Swipe Left

Mistakes To Avoid As A Javascript Developer

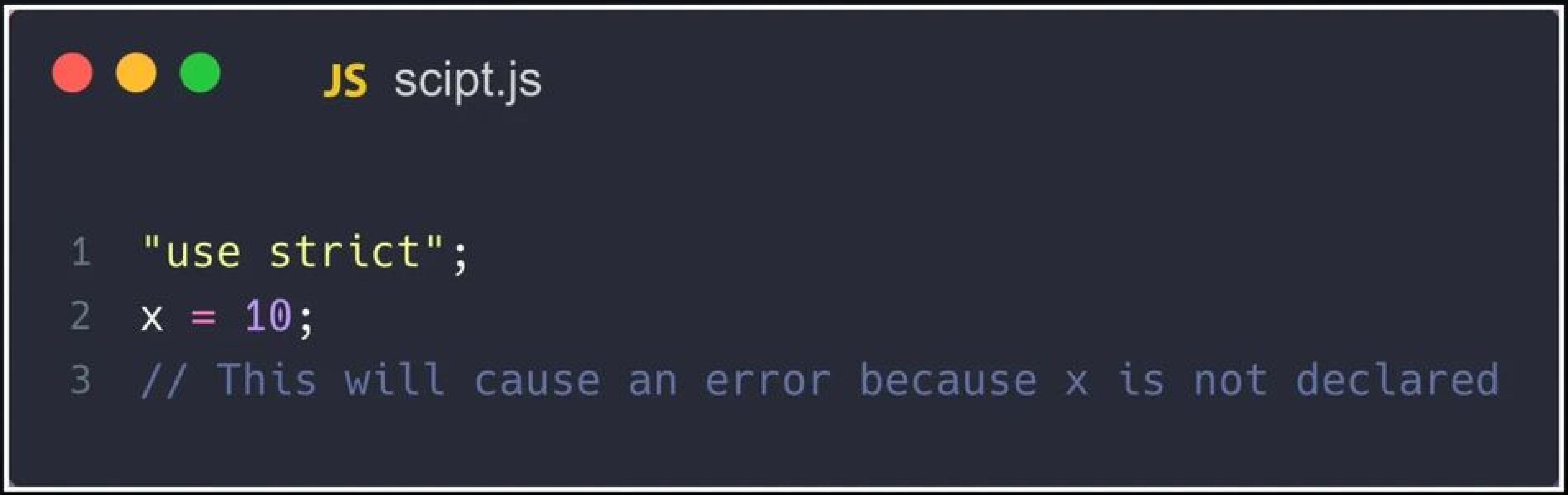
Ariba M.
@frontendcharm

01.

Not Using Strict Mode

Strict mode is a feature in JavaScript that enforces stricter rules for certain behaviours. It can help you avoid common mistakes and ensure that your code is more secure and performant.

For example, using strict mode can prevent you from accidentally creating global variables.



```js
"use strict";
x = 10;
// This will cause an error because x is not declared
```

Ariba M.
@frontendcharm

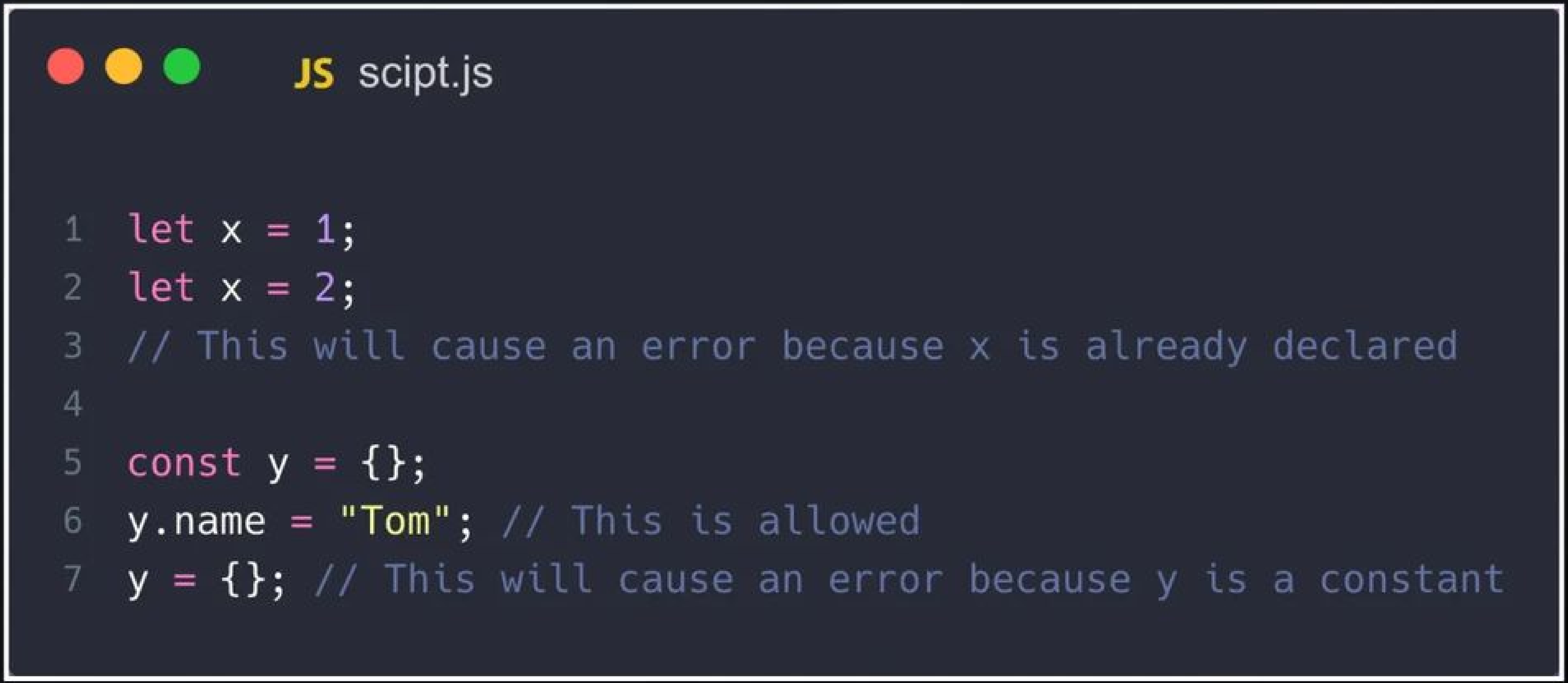
Keep Swiping...

