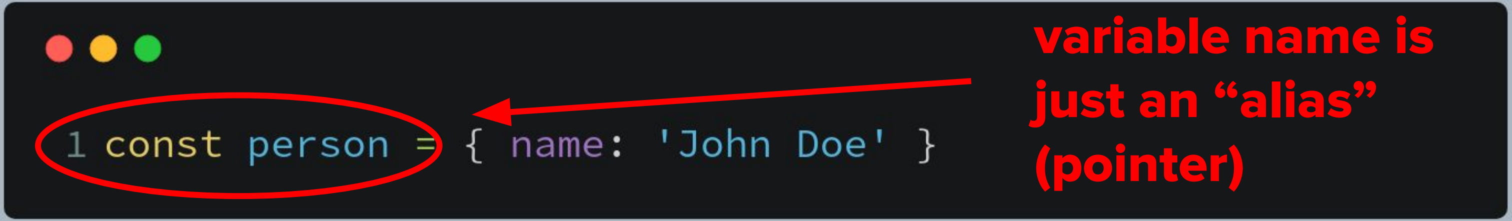




```
const person = { name: 'John Doe' }
```

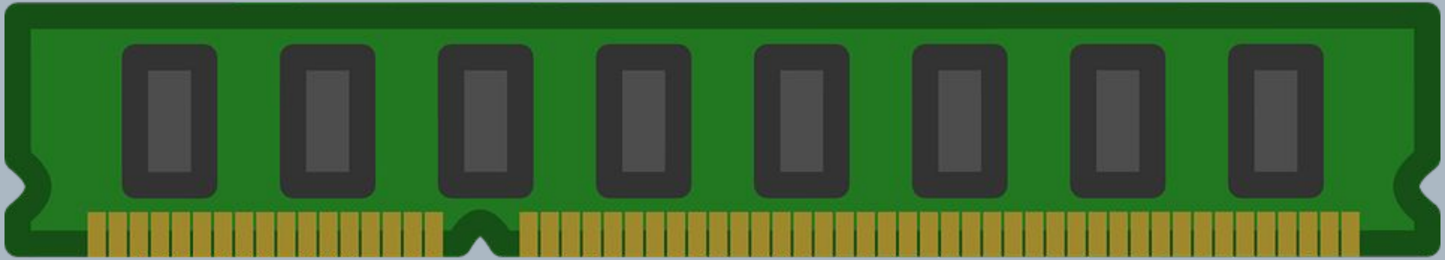
new object has a memory reference

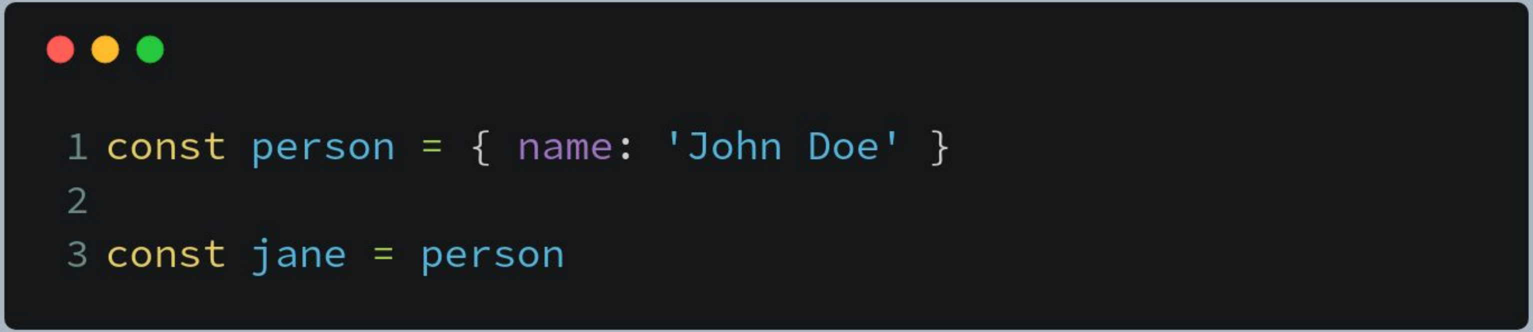




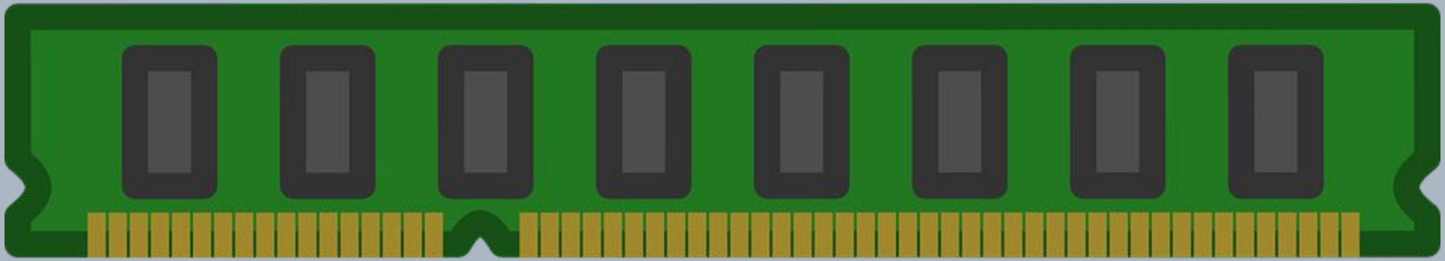
```
const person = { name: 'John Doe' }
```

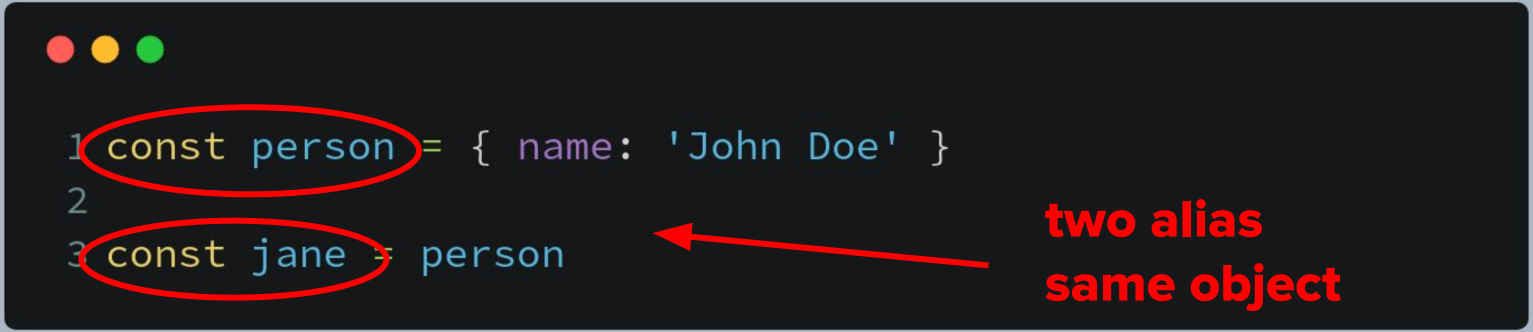
variable name is just an “alias” (pointer)



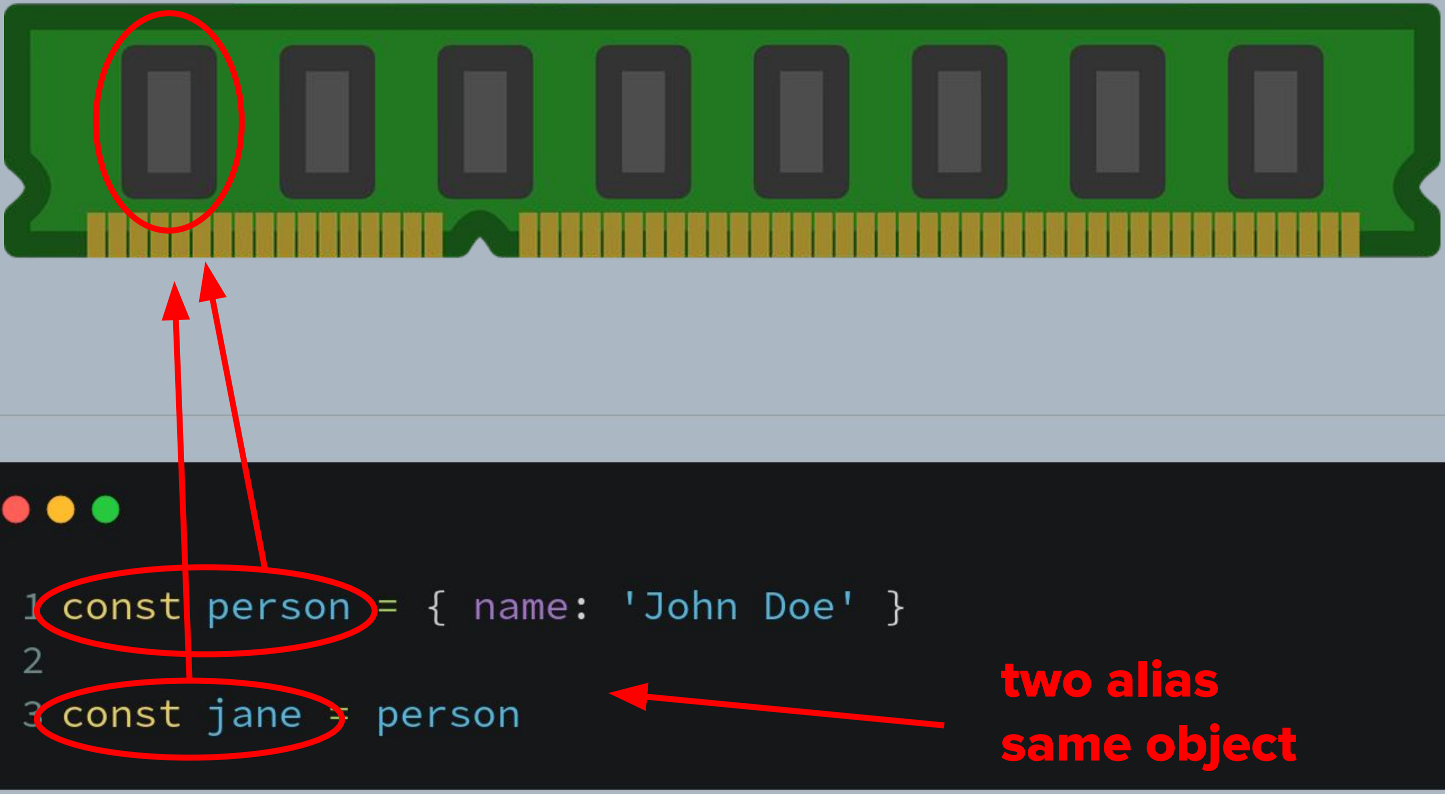


```
const person = { name: 'John Doe' }

const jane = person
```

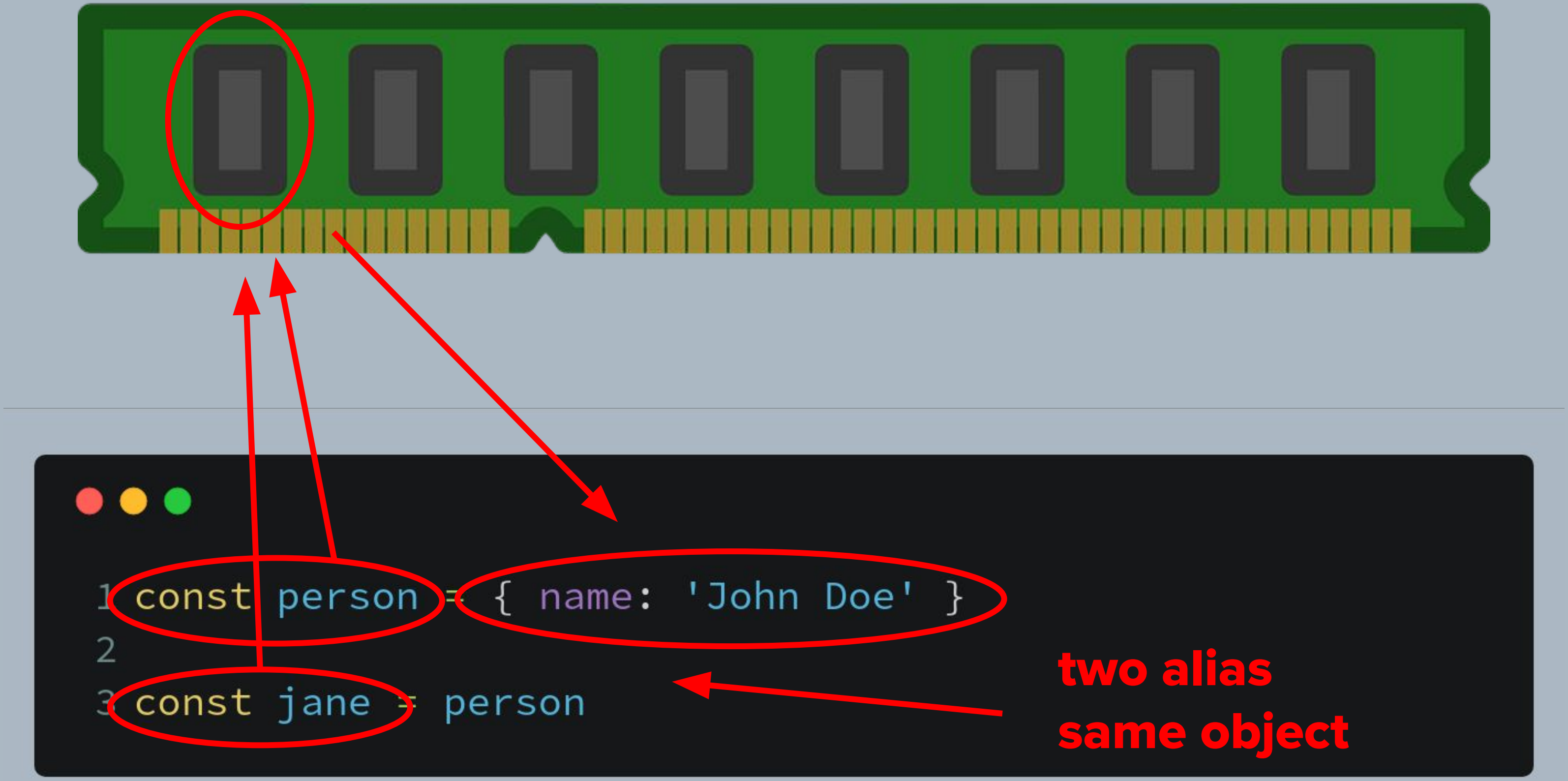




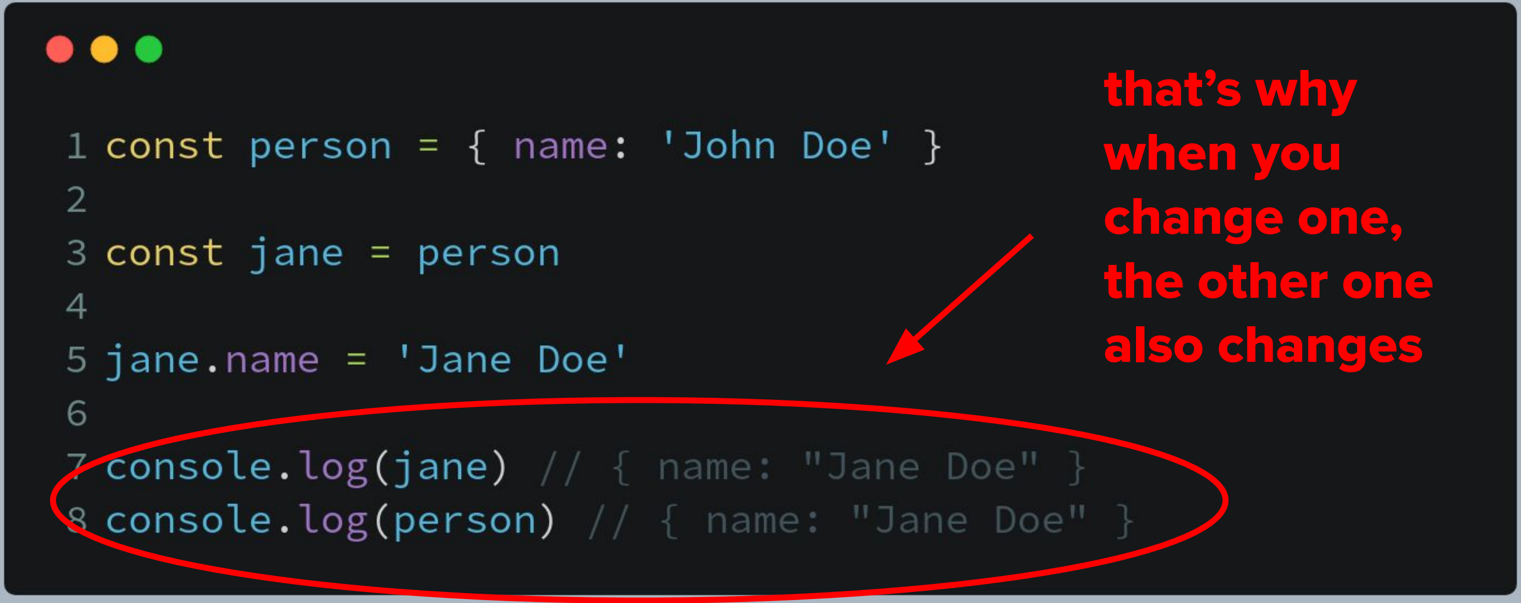
```
const person = { name: 'John Doe' }

const jane = person
```



