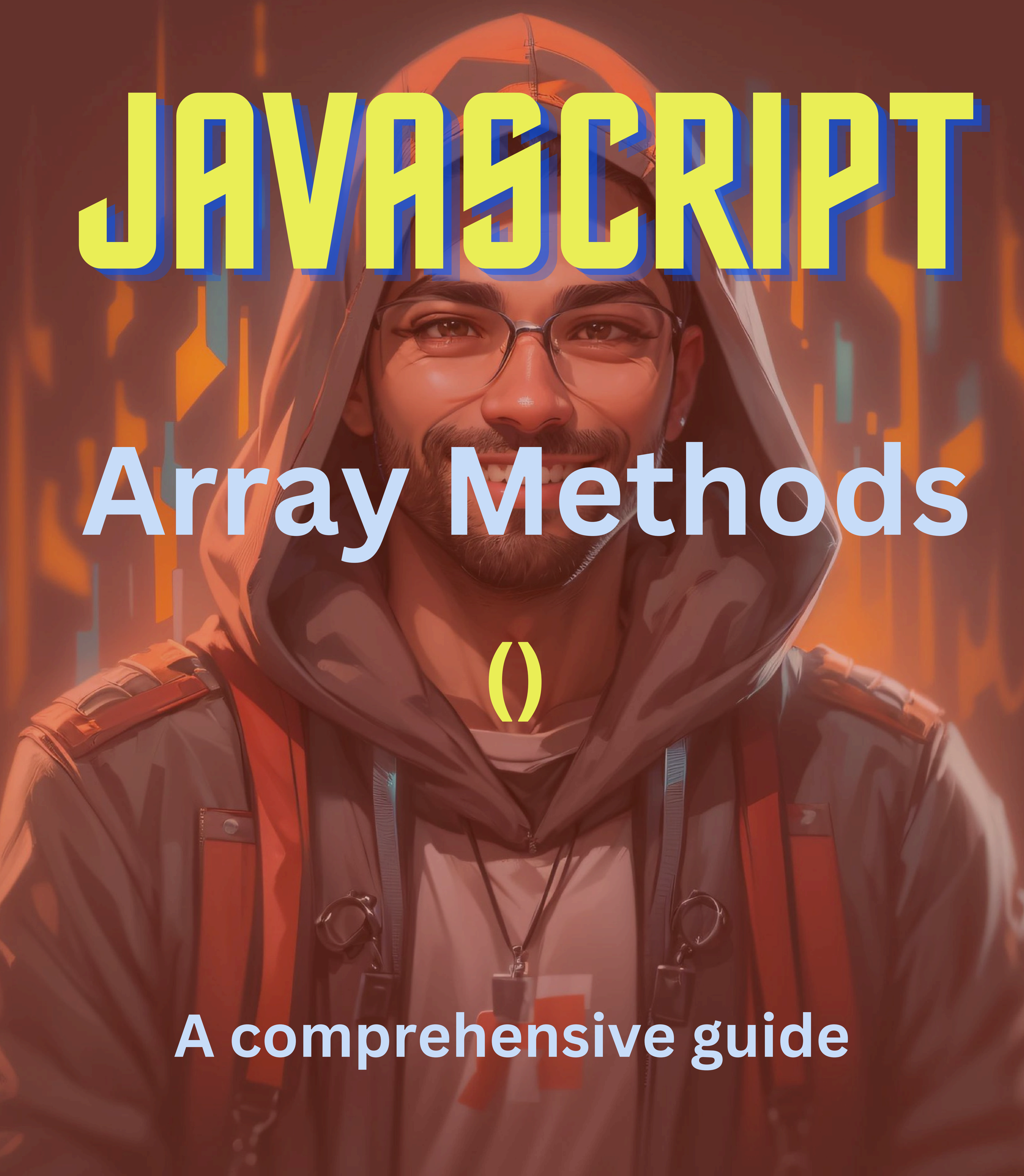
kodemaven-portfolio.vercel.app



JAVASCRIPT

Array Methods

()

A comprehensive guide

Overview

JavaScript arrays are versatile data structures that allow you to store multiple values in a single variable.