02.

Not Using Let And Const

Using let and const instead of var can help you avoid variable hoisting and prevent accidental re-declarations.

For example :



JS scipt.js

```javascript
let x = 1;
let x = 2;
// This will cause an error because x is already declared

const y = {};
y.name = "Tom"; // This is allowed
y = {}; // This will cause an error because y is a constant
```

Ariba M.
@frontendcharm

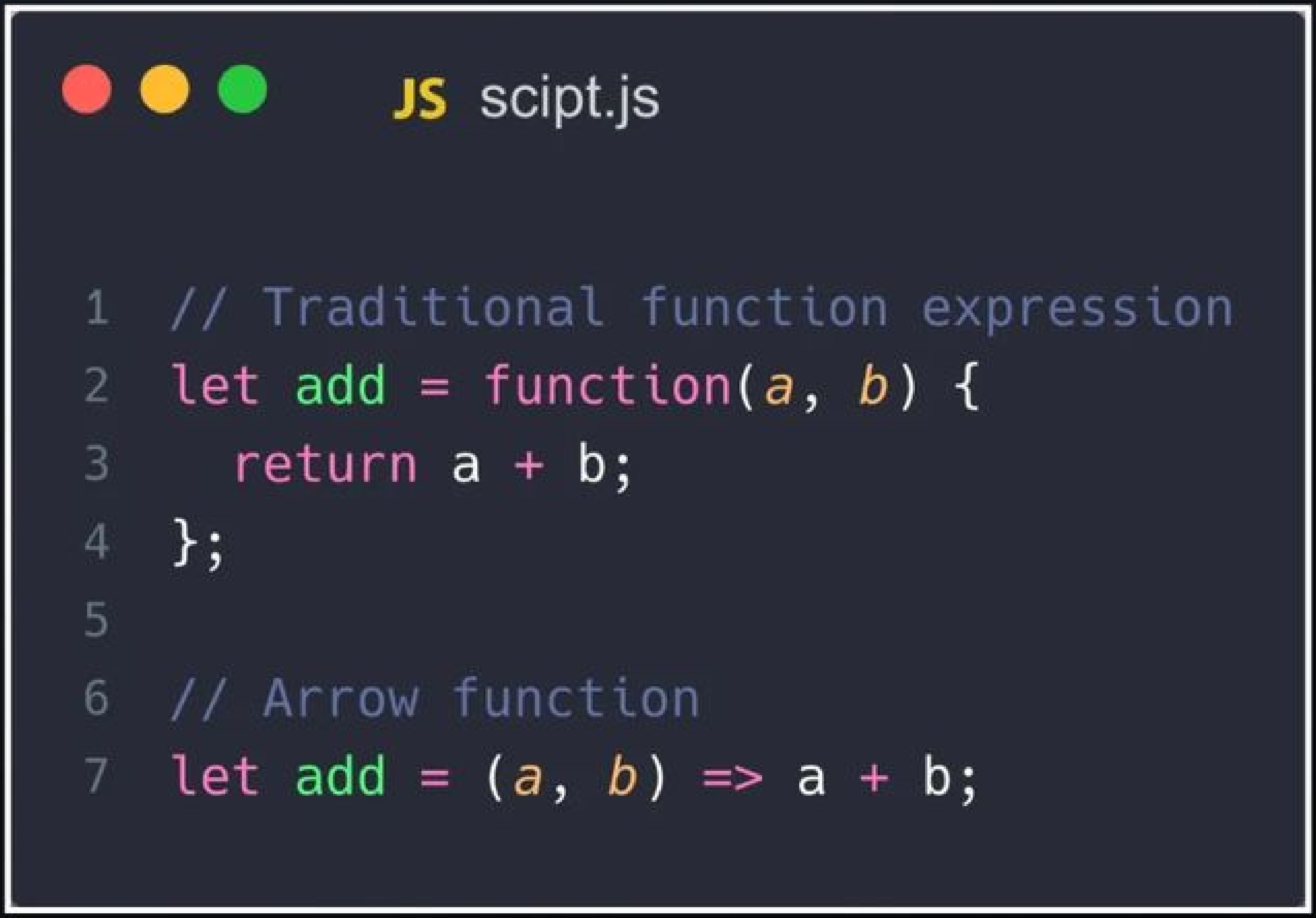
Keep Swiping...