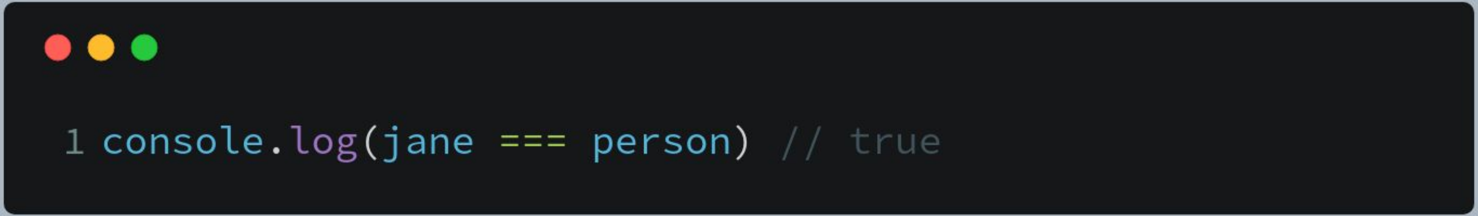
```
const person = { name: 'John Doe' }

const jane = person
```



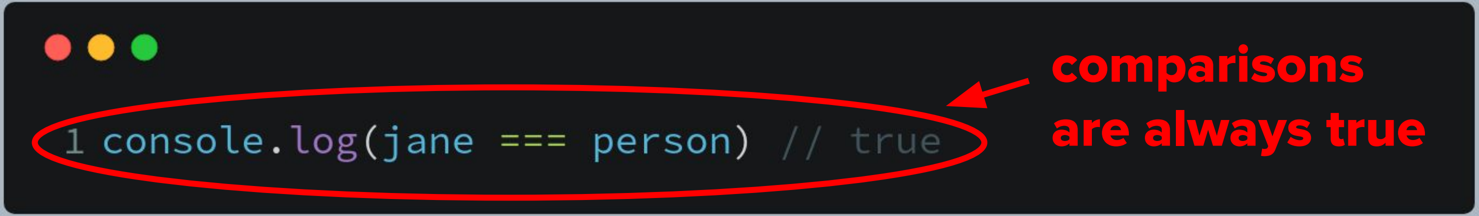
```
const person = { name: 'John Doe' }

const jane = person
```



```
const person = { name: 'John Doe' }

const jane = person

jane.name = 'Jane Doe'

console.log(jane) // { name: "Jane Doe" }
console.log(person) // { name: "Jane Doe" }
```

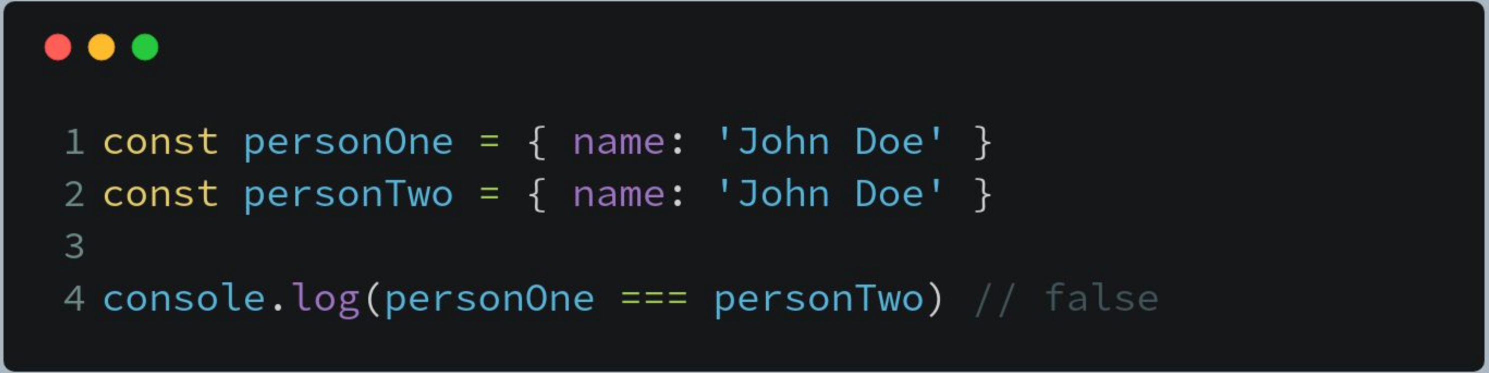


```
console.log(jane === person) // true
```

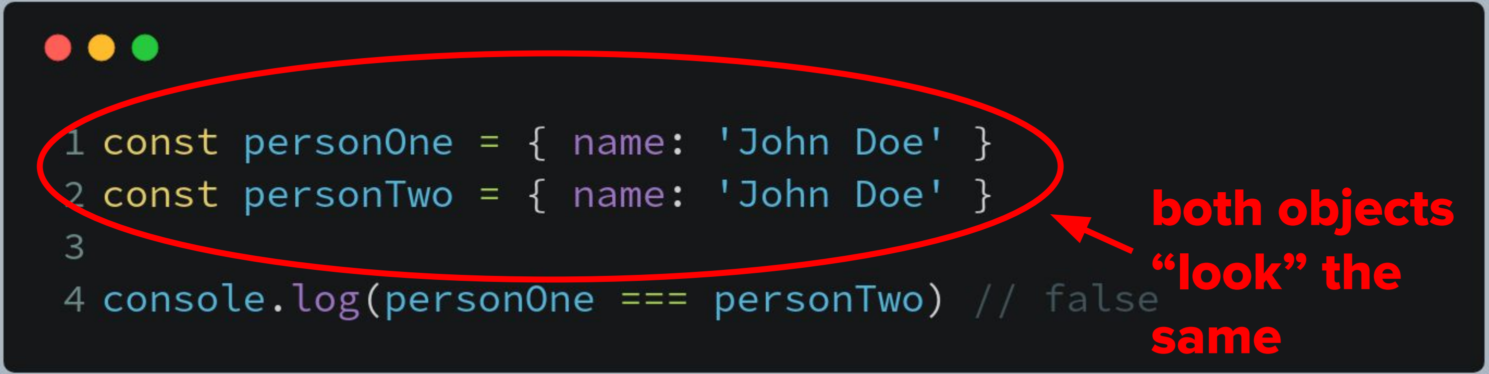


```
console.log(jane === person) // true
```

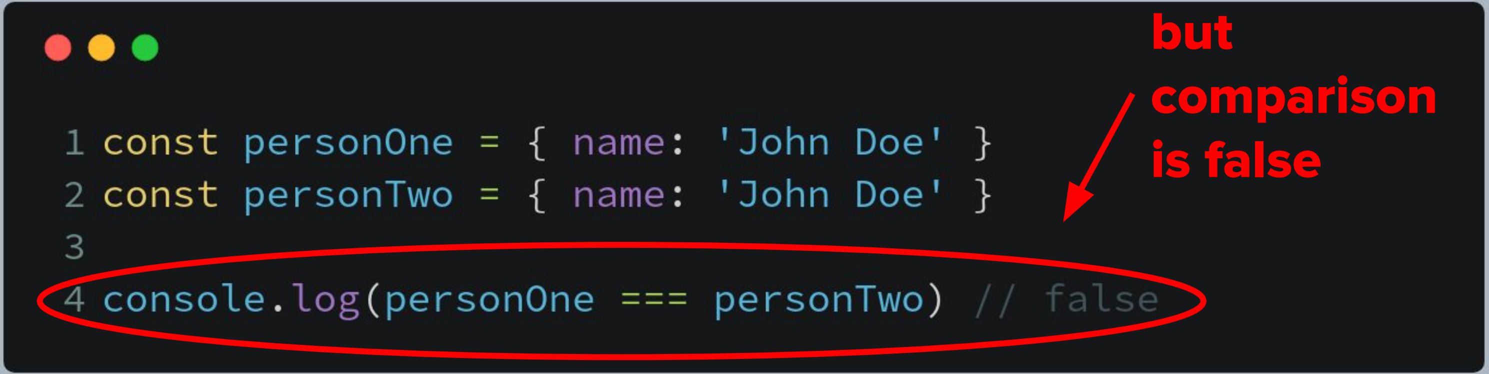
One more example



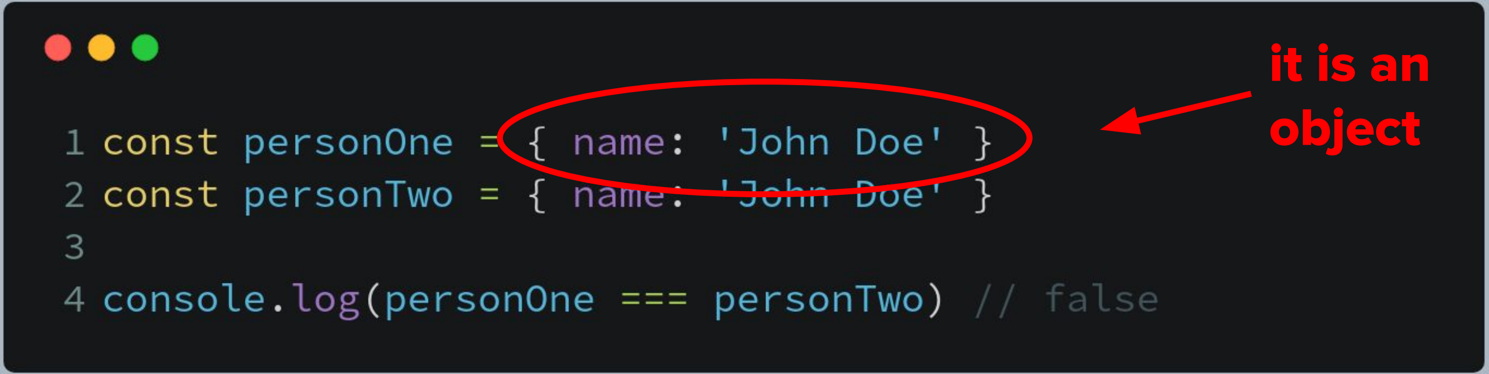
```
const personOne = { name: 'John Doe' }
const personTwo = { name: 'John Doe' }

console.log(personOne === personTwo) // false
```



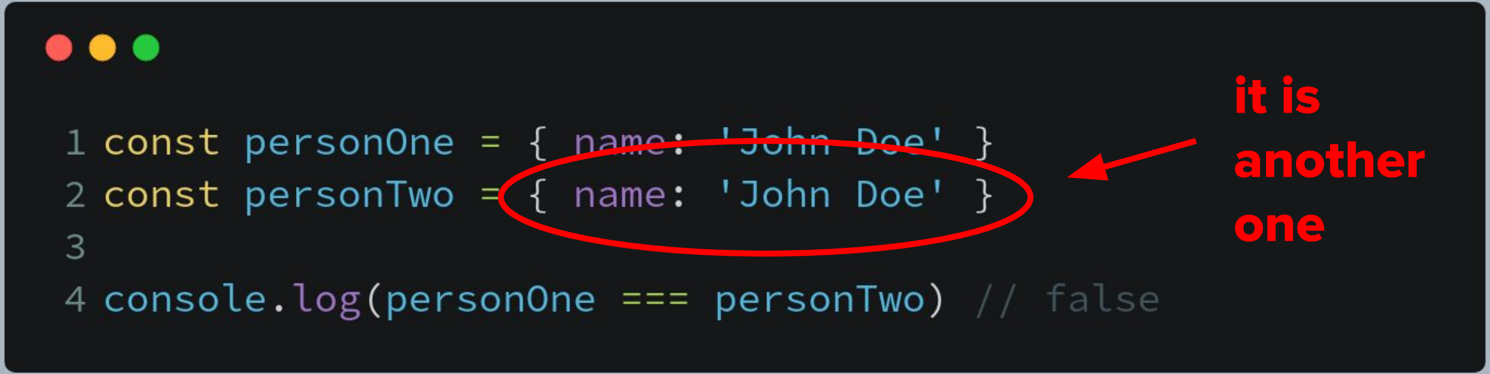
```
const personOne = { name: 'John Doe' }
const personTwo = { name: 'John Doe' }

console.log(personOne === personTwo) // false
```



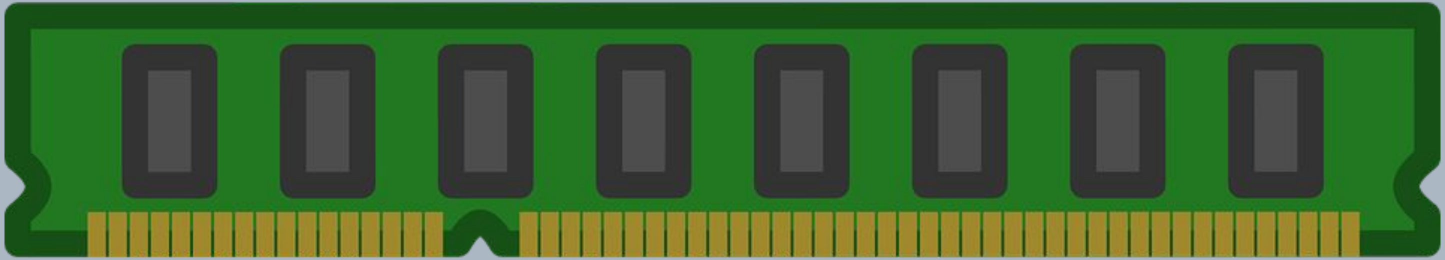
```
const personOne = { name: 'John Doe' }
const personTwo = { name: 'John Doe' }

console.log(personOne === personTwo) // false
```

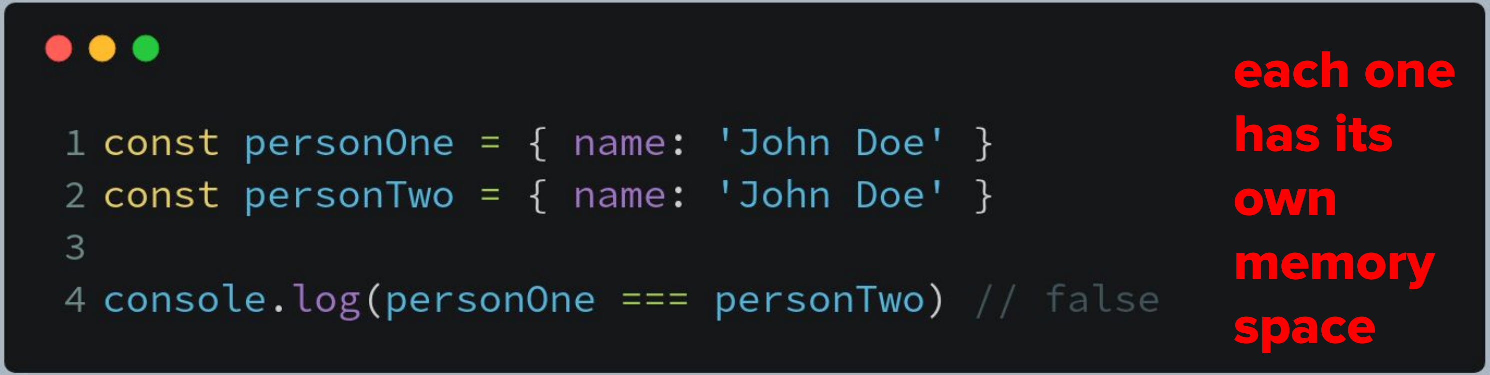


```javascript
const personOne = { name: 'John Doe' }
const personTwo = { name: 'John Doe' }

console.log(personOne === personTwo) // false
```



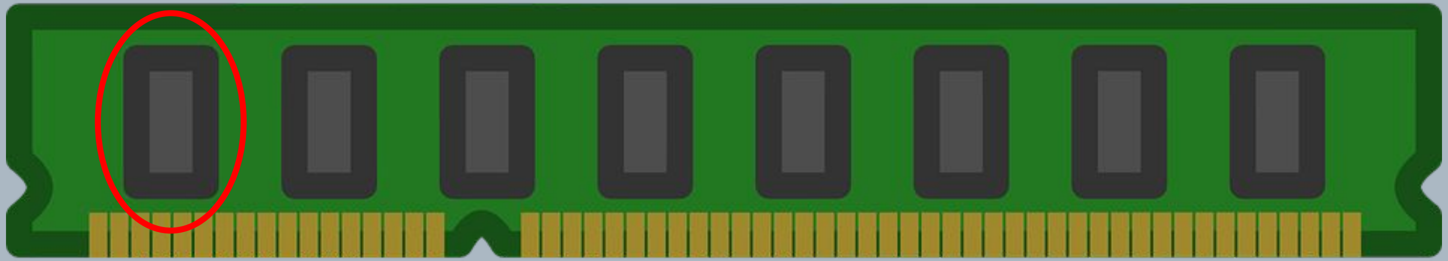
```
const personOne = { name: 'John Doe' }
const personTwo = { name: 'John Doe' }

console.log(personOne === personTwo) // false
```

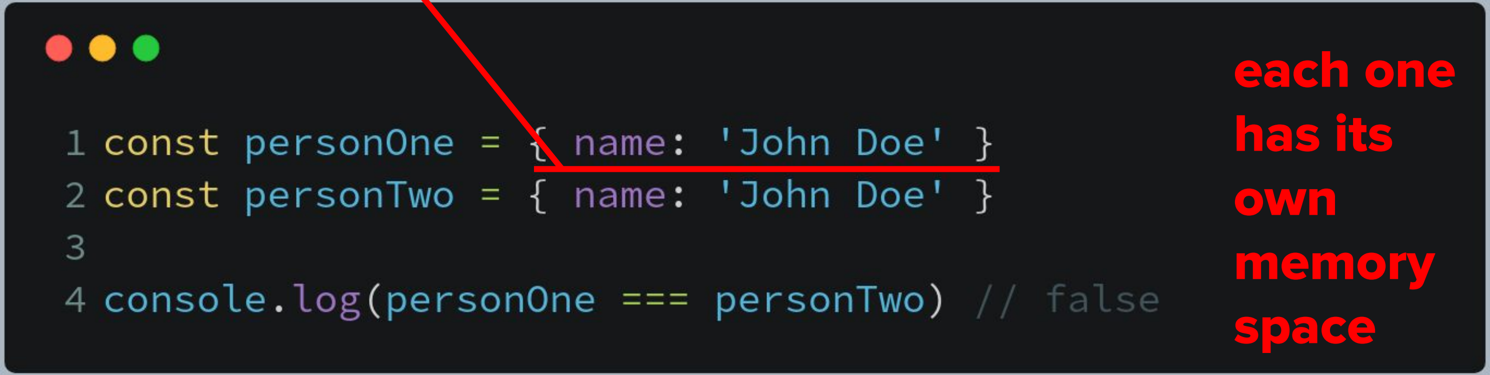




```
const personOne = { name: 'John Doe' }
const personTwo = { name: 'John Doe' }

console.log(personOne === personTwo) // false
```