Arrays come with a rich set of built-in methods that make it easy to manipulate, search, and transform data.

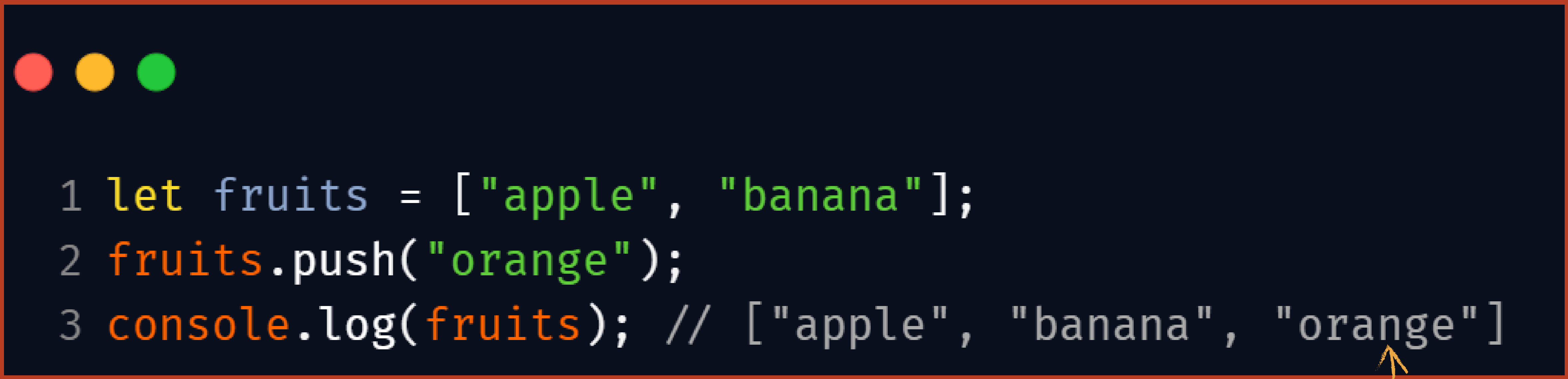
Understanding these methods is key to writing efficient and clean code.

Common Array methods

.push()	.filter()	.every()
. pop()	.reduce()	.split()
.shift()	.find()	.reverse()
.unshift()	.findIndex()	.join()
.slice()	.includes()	
.splice()	.concat()	
.forEach()	.sort()	
.map()	.some()	

.push() & .pop()

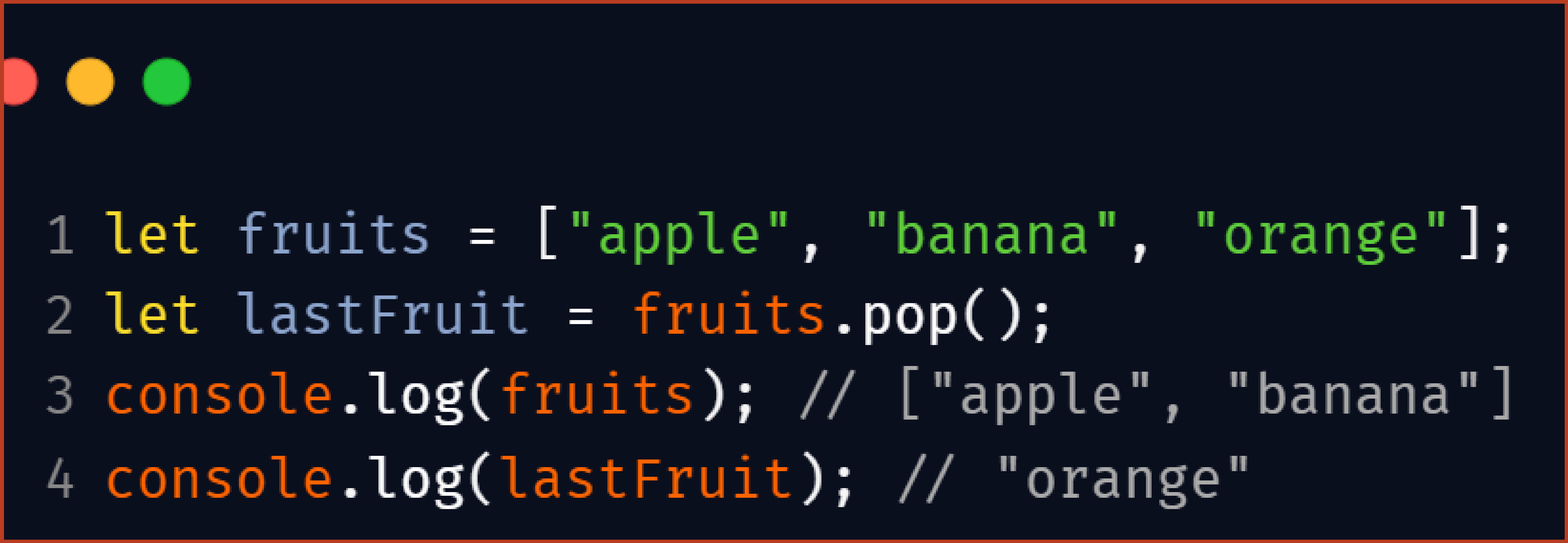
.push(): Adds one or more elements to the **end** of an array and returns the new length of the array.



```javascript
let fruits = ["apple", "banana"];
fruits.push("orange");
console.log(fruits); // ["apple", "banana", "orange"]
```