03.

Not Using Arrow Functions

Arrow functions are a shorthand notation for writing function expressions in JavaScript.

They are more concise and easier to read than traditional function expressions and can help improve the performance of your code.



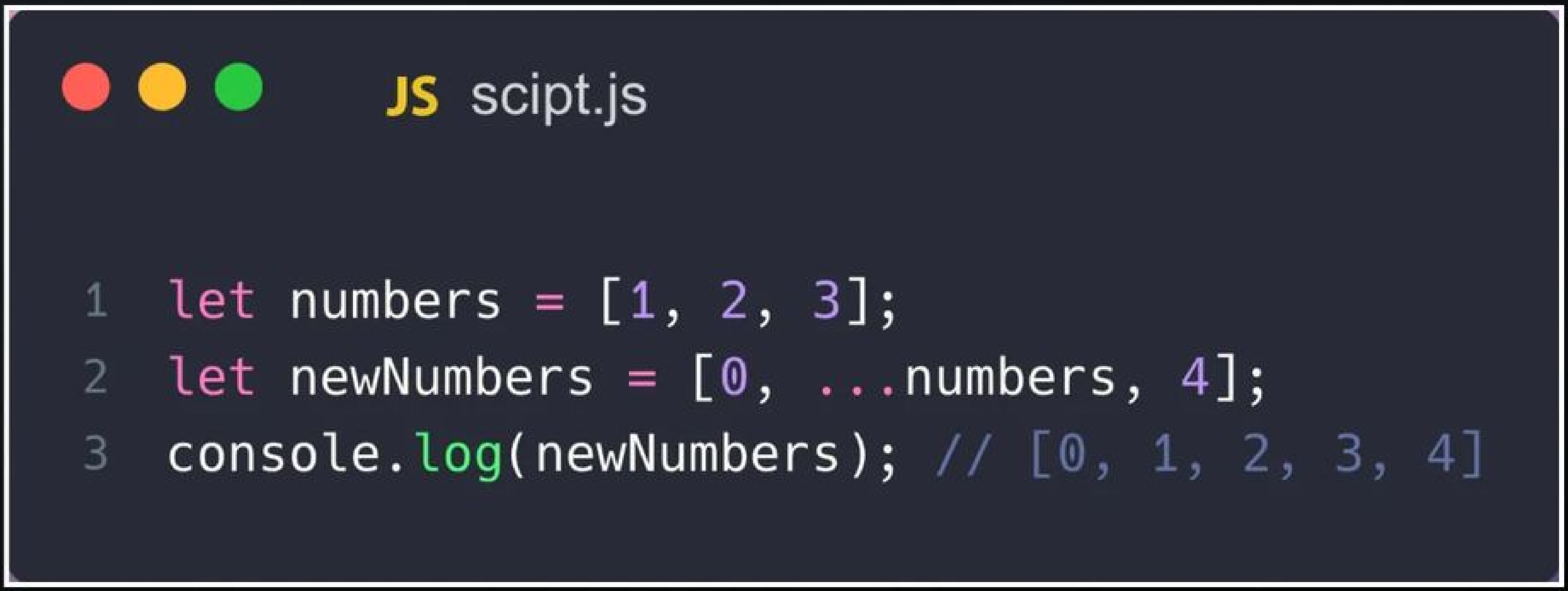
```javascript
// Traditional function expression
let add = function(a, b) {
  return a + b;
};

// Arrow function
let add = (a, b) => a + b;
```

04.

Not Using Spread Operator

The spread operator allows you to expand an array or object into its individual elements.

It can help you write more concise and performant code by avoiding unnecessary loops and operations.