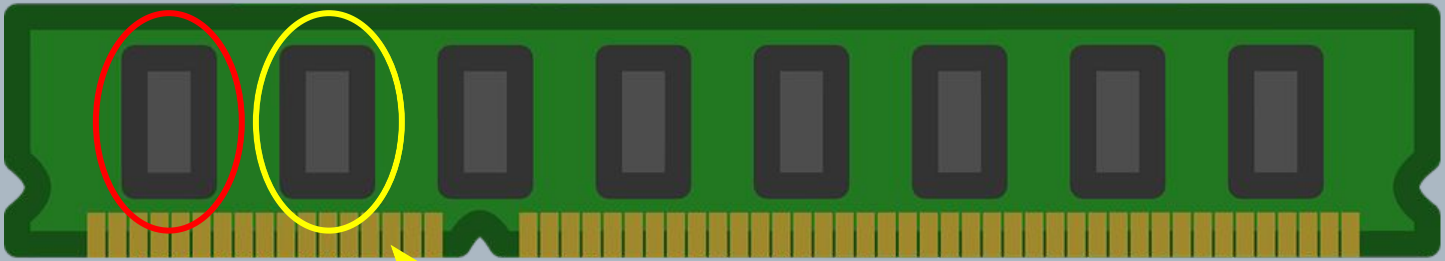
each one has its own memory space

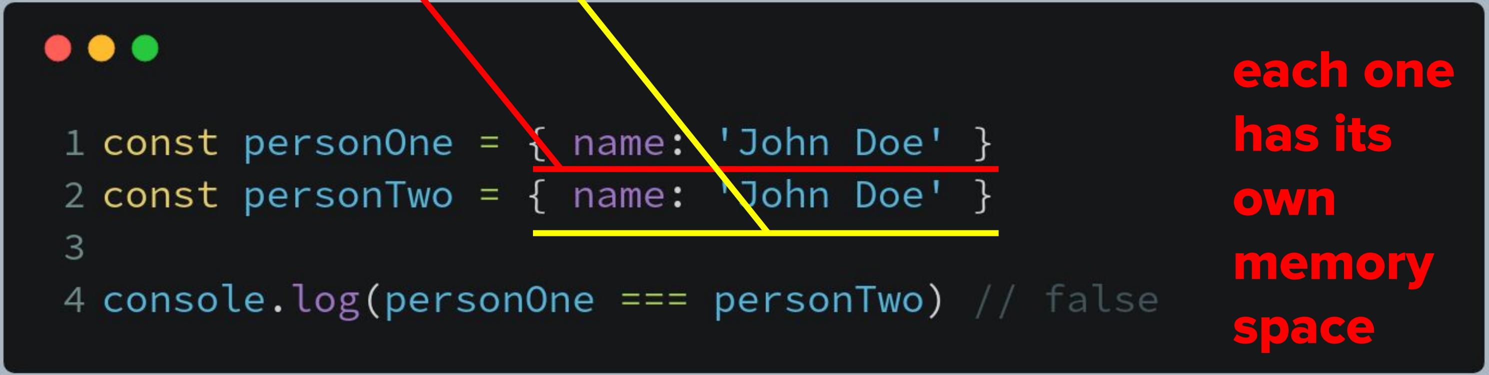




```
const personOne = { name: 'John Doe' }
const personTwo = { name: 'John Doe' }

console.log(personOne === personTwo) // false
```

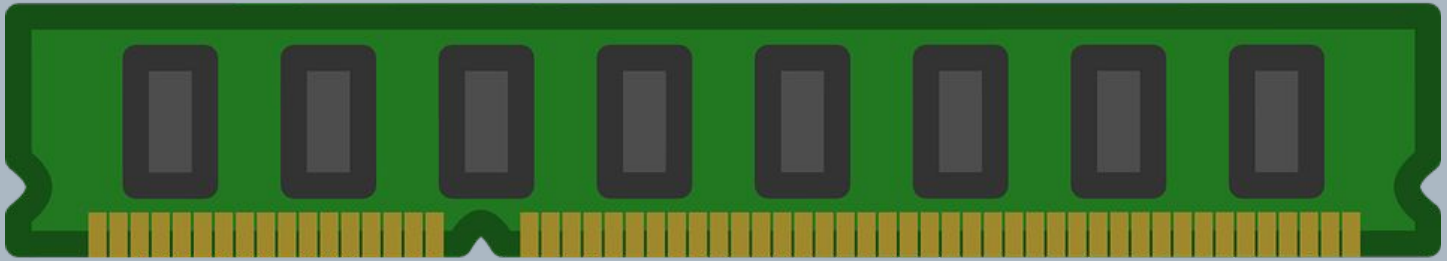
each one has its own memory space

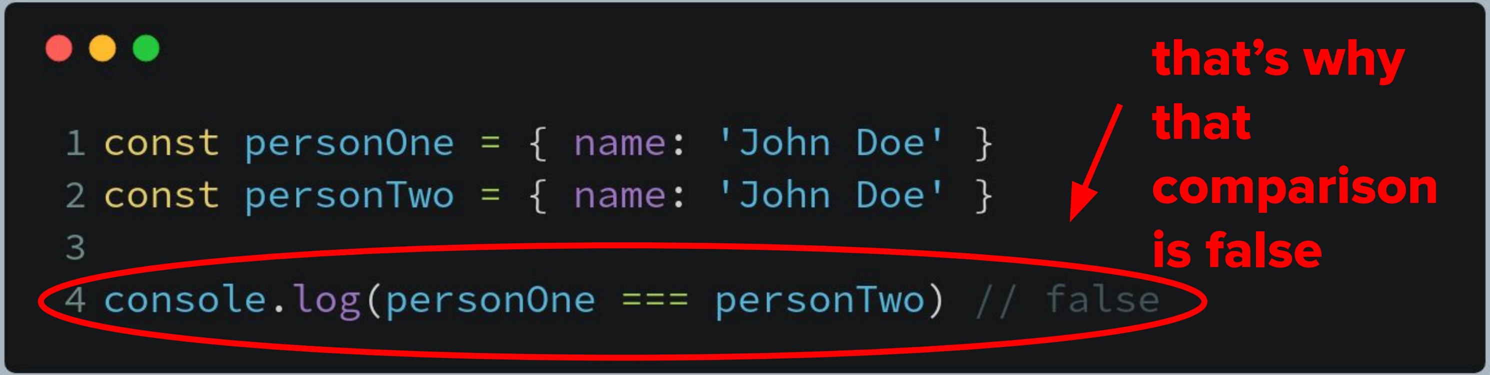




```
const personOne = { name: 'John Doe' }
const personTwo = { name: 'John Doe' }

console.log(personOne === personTwo) // false
```

each one has its own memory space

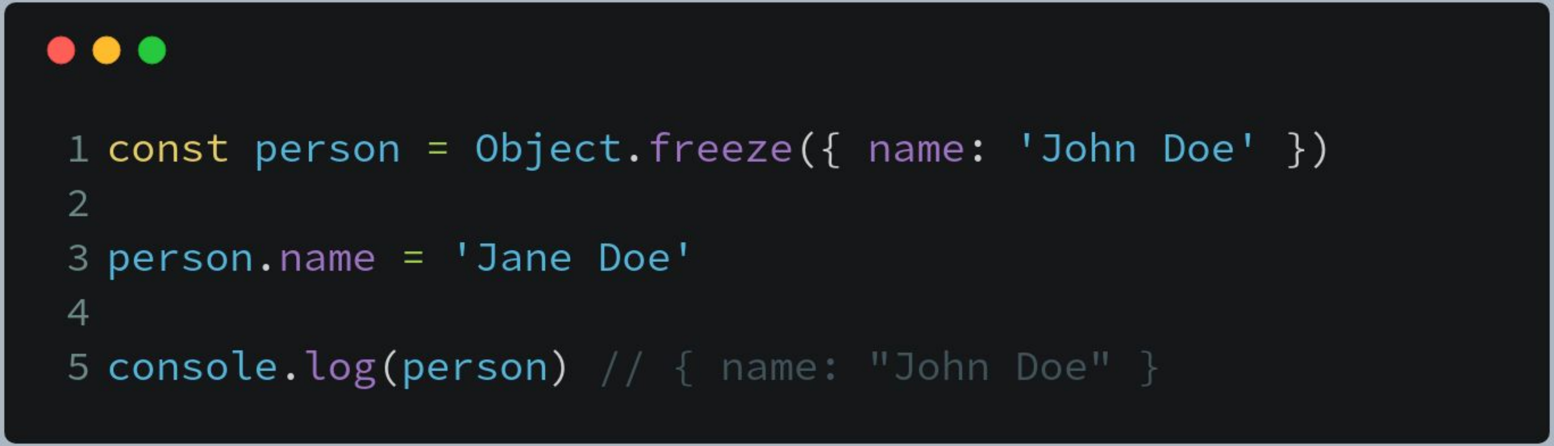




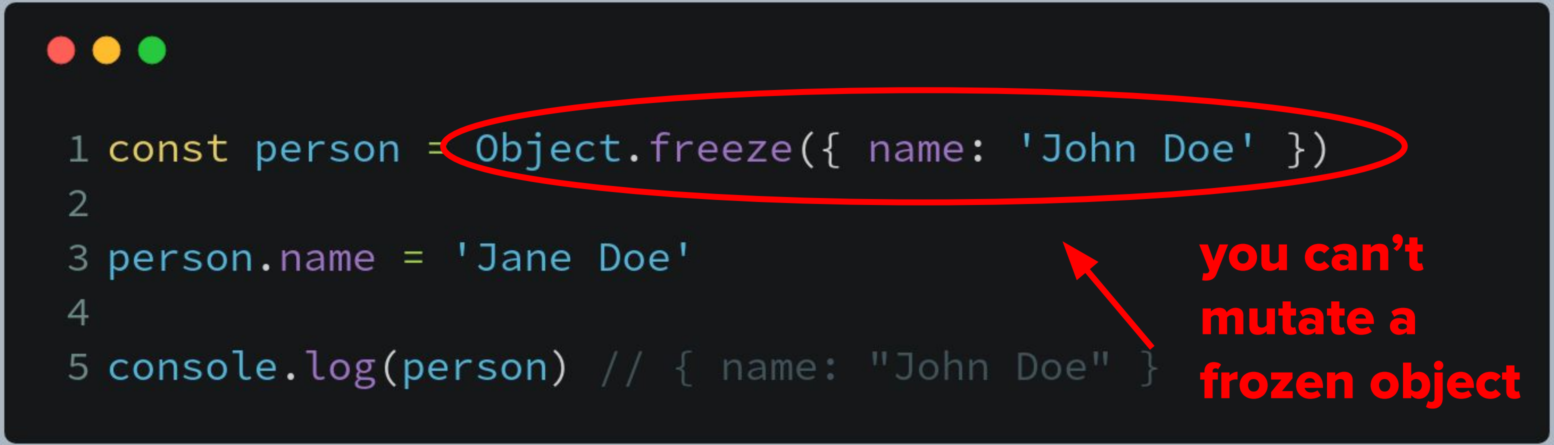
```
const personOne = { name: 'John Doe' }
const personTwo = { name: 'John Doe' }

console.log(personOne === personTwo) // false
```

that's why that comparison is false

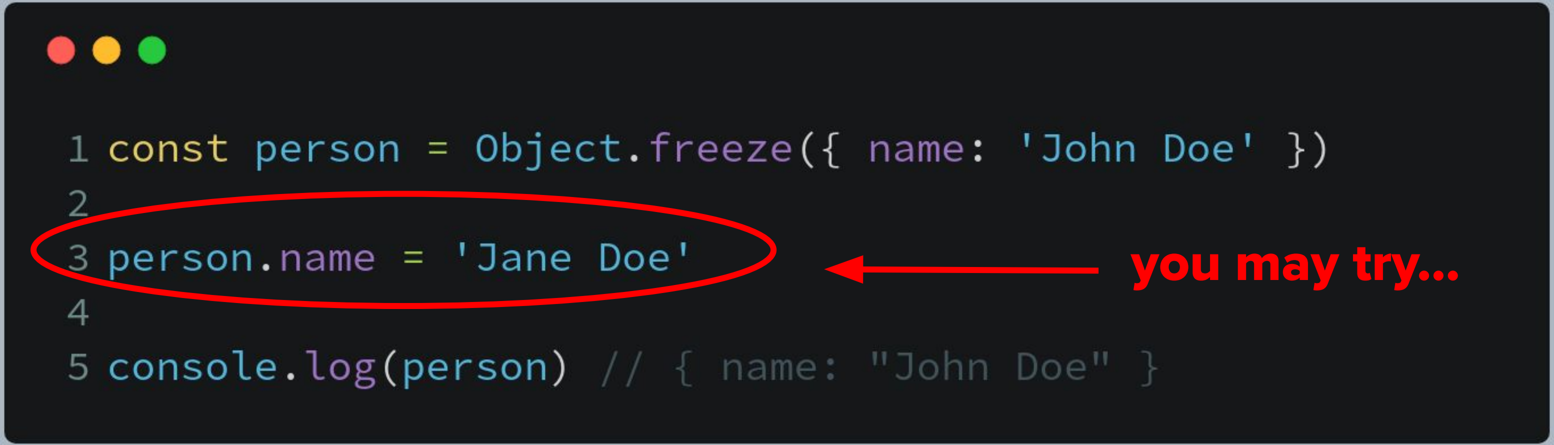
So, how to avoid mutability with objects?



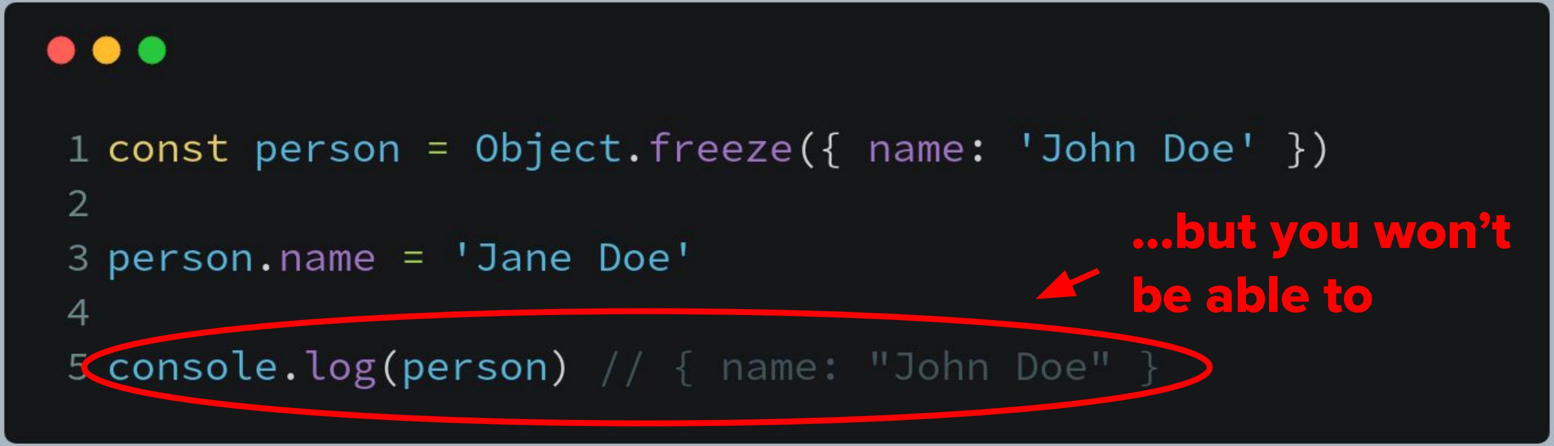
```javascript
const person = Object.freeze({ name: 'John Doe' })

person.name = 'Jane Doe'

console.log(person) // { name: "John Doe" }
```



```
const person = Object.freeze({ name: 'John Doe' })

person.name = 'Jane Doe'

console.log(person) // { name: "John Doe" }
```