orange gets added to the array

.pop(): Removes the **last** element from an array and returns that element.



```javascript
let fruits = ["apple", "banana", "orange"];
let lastFruit = fruits.pop();
console.log(fruits); // ["apple", "banana"]
console.log(lastFruit); // "orange"
```

.shift() & .unshift()

.shift(): Removes the **first** element from an array and returns that element.

```javascript
let fruits = ["apple", "banana", "orange"];
let firstFruit = fruits.shift();
console.log(fruits); // ["banana", "orange"]
console.log(firstFruit); // "apple"
```

apple removed from the beginning of the array

.unshift(): Adds one or more elements to the beginning of an array and returns the new length of the array.

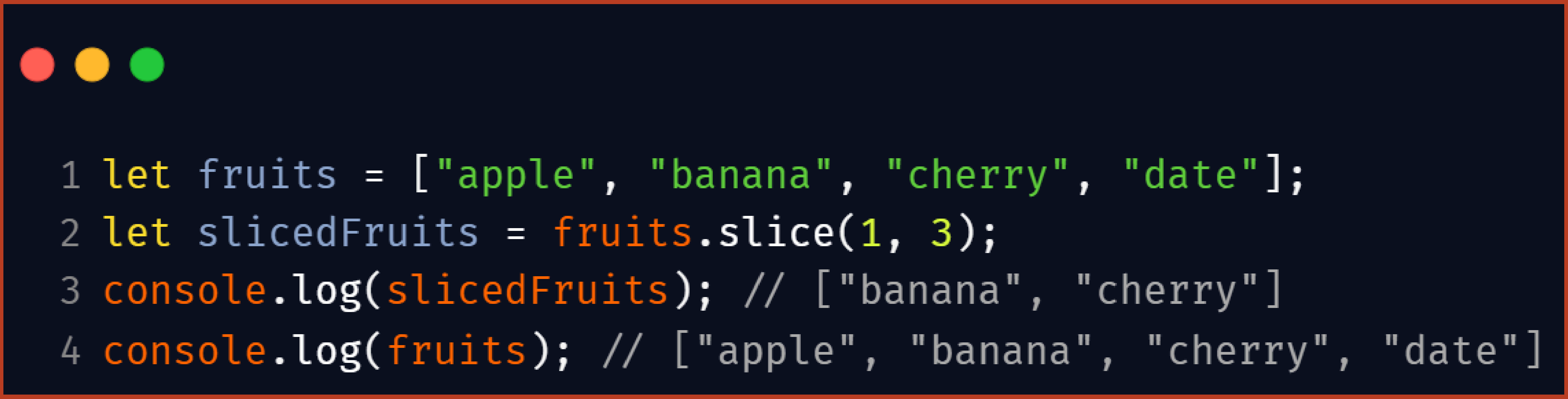
```javascript
let fruits = ["banana", "orange"];
fruits.unshift("mango");
console.log(fruits); // ["mango", banana", "orange"]
```

mango added to the beginning of the array

kodemaven-portfolio.vercel.app

.slice()