JS scipt.js

```js
let numbers = [1, 2, 3];
let newNumbers = [0, ...numbers, 4];
console.log(newNumbers); // [0, 1, 2, 3, 4]
```

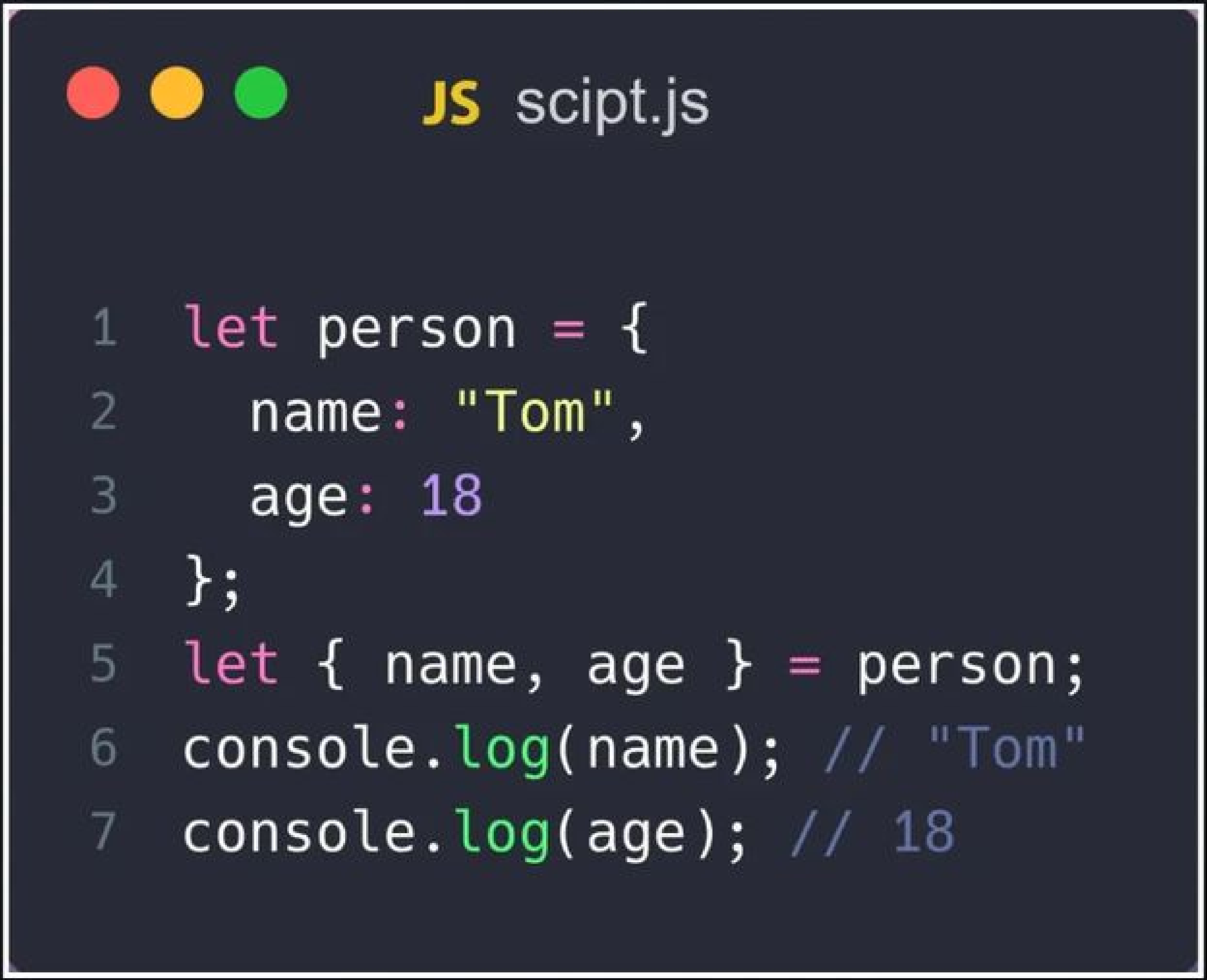
Ariba M.
@frontendcharm

Keep Swiping...

05.

Not Using Destructuring Assignment

Destructuring assignment allows you to extract values from arrays and objects in a more concise and readable way. It can help you write more efficient and maintainable code.