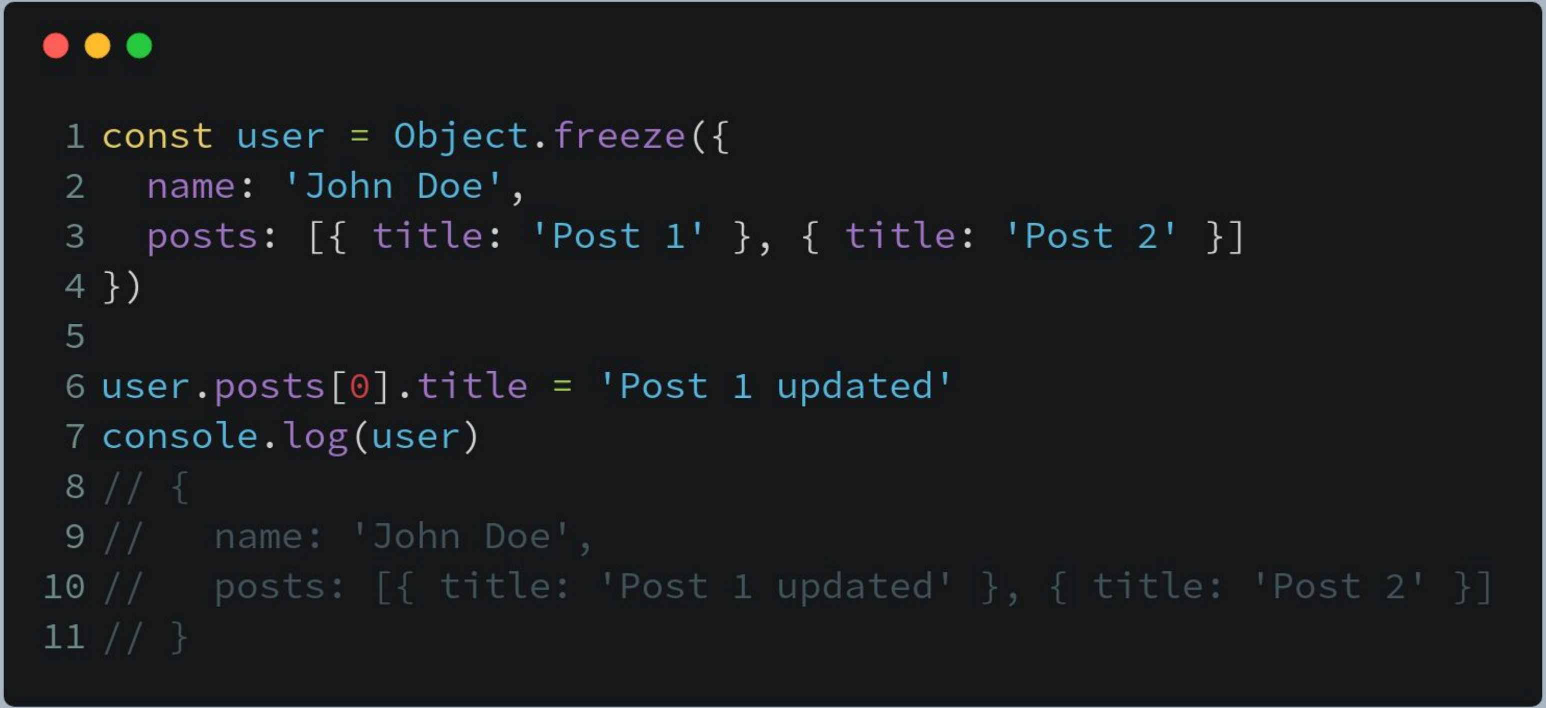


```
const person = Object.freeze({ name: 'John Doe' })

person.name = 'Jane Doe'

console.log(person) // { name: "John Doe" }
```

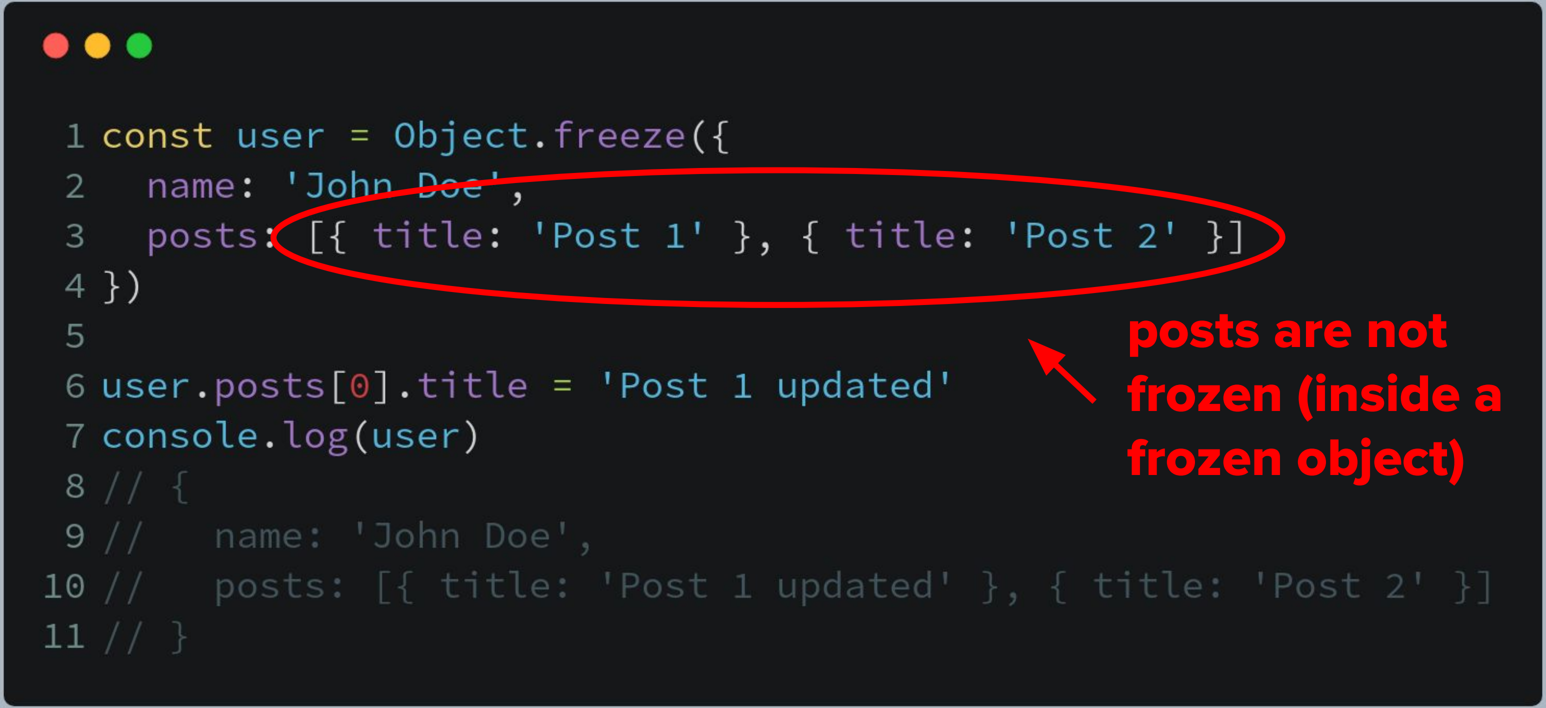


```
const person = Object.freeze({ name: 'John Doe' })

person.name = 'Jane Doe'

console.log(person) // { name: "John Doe" }
```

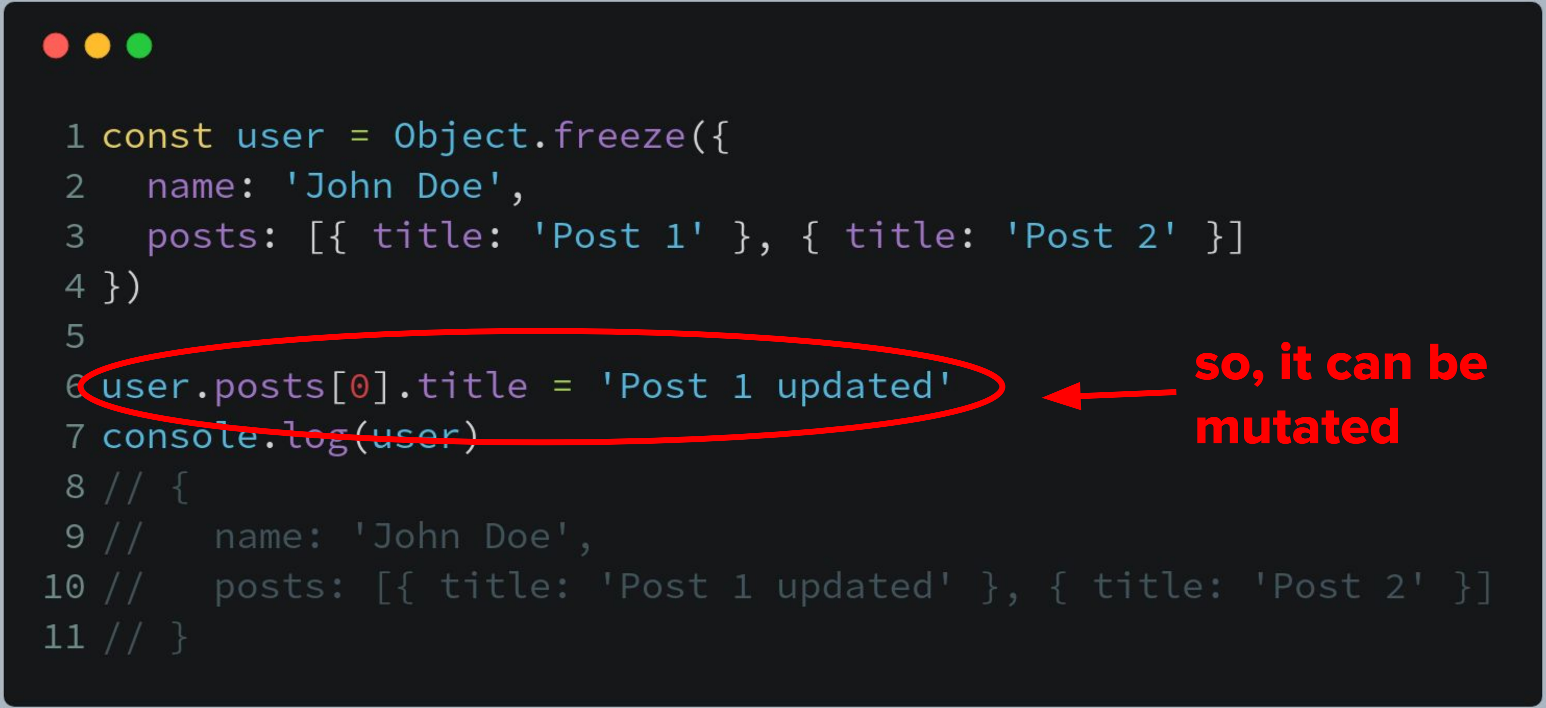
But **internal objects** are **new references**



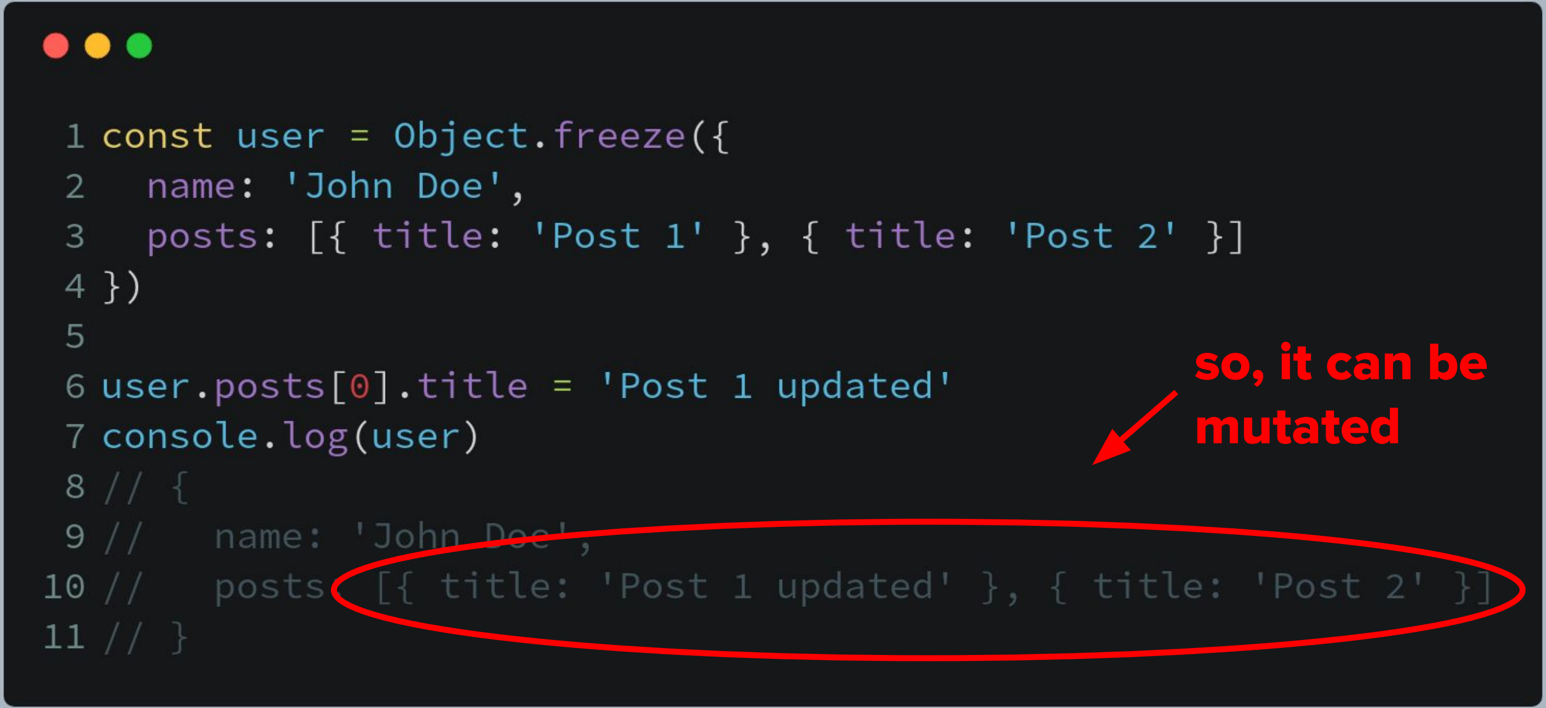
```javascript
const user = Object.freeze({
  name: 'John Doe',
  posts: [{ title: 'Post 1' }, { title: 'Post 2' }]
})

user.posts[0].title = 'Post 1 updated'
console.log(user)
// {
//   name: 'John Doe',
//   posts: [{ title: 'Post 1 updated' }, { title: 'Post 2' }]
// }
```



```javascript
const user = Object.freeze({
  name: 'John Doe',
  posts: [{ title: 'Post 1' }, { title: 'Post 2' }]
})

user.posts[0].title = 'Post 1 updated'
console.log(user)
// {
//   name: 'John Doe',
//   posts: [{ title: 'Post 1 updated' }, { title: 'Post 2' }]
// }
```



```
const user = Object.freeze({
  name: 'John Doe',
  posts: [{ title: 'Post 1' }, { title: 'Post 2' }]
})

user.posts[0].title = 'Post 1 updated'
console.log(user)
// {
//   name: 'John Doe',
//   posts: [{ title: 'Post 1 updated' }, { title: 'Post 2' }]
// }
```



```javascript
const user = Object.freeze({
  name: 'John Doe',
  posts: [{ title: 'Post 1' }, { title: 'Post 2' }]
})

user.posts[0].title = 'Post 1 updated'
console.log(user)
// {
//   name: 'John Doe',
//   posts: [{ title: 'Post 1 updated' }, { title: 'Post 2' }]
// }
```

The **best** way

to **make**

immutable

code...

Is **thinking** in **immutability**

Instead of

freeze

an object…

Just
don’t
mutate it

And if do I need to change an object?

Then

you are

going to

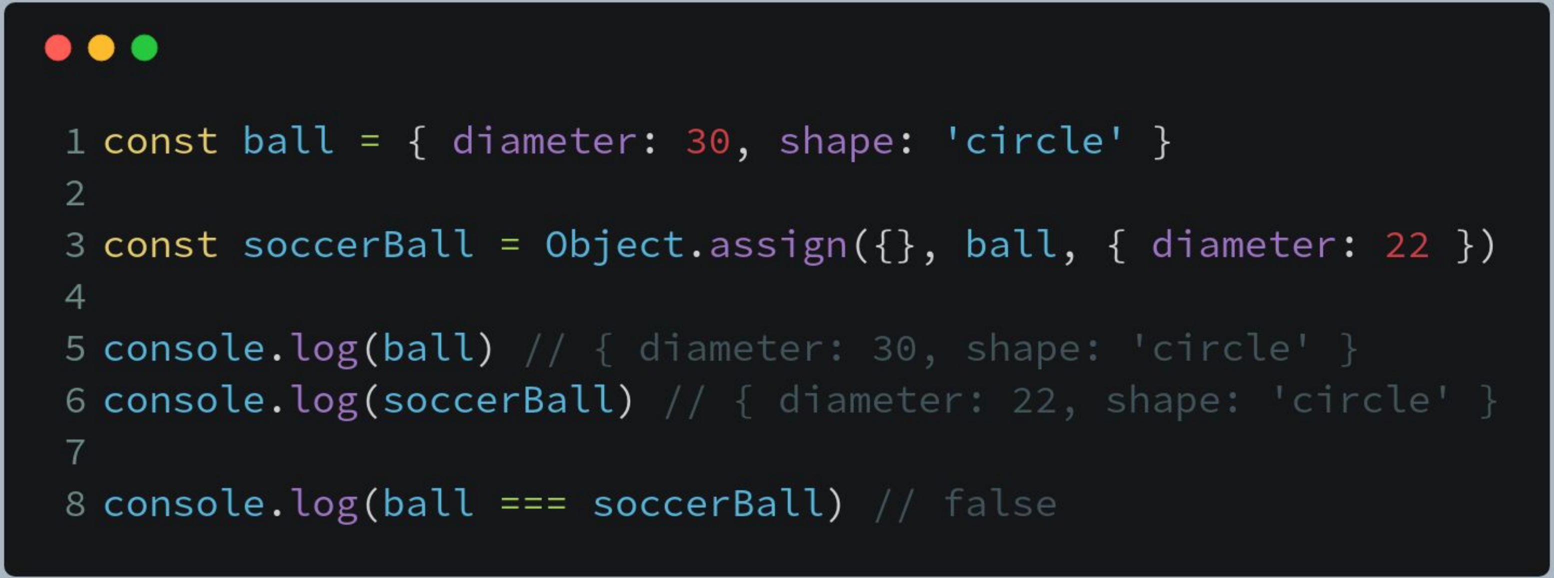
transform

it...