.slice(): Returns a shallow copy of a portion of an array into a new array object. It does not modify the original array.



```
let fruits = ["apple", "banana", "cherry", "date"];
let slicedFruits = fruits.slice(1, 3);
console.log(slicedFruits); // ["banana", "cherry"]
console.log(fruits); // ["apple", "banana", "cherry", "date"]
```

The first argument specifies the starting index, and the second argument specifies the ending index (not included in the result).

.splice()

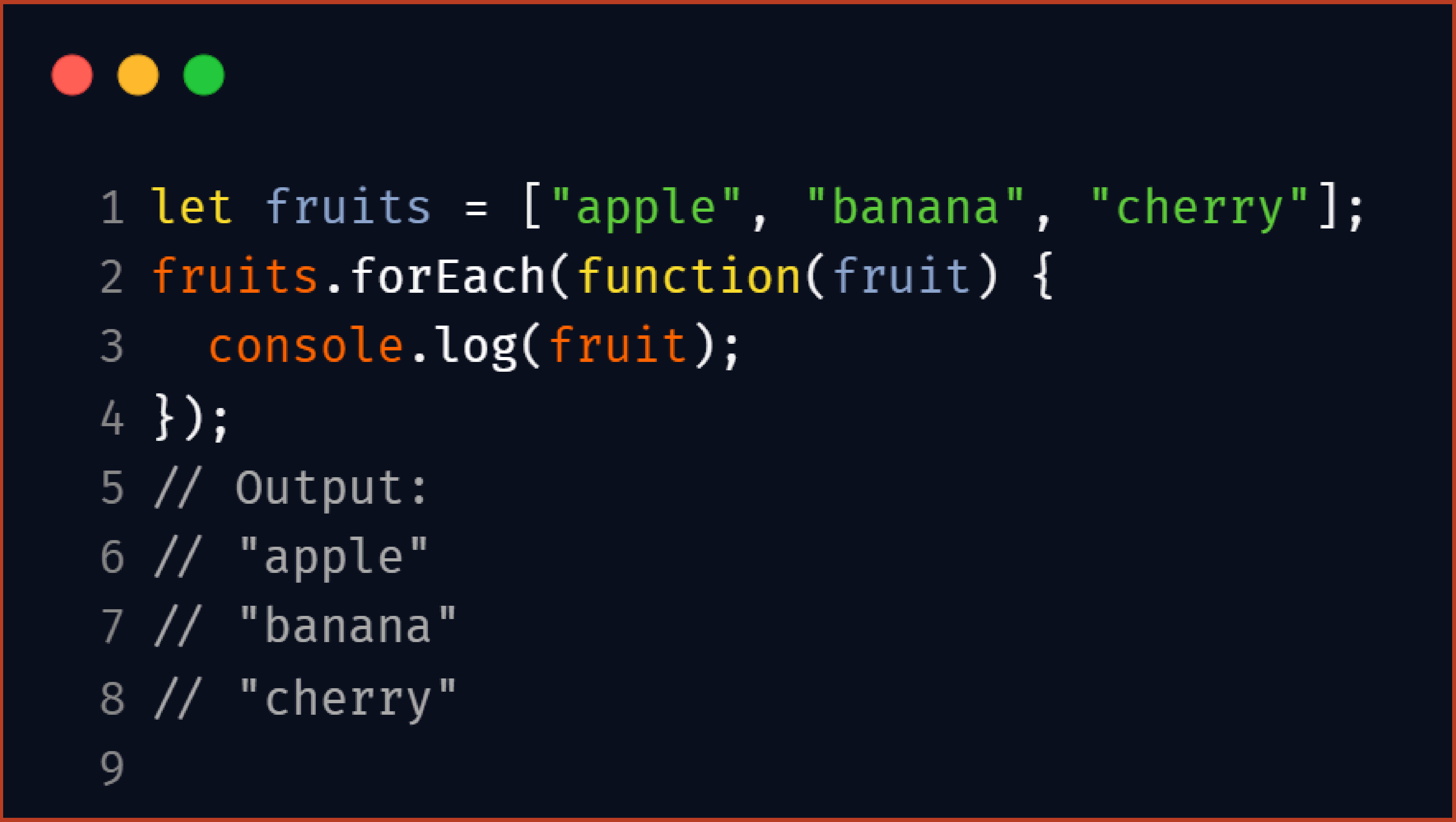
.splice(): Changes the contents of an array by removing, replacing, or adding elements.

```javascript
let fruits = ["apple", "banana", "cherry"];
fruits.splice(1, 1, "orange");
console.log(fruits); // ["apple", "orange", "cherry"]
```