JS scipt.js

```
let person = {
  name: "Tom",
  age: 18
};
let { name, age } = person;
console.log(name); // "Tom"
console.log(age); // 18
```

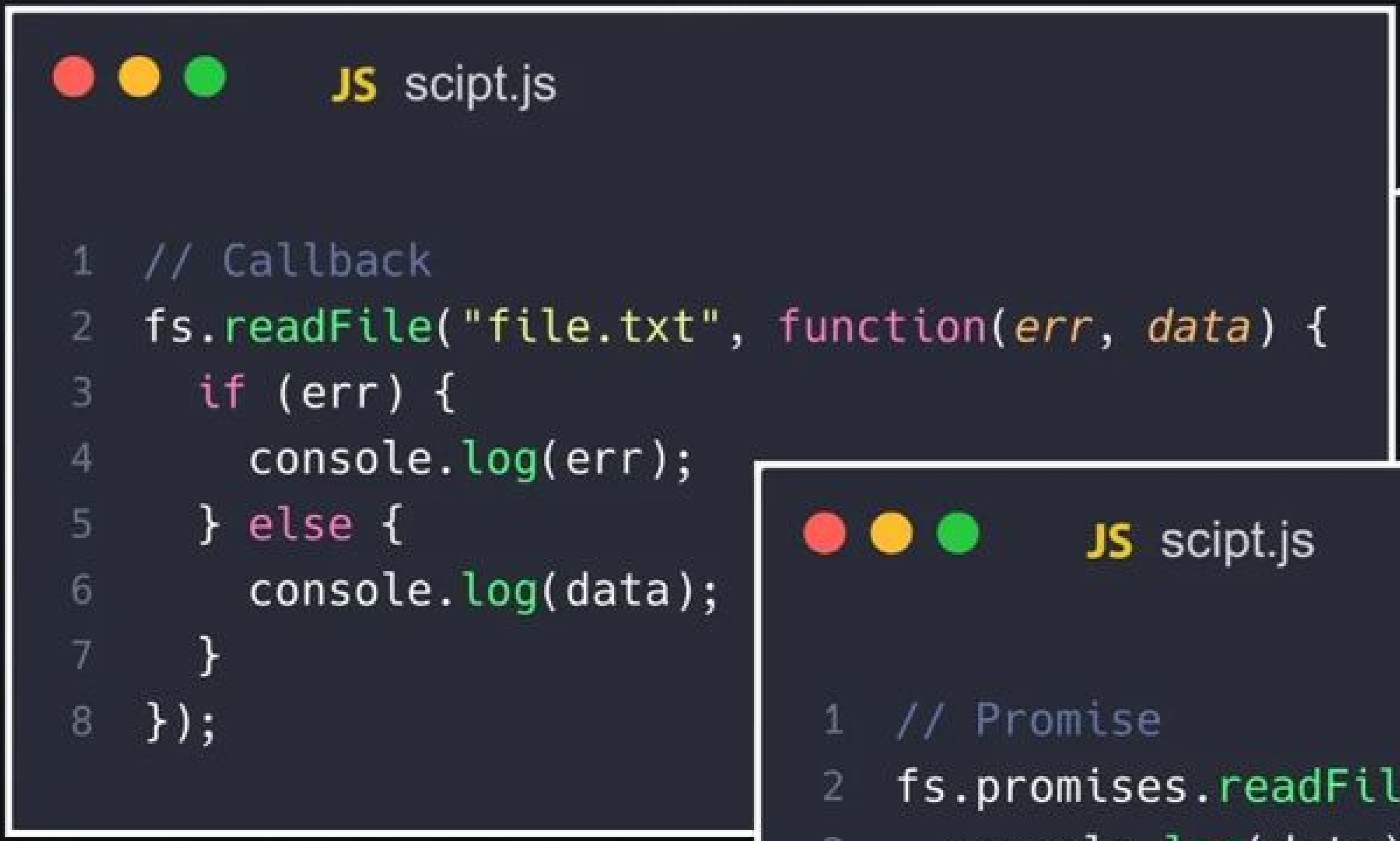
Ariba M.
@frontendcharm

Keep Swiping...

06.