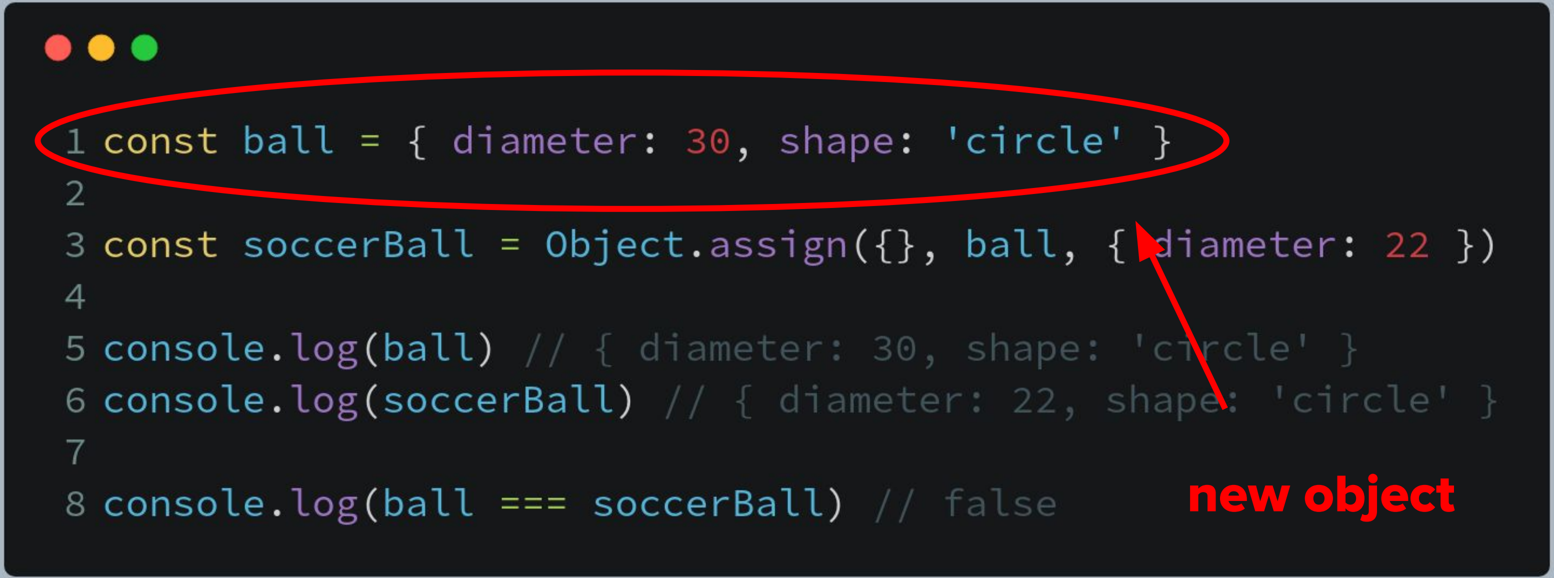
And will **create** a

new copy

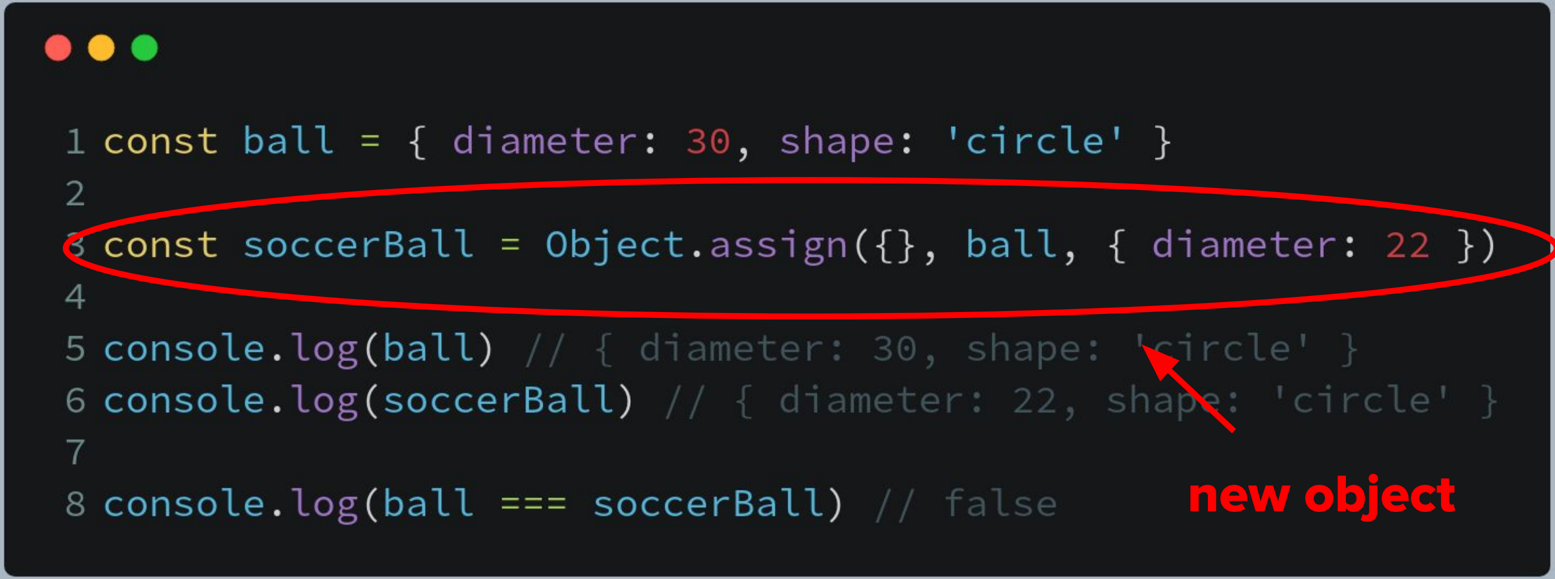
from it



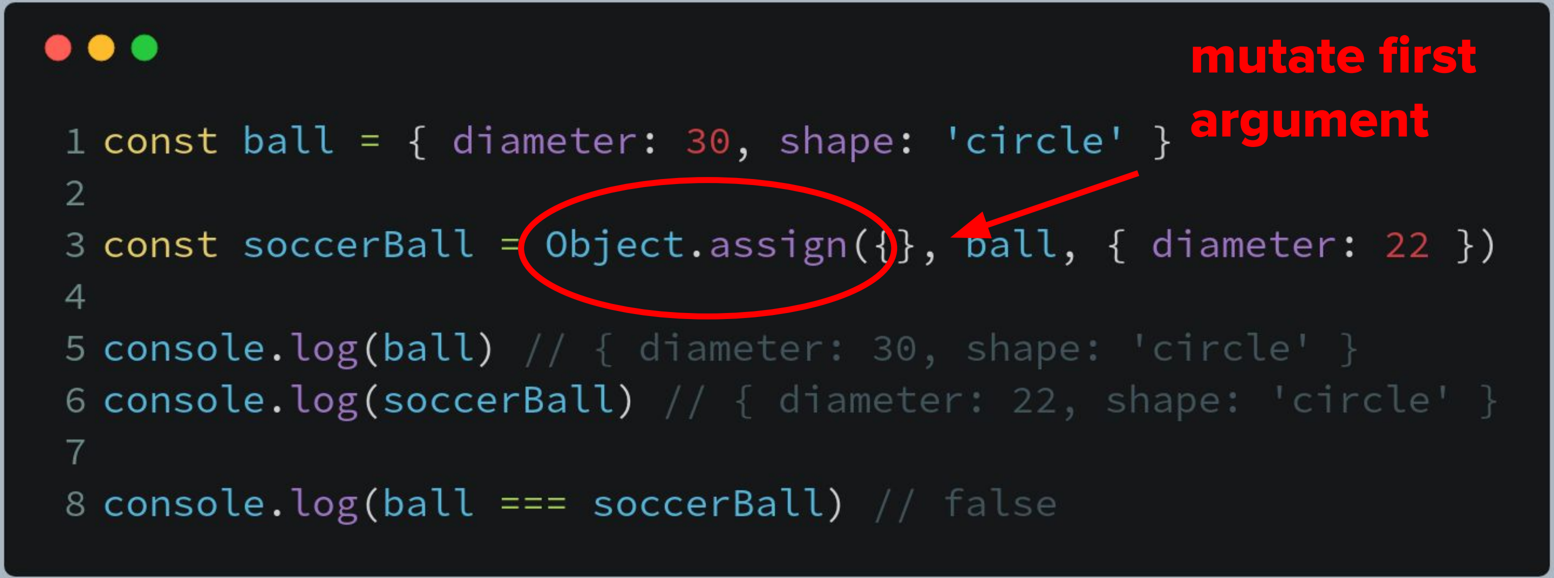
```javascript
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = Object.assign({}, ball, { diameter: 22 })

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```



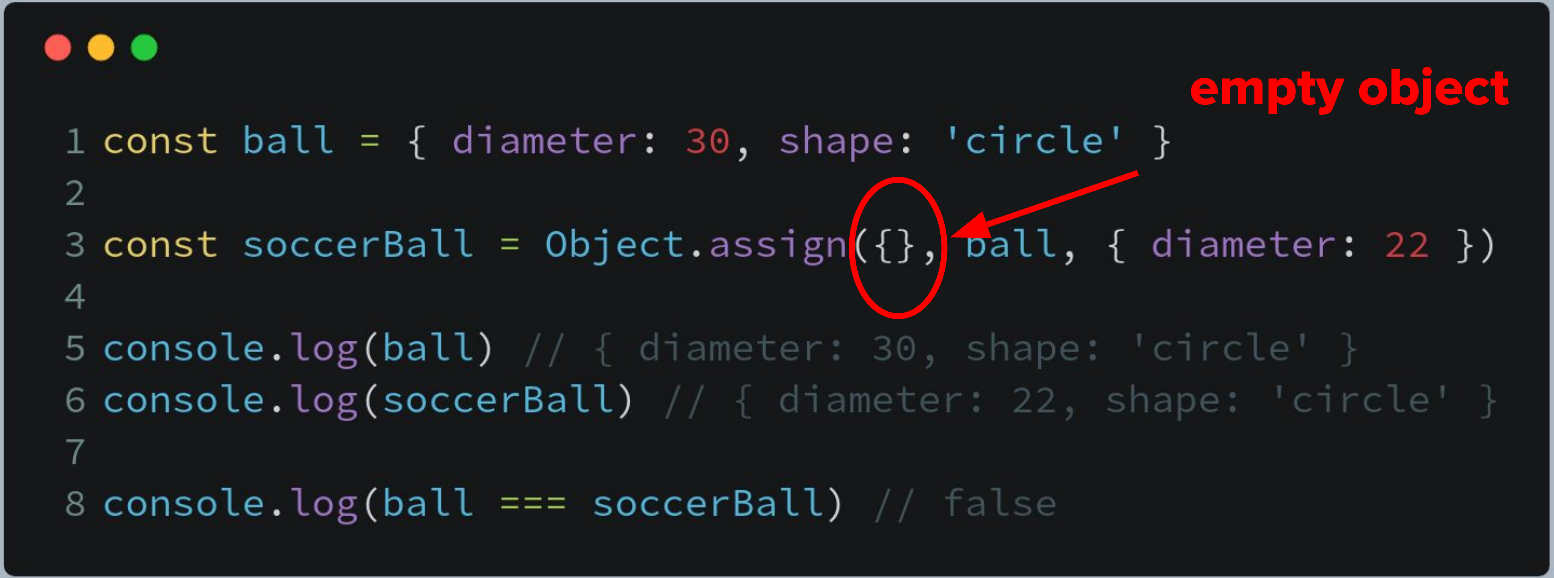
```
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = Object.assign({}, ball, { diameter: 22 })

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```



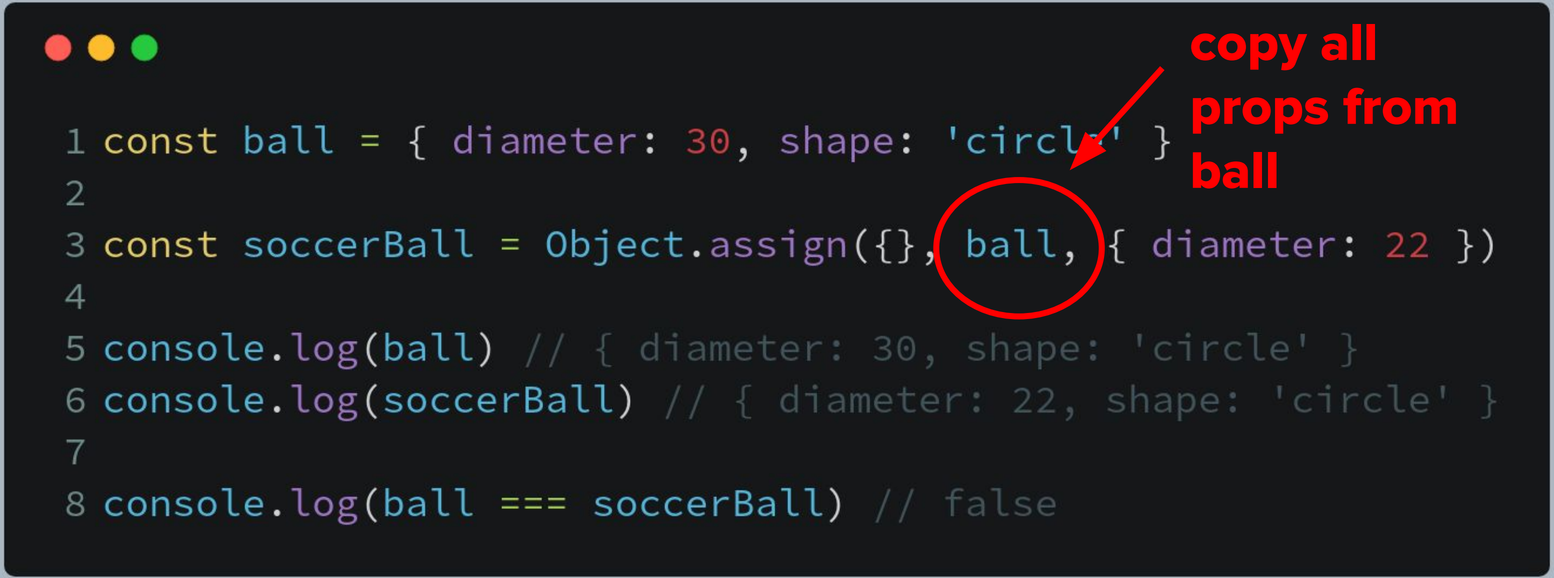
```javascript
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = Object.assign({}, ball, { diameter: 22 })

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```



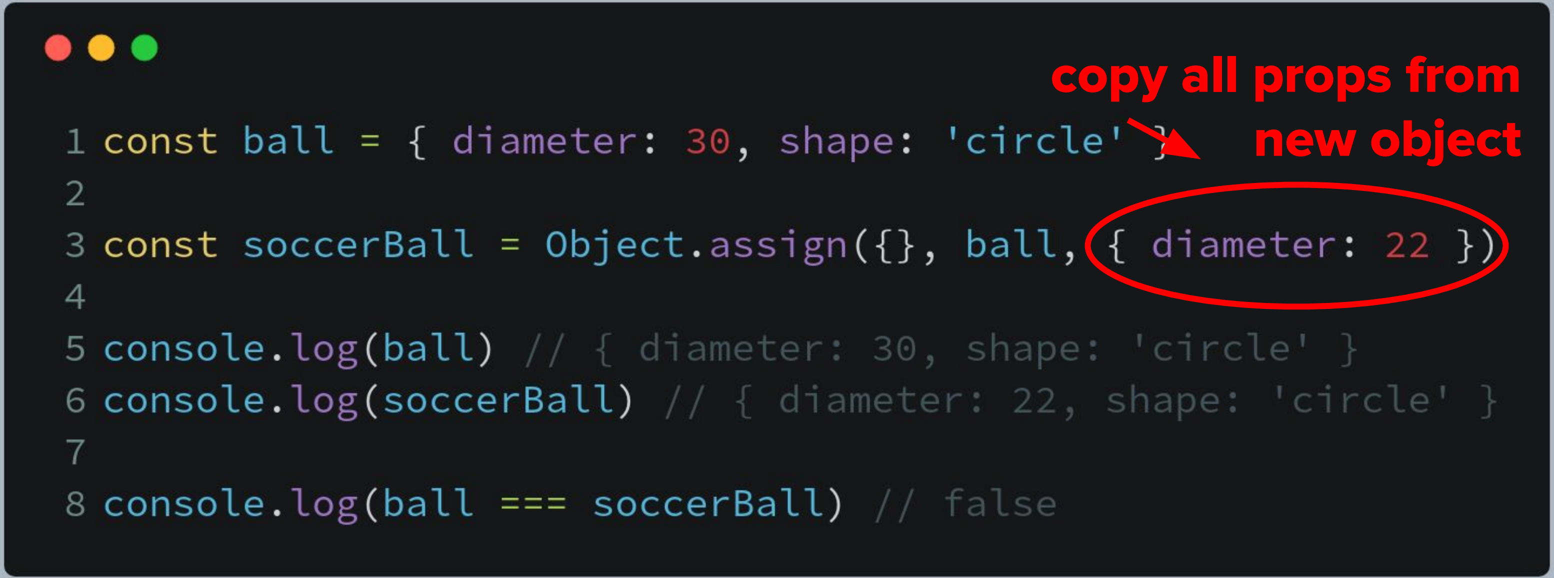
```javascript
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = Object.assign({}, ball, { diameter: 22 })