The first argument specifies the index at which to start changing the array, the second argument specifies the number of elements to remove, and the subsequent arguments specify the elements to add.

.forEach()

.forEach(): Executes a provided function once for each array element.



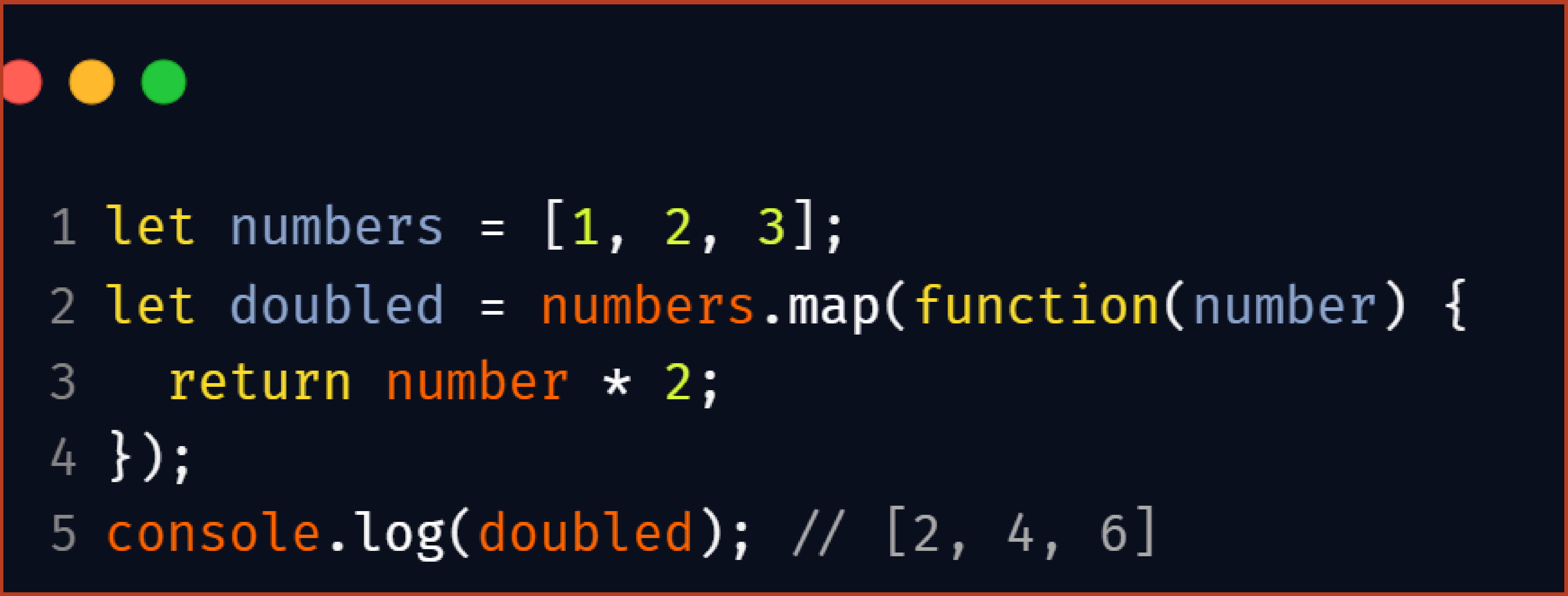
```javascript
let fruits = ["apple", "banana", "cherry"];
fruits.forEach(