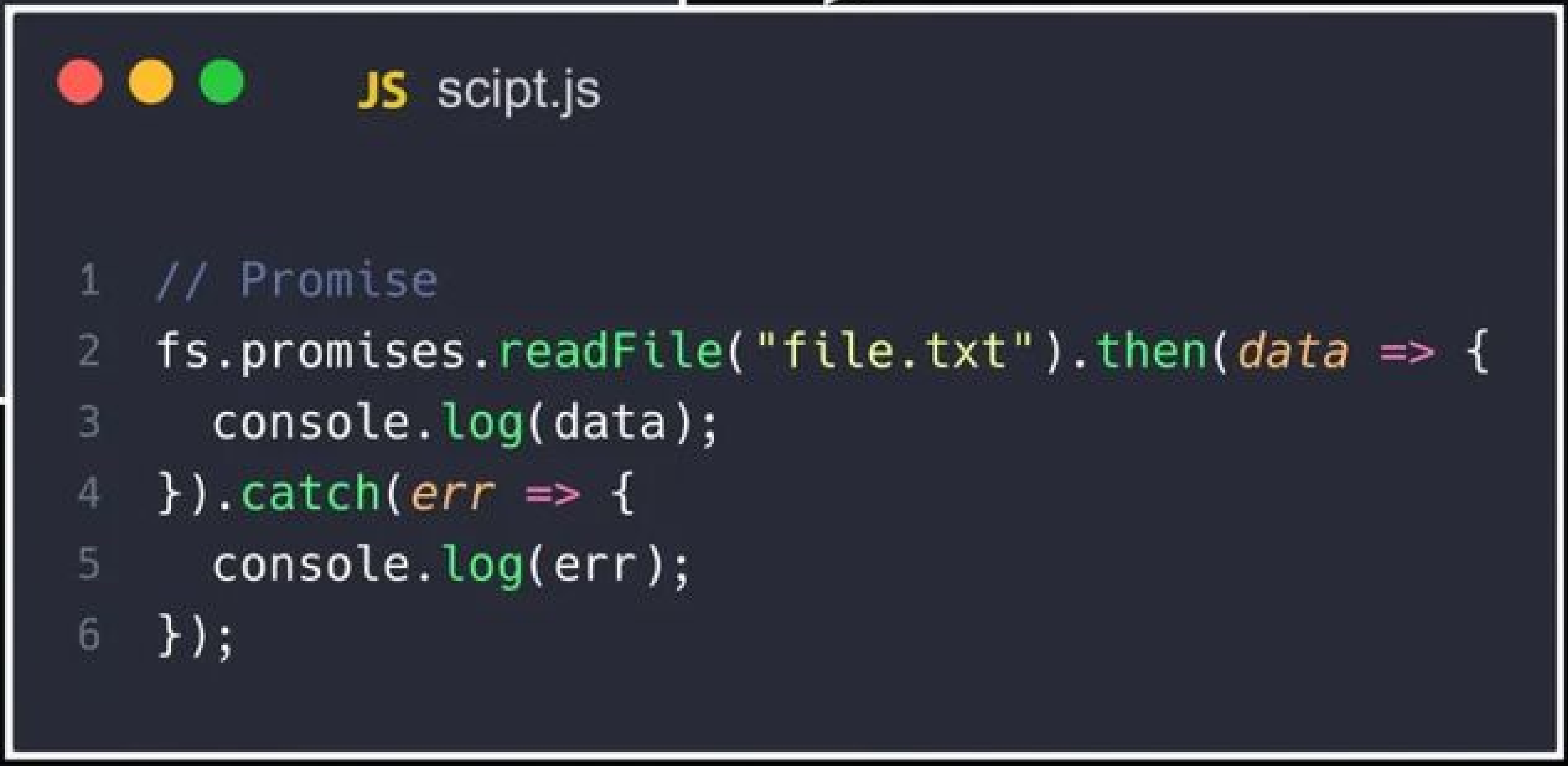
Not Using Promises

Promises are a more modern way of handling asynchronous operations in JavaScript. They can help you write more readable and maintainable code by avoiding callback hell and chaining asynchronous operations.



```js
// Callback
fs.readFile("file.txt", function(err, data) {
  if (err) {
    console.log(err);
  } else {
    console.log(data);
  }
});
```

Instead of callbacks use promises



```js
// Promise
fs.promises.readFile("file.txt").then(data => {
  console.log(data);
}).catch(err => {
  console.log(err);
});
```