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```



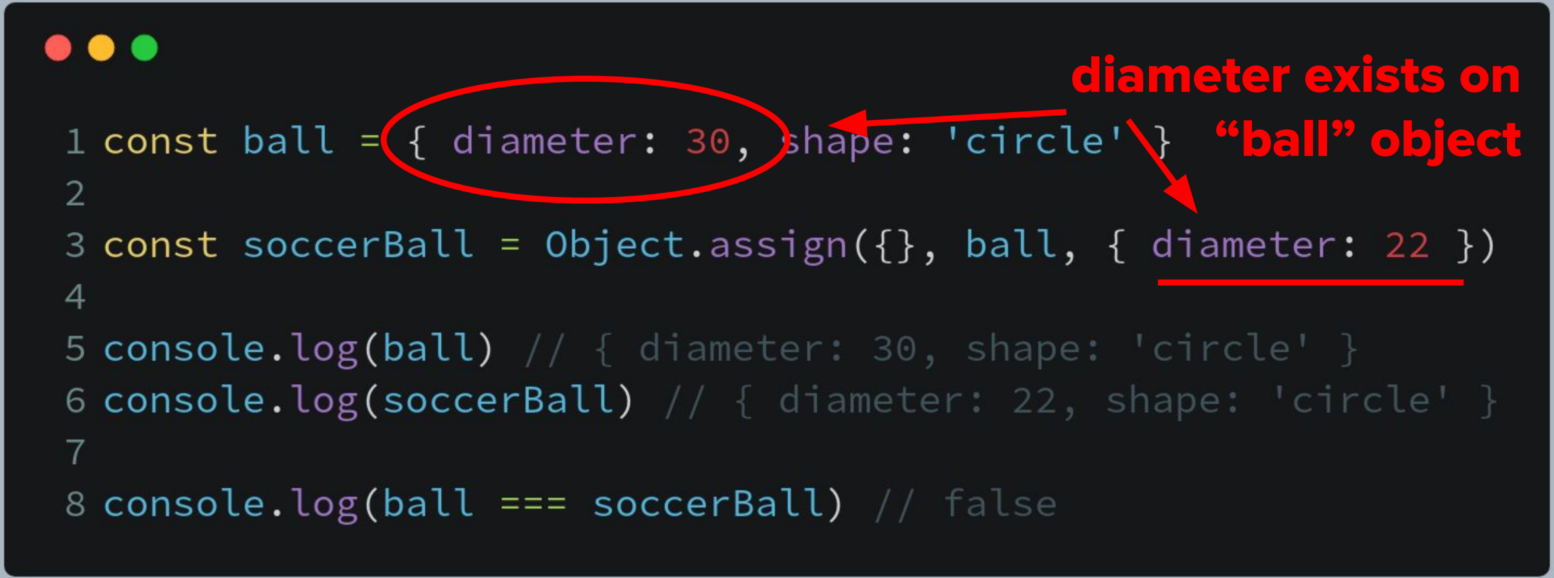
```
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = Object.assign({}, ball, { diameter: 22 })

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```



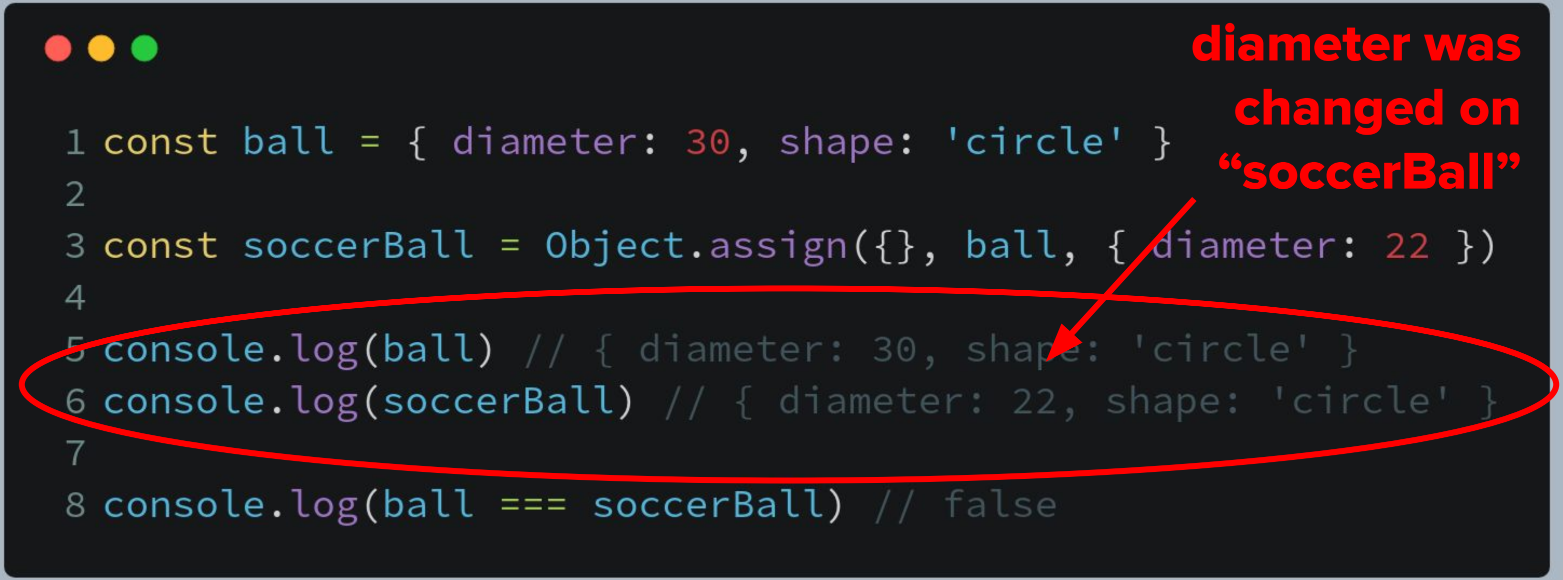
```
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = Object.assign({}, ball, { diameter: 22 })

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```



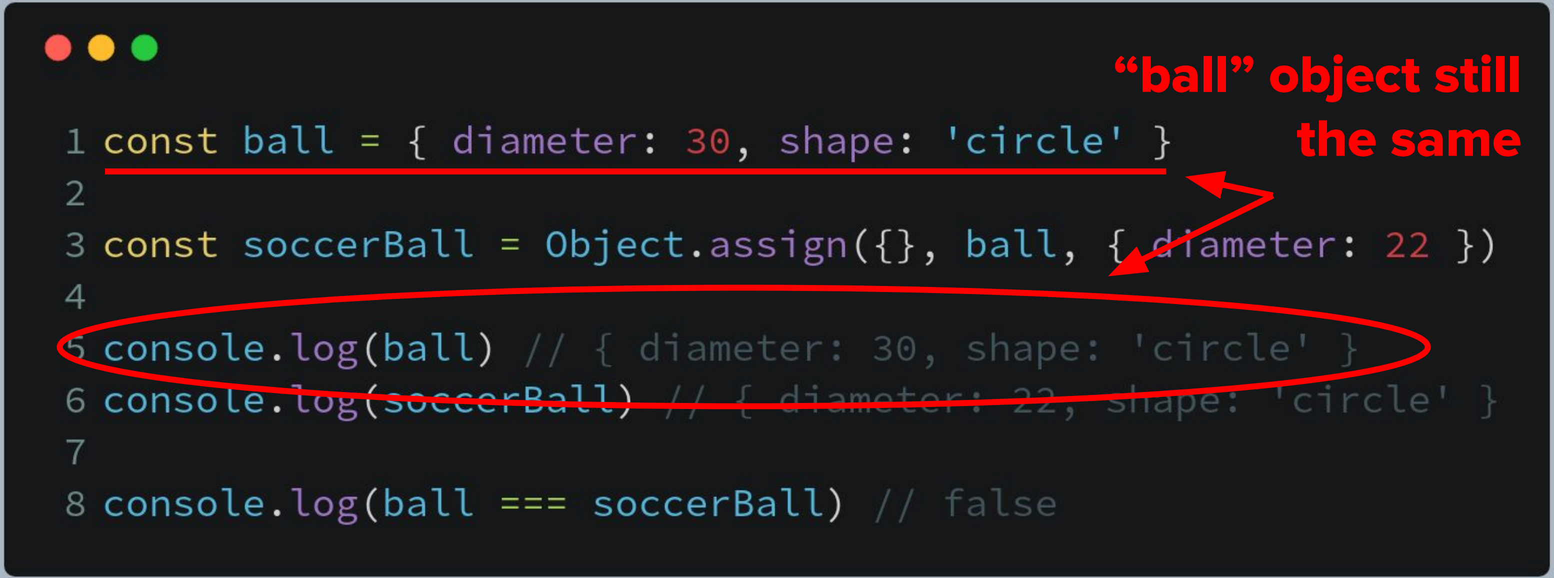
```
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = Object.assign({}, ball, { diameter: 22 })

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```



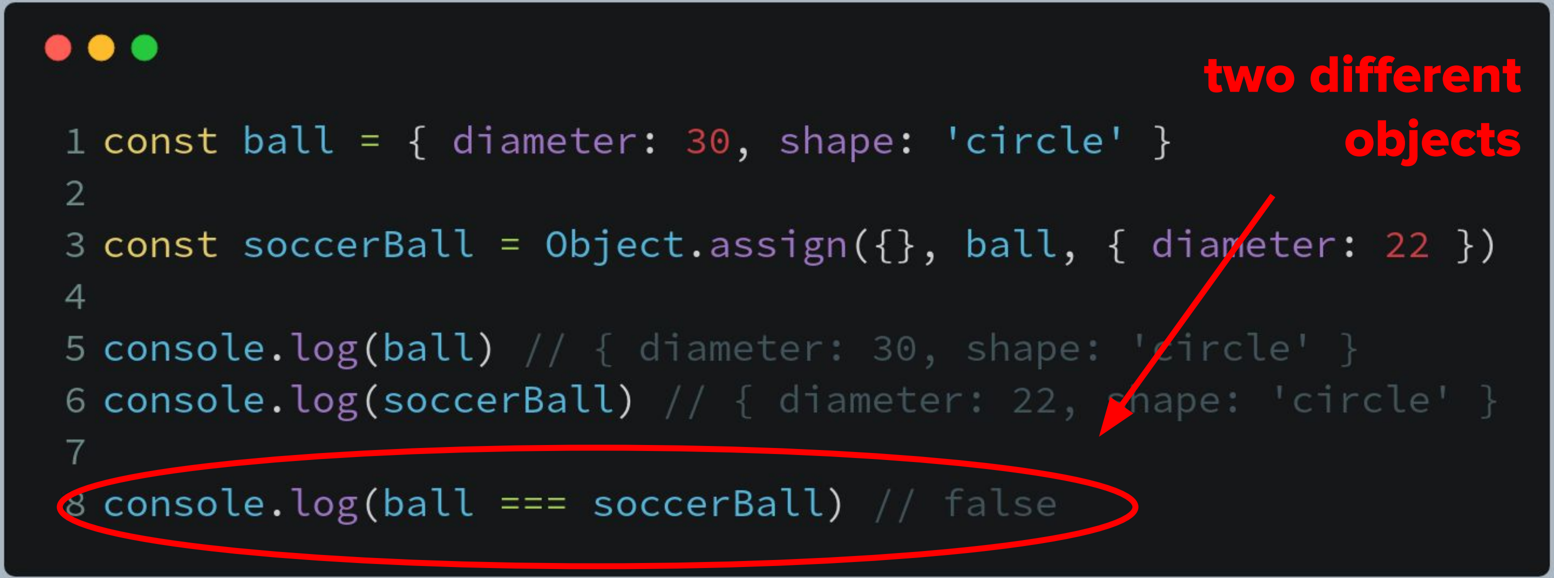
```
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = Object.assign({}, ball, { diameter: 22 })

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```



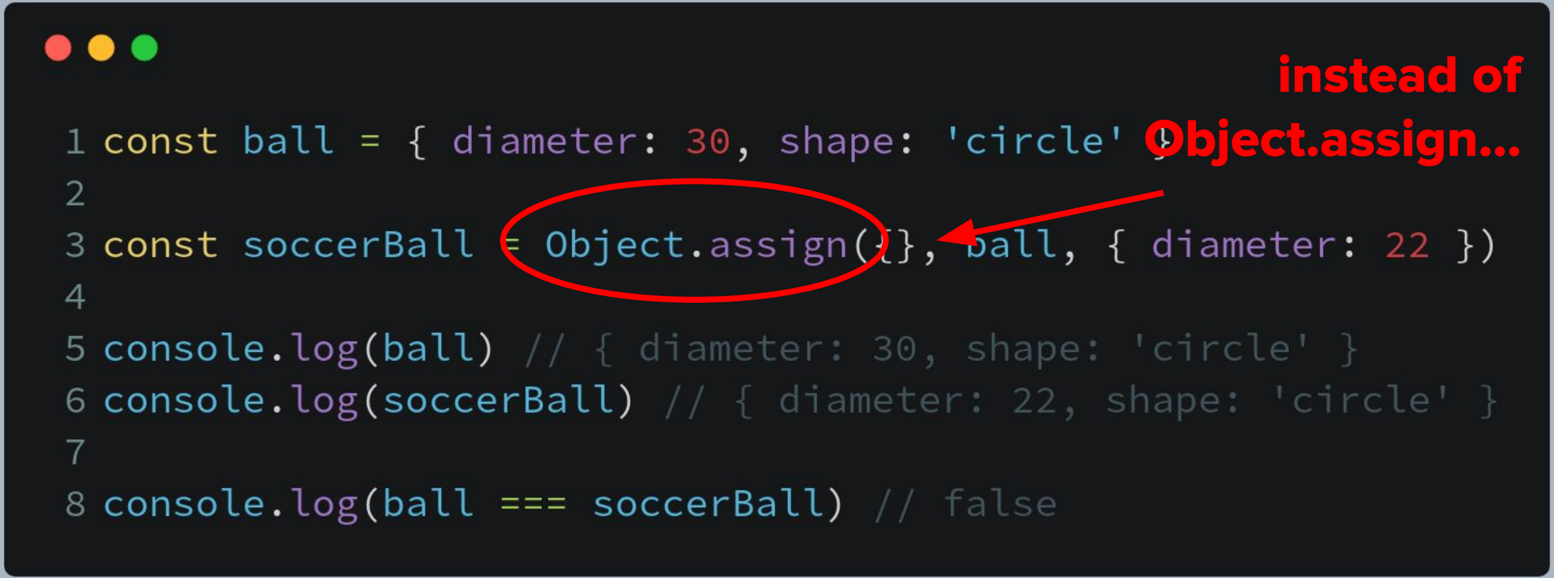
```
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = Object.assign({}, ball, { diameter: 22 })

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```



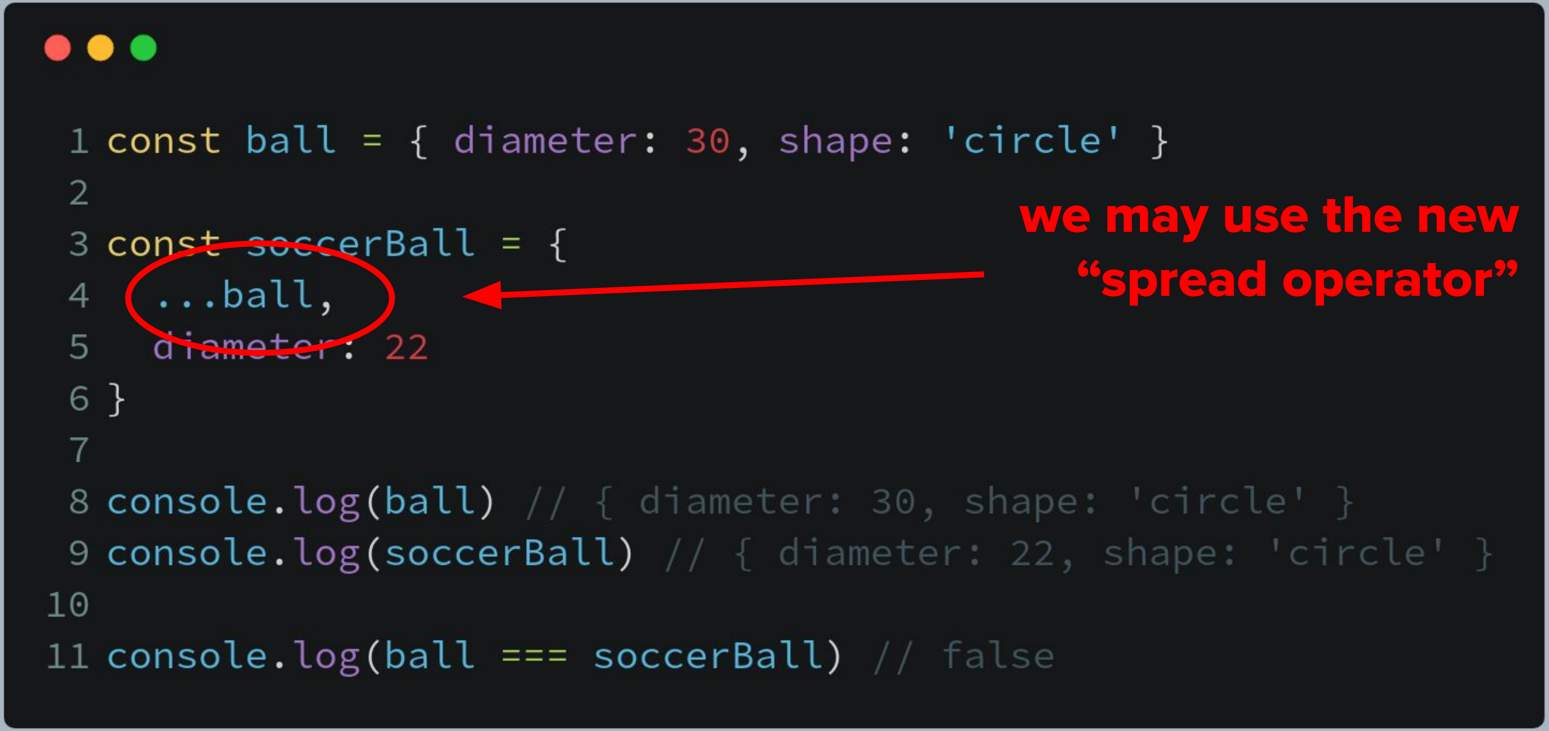
```
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = Object.assign({}, ball, { diameter: 22 })