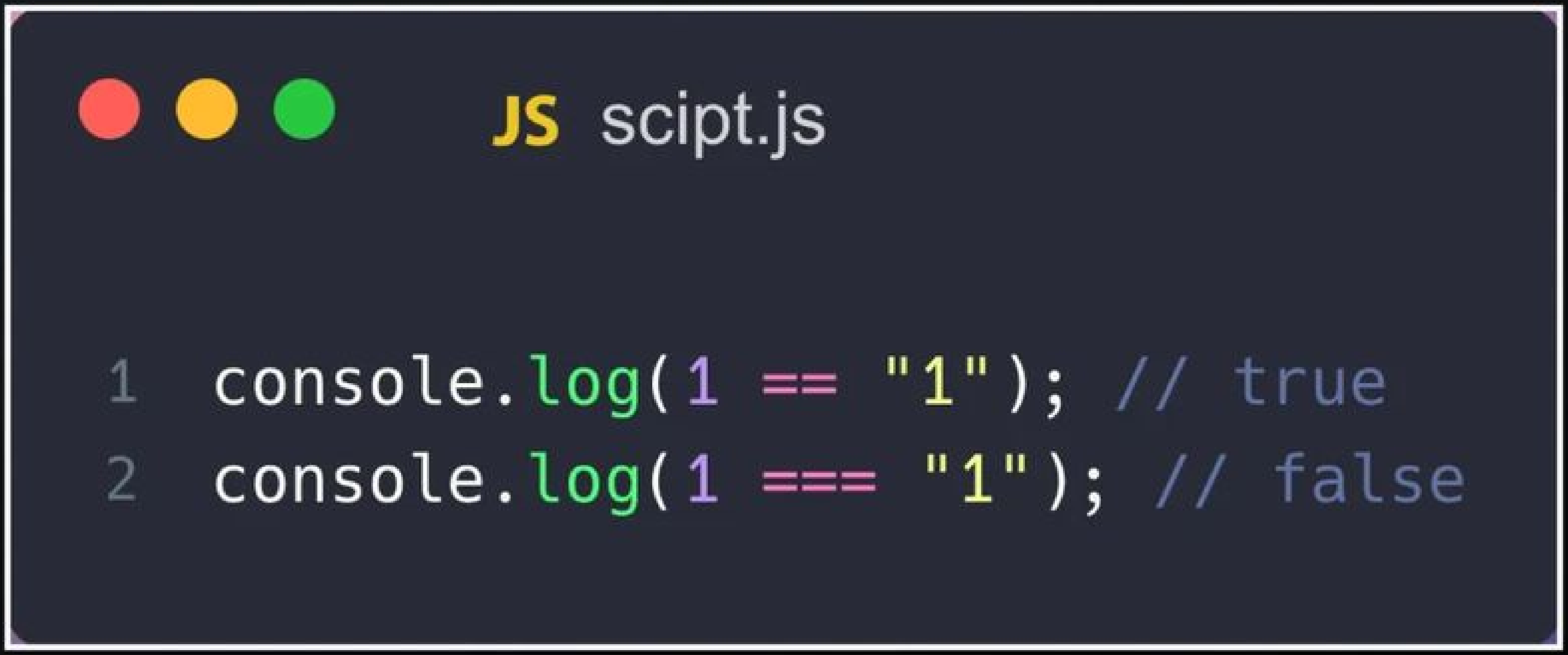
Ariba M.
@frontendcharm

07.

Not Using === Instead Of ==

Strict equality(===) compares values without type coercion, whereas Loose equality(==) compares values after type coercion.

Using strict equality can help you avoid unexpected results and improve the performance of your code.

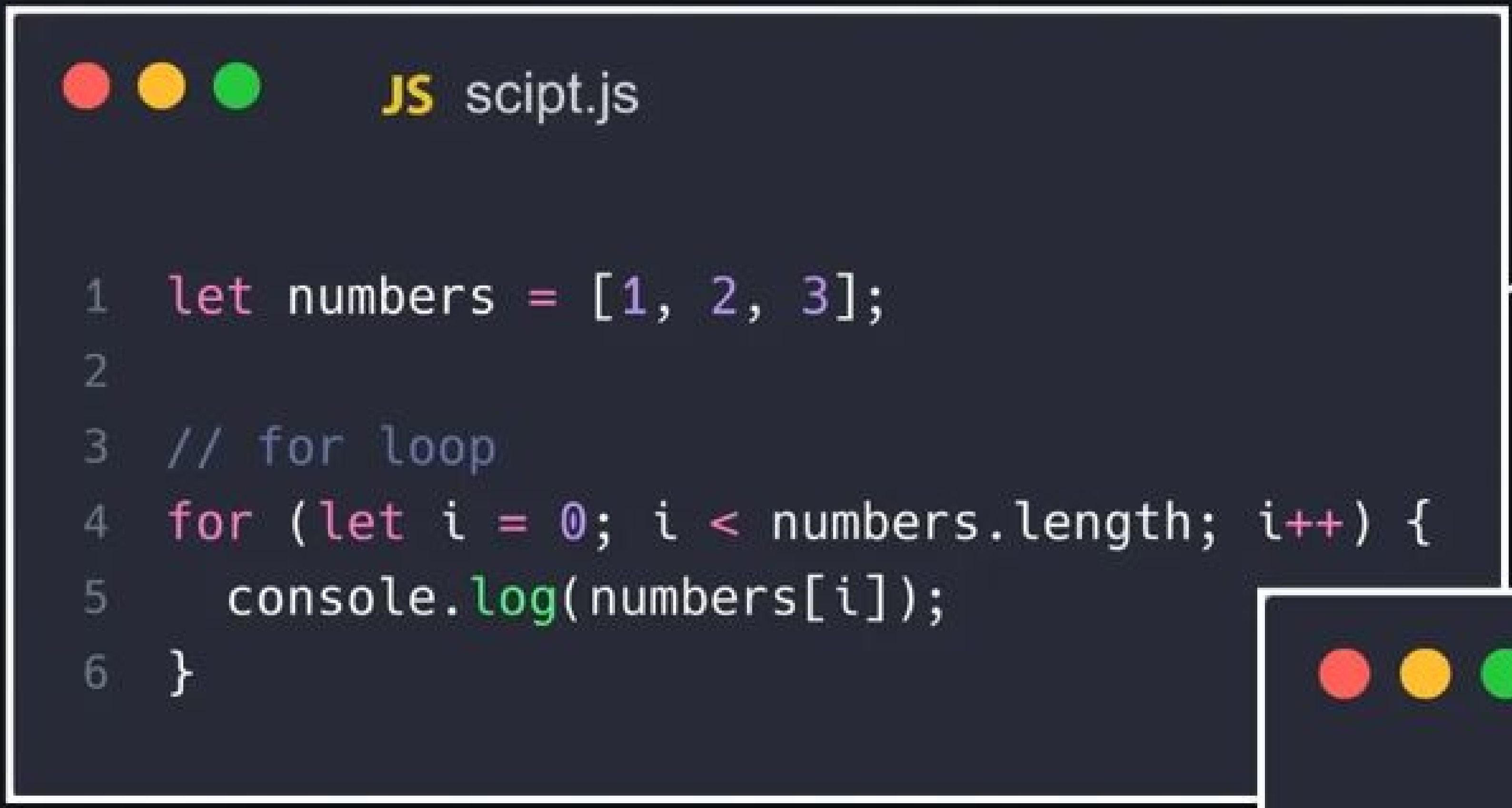


```
console.log(1 == "1"); // true
console.log(1 === "1"); // false
```