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```



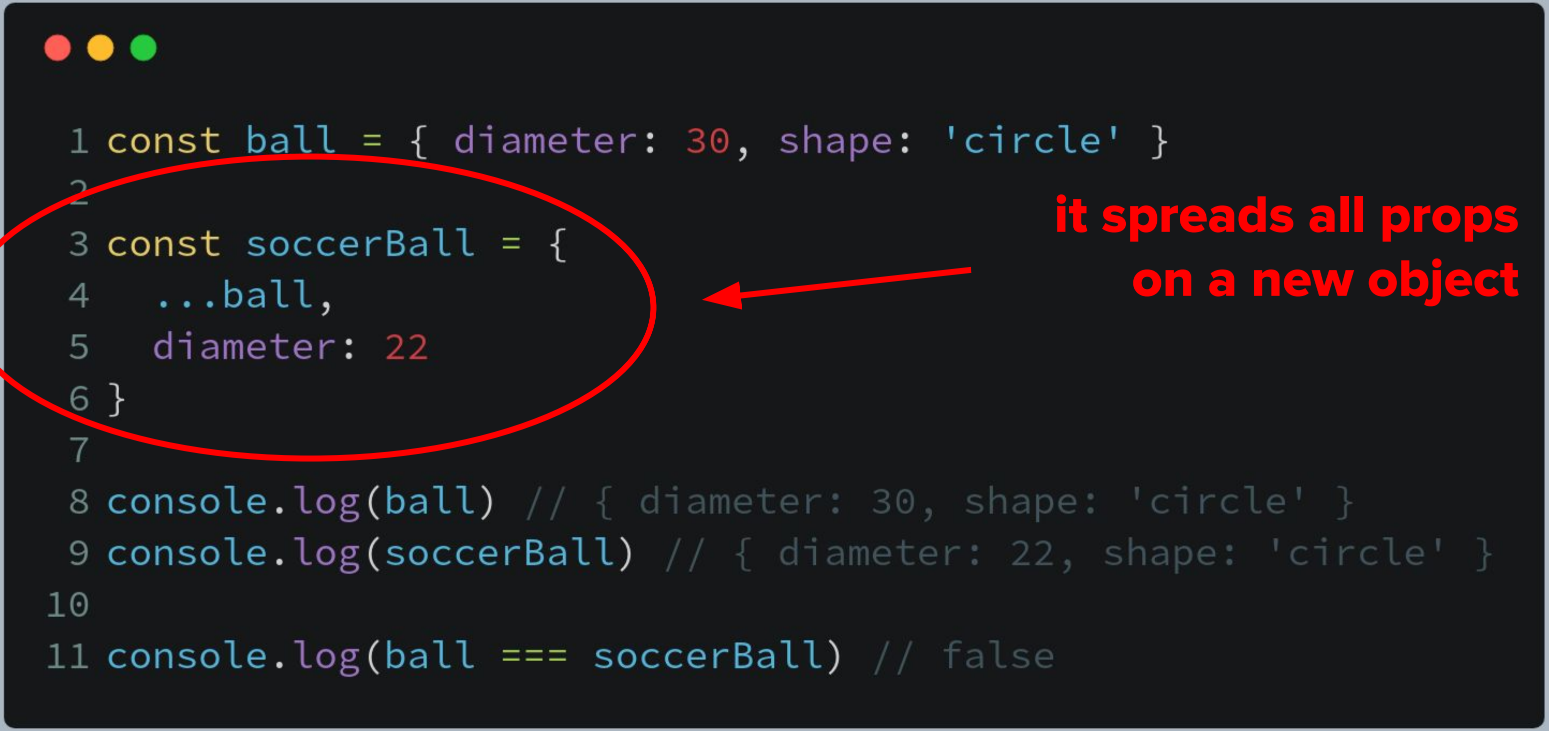
```javascript
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = Object.assign({}, ball, { diameter: 22 })

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```



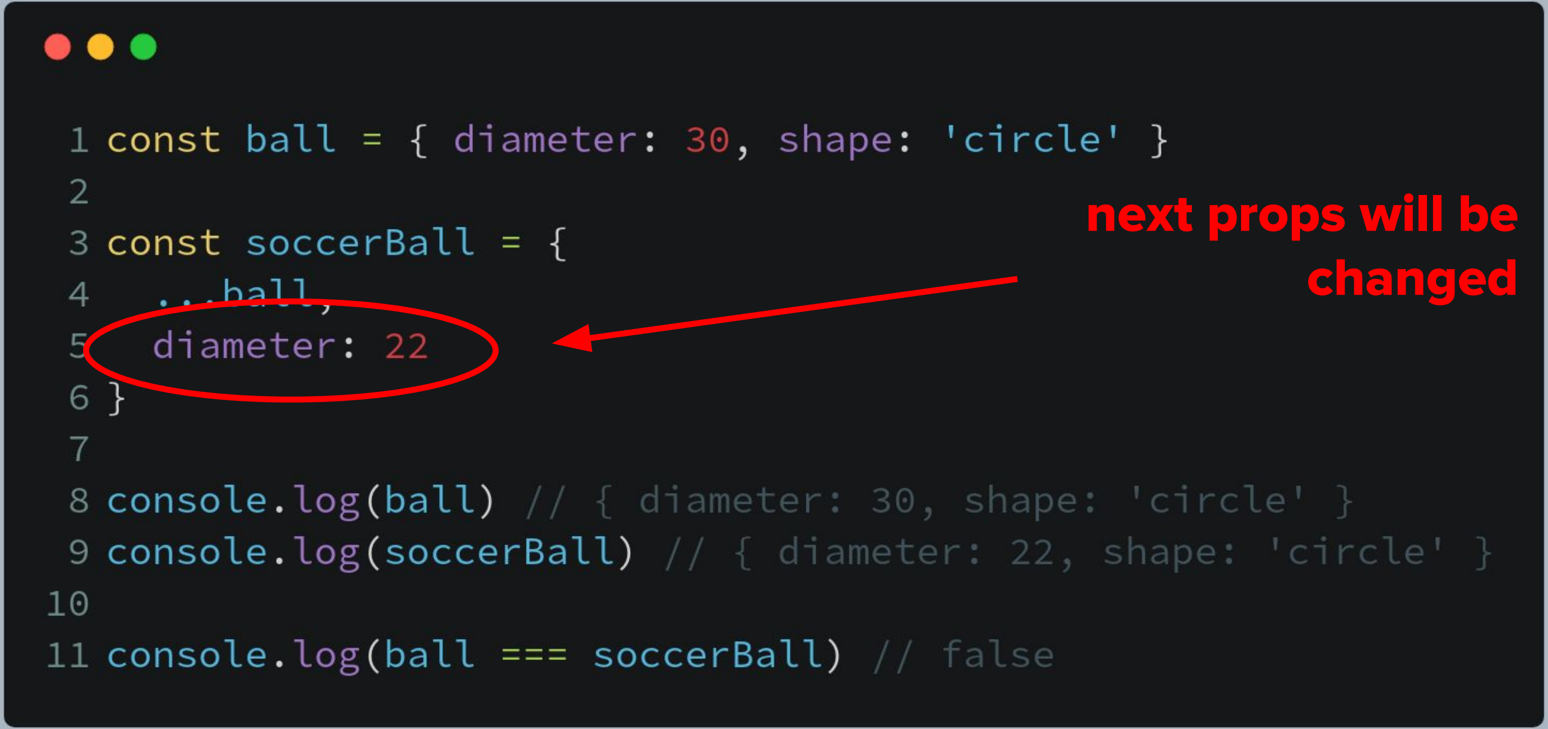
```
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = Object.assign({}, ball, { diameter: 22 })

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```



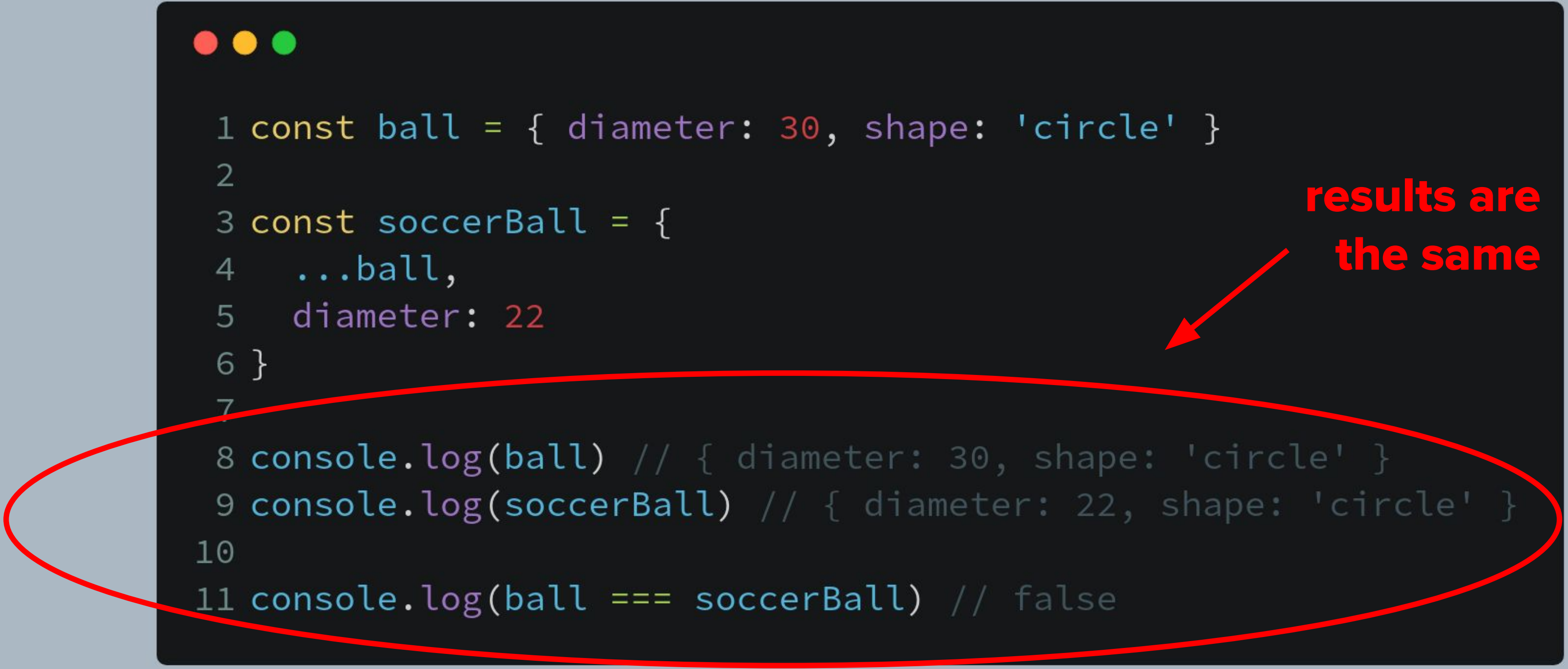
```
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = {
  ...ball,
  diameter: 22
}

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```



```javascript
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = {
  ...ball,
  diameter: 22
}

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```

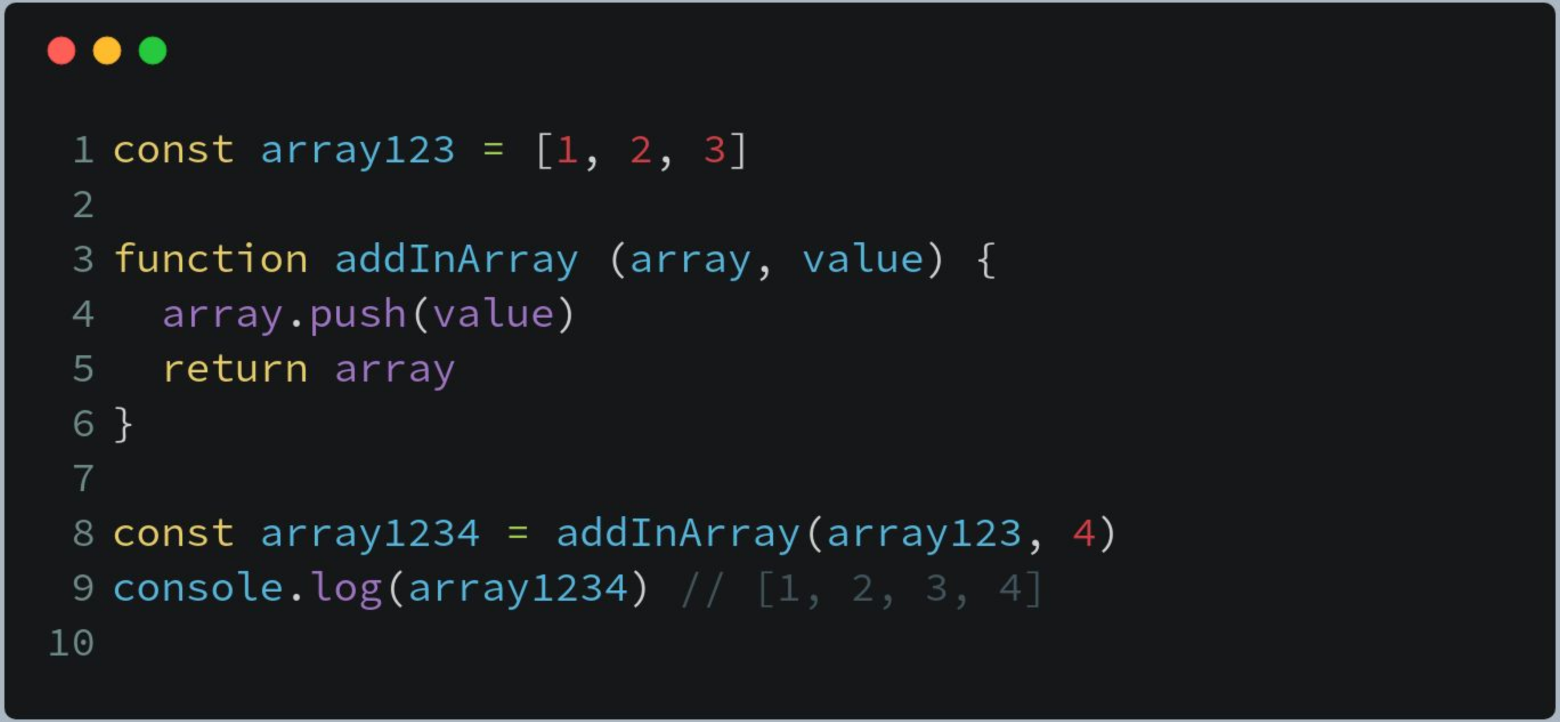


```
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = {
  ...ball,
  diameter: 22
}

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```

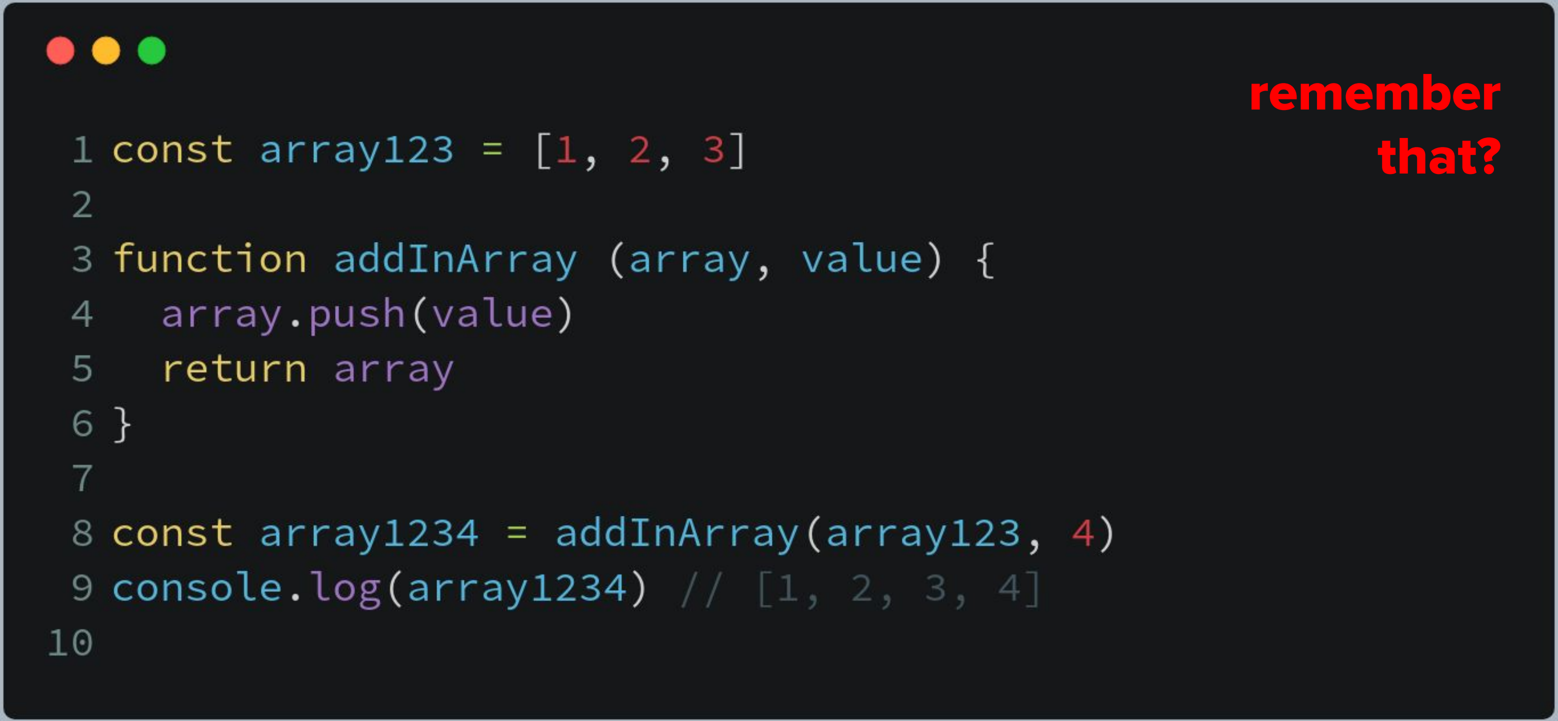