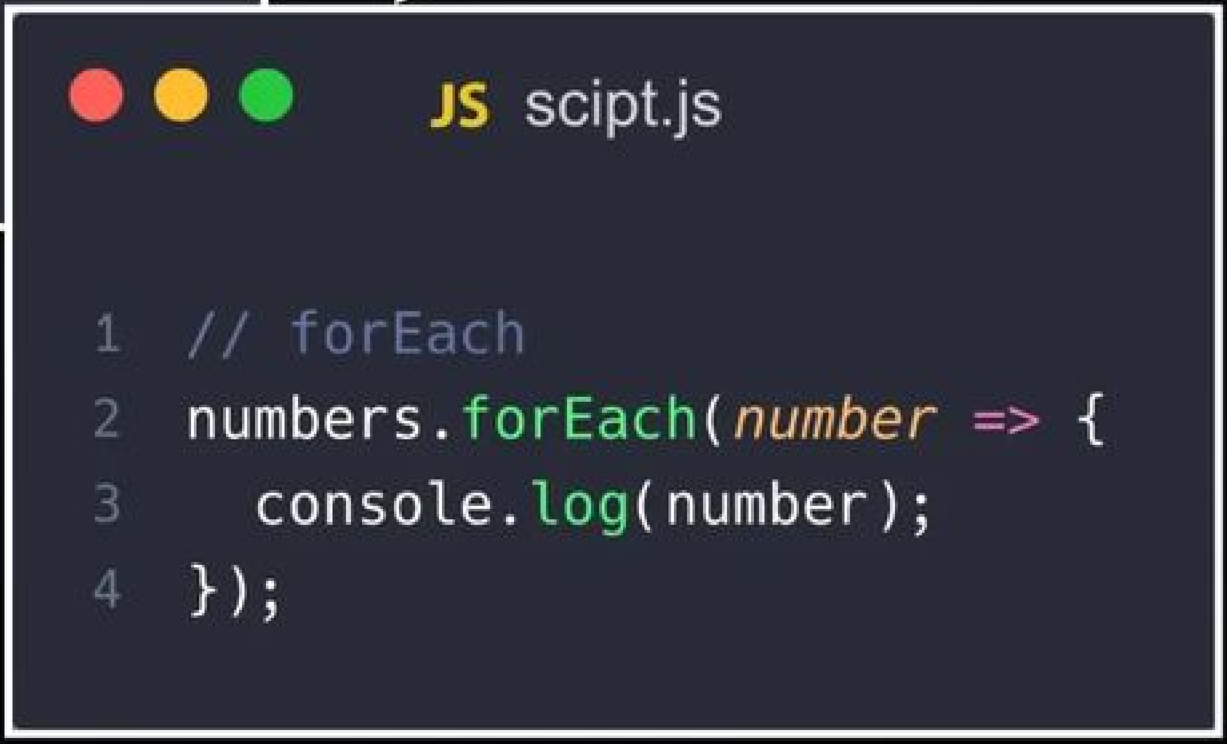
08.

Not Using ForEach

forEach is a higher-order function that can be used to iterate over an array and perform a specific operation on each element. It can help you write more concise and readable code by avoiding for loops and reducing the amount of boilerplate code.



```javascript
let numbers = [1, 2, 3];

// for loop
for (let i = 0; i < numbers.length; i++) {
  console.log(numbers[i]);
}
```



```javascript
// forEach
numbers.forEach(number => {
  console.log(number);
});
```

Turn ON notification

Did you Find it Useful?

@frontendcharm
Ariba

Follow For More !