```
const ball = { diameter: 30, shape: 'circle' }

const soccerBall = {
  ...ball,
  diameter: 22
}

console.log(ball) // { diameter: 30, shape: 'circle' }
console.log(soccerBall) // { diameter: 22, shape: 'circle' }

console.log(ball === soccerBall) // false
```

The **same** goes for **any** kind of **object**

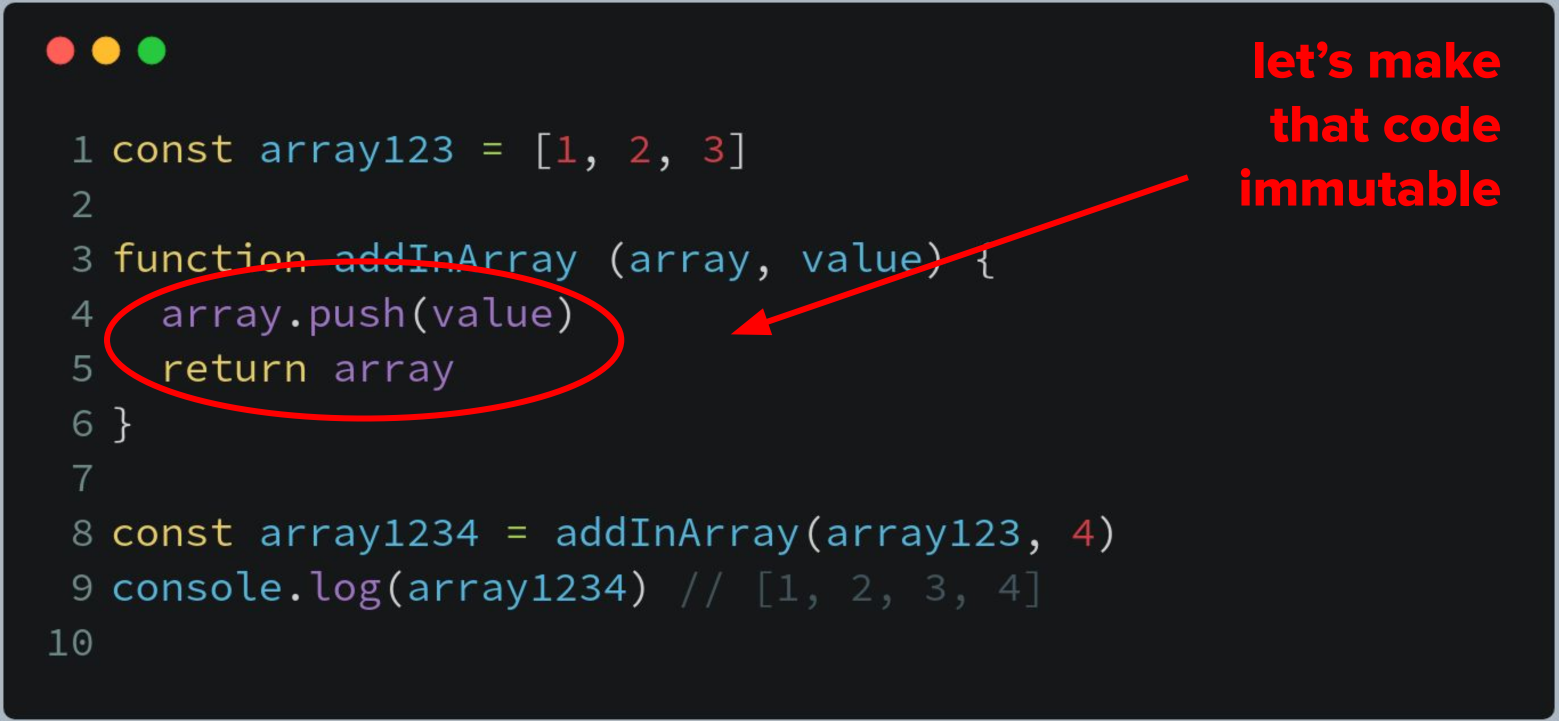


```javascript
const array123 = [1, 2, 3]

function addInArray (array, value) {
  array.push(value)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]

```

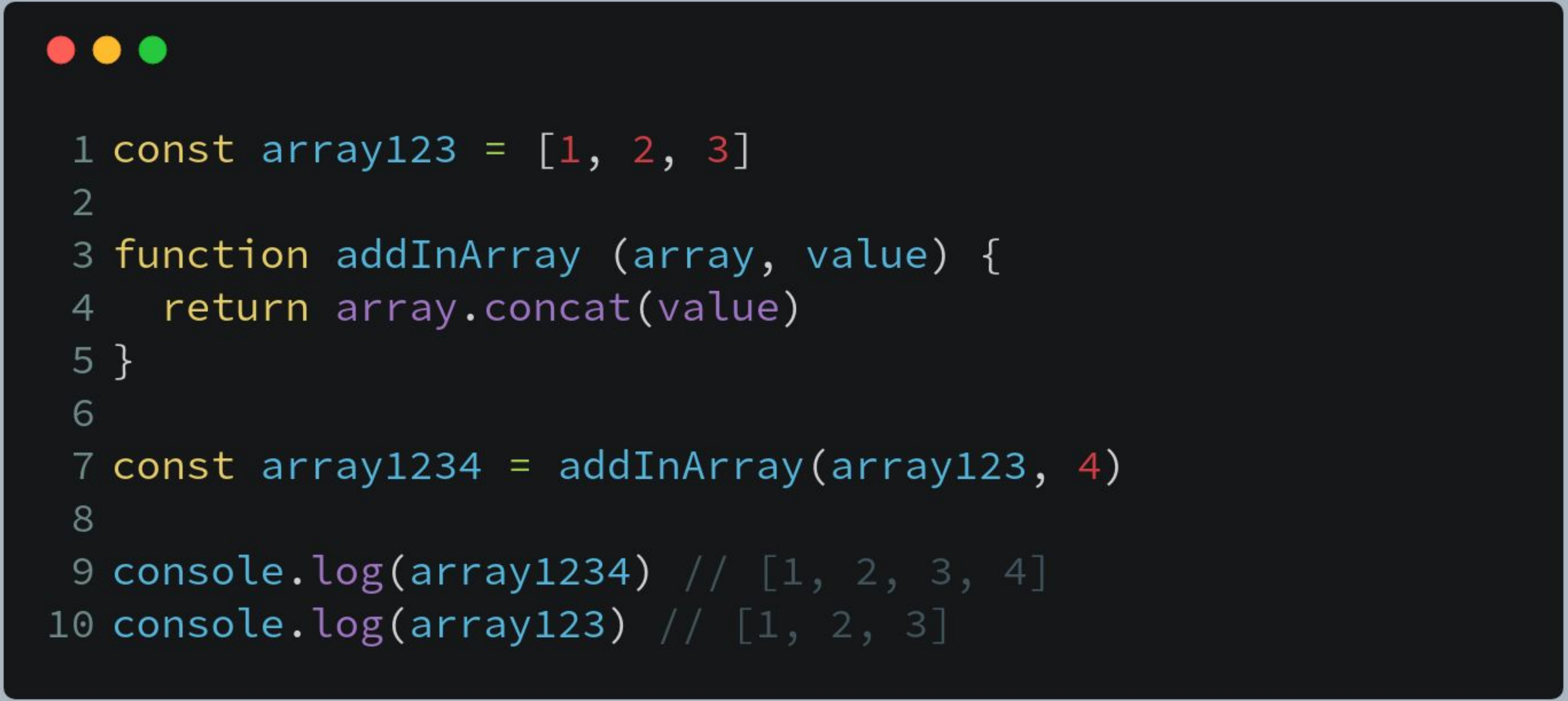


```javascript
const array123 = [1, 2, 3]

function addInArray (array, value) {
  array.push(value)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]
```

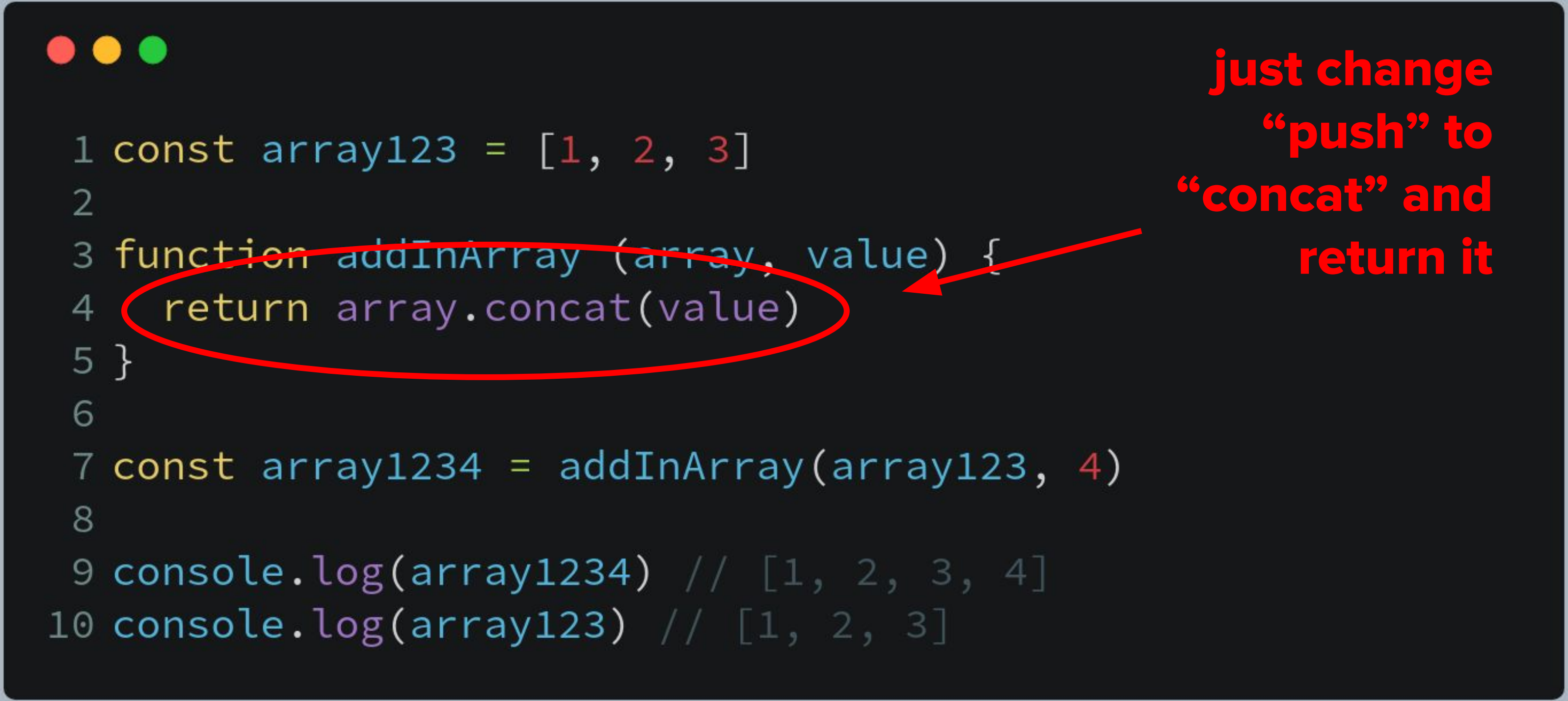
remember that?



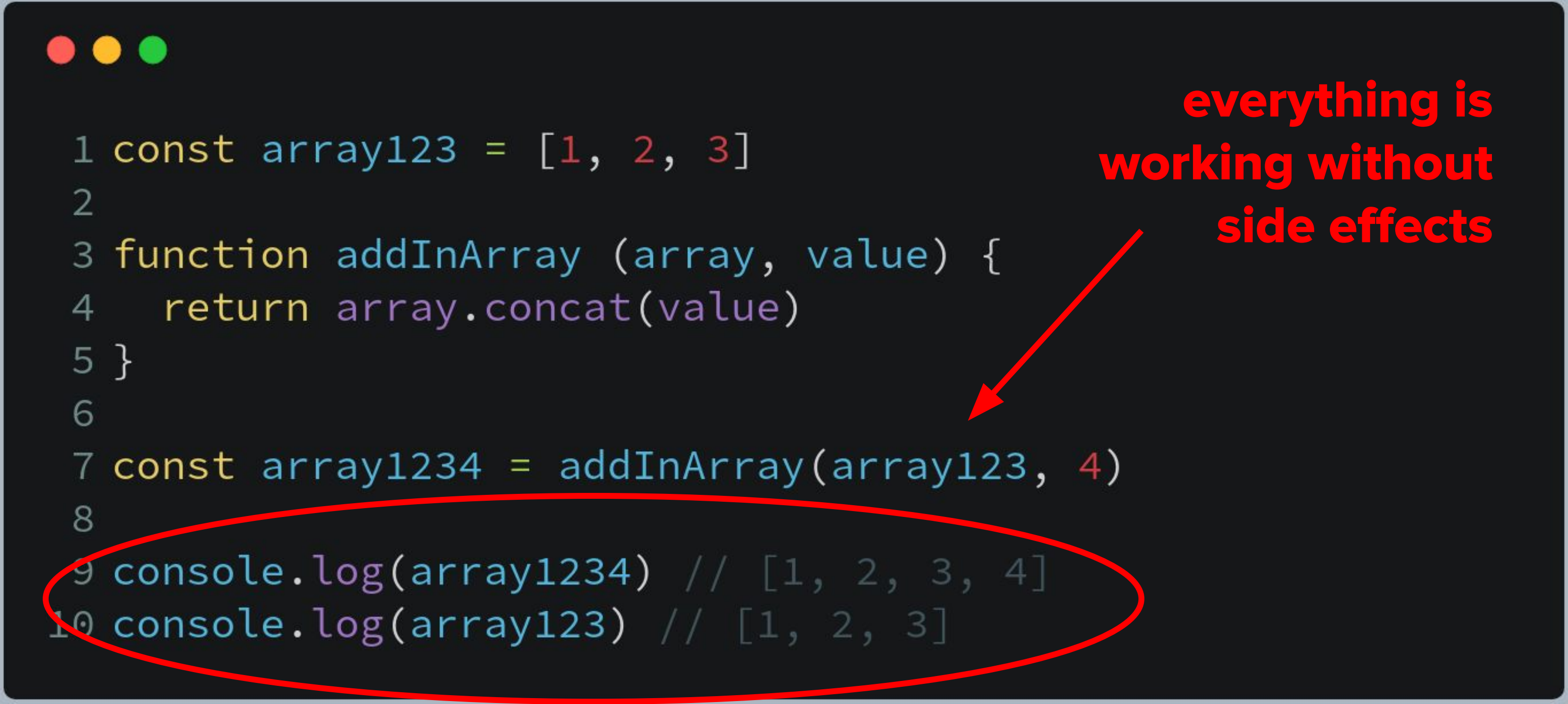
```
const array123 = [1, 2, 3]

function addInArray (array, value) {
  array.push(value)
  return array
}

const array1234 = addInArray(array123, 4)
console.log(array1234) // [1, 2, 3, 4]
```



```javascript
const array123 = [1, 2, 3]

function addInArray (array, value) {
  return array.concat(value)
}

const array1234 = addInArray(array123, 4)

console.log(array1234) // [1, 2, 3, 4]
console.log(array123) // [1, 2, 3]
```



```javascript
const array123 = [1, 2, 3]

function addInArray (array, value) {
  return array.concat(value)
}

const array1234 = addInArray(array123, 4)

console.log(array1234) // [1, 2, 3, 4]
console.log(array123) // [1, 2, 3]
```



```
const array123 = [1, 2, 3]

function addInArray (array, value) {
  return array.concat(value)
}

const array1234 = addInArray(array123, 4)

console.log(array1234) // [1, 2, 3, 4]
console.log(array123) // [1, 2, 3]
```

Immutability

in **practice**

Instead of

array.push

use

array.concat

Instead of

array.splice

use

array.slice

Instead of

array.pop and

array.shift

use

array.filter

Create a **new array** before using **mutable** methods...

like
array.sort
and
array.reverse
for example

Instead of

loops (for/while)

use

array methods

array.map

array.filter

array.find

array.reduce

array.some

array.every

Transform **objects** in

arrays

before **iterate:**

Object.keys()

Object.values()

Object.entries()

Or **prefer** to use the new **for of**

Should I use immutability everywhere in my app?

No!

Just **try** to keep

mutable state

isolated

If you find **bugs**, they will **probably** be on **that** state

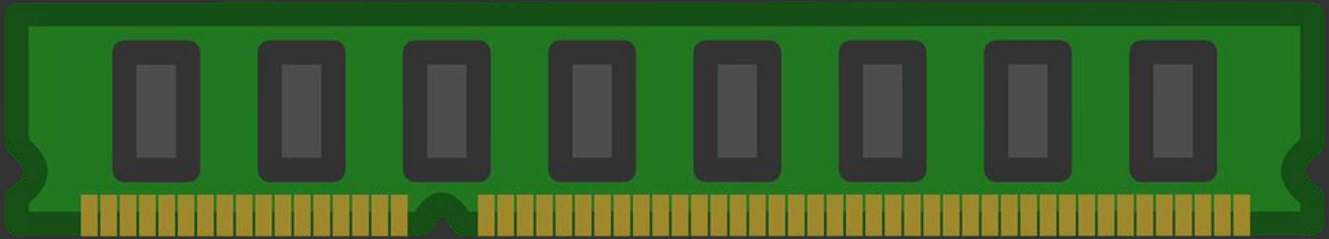
But **creating** more objects spend more **memory...**

What about performance?

Don’t

worry!

Memory chips are cheap



With **little** **data**, it makes **no difference** whether or not to use **immutability**

And
with
lots of
data…

Neither **mutable**

nor **immutable**

code will help you

Worry

about

perceived

performance

Use async code

How to learn more about immutability?

Practicing

a **lot!**

And **getting** involved

with the **community**

/training-center/sobre

/frontendbr/forum

https://bit.ly/frontninja

You will

always

find help

As long as **you** have...

Respect.

Fernando Daciuk

```
$ npm install fdaciuk
```

/fdaciuk/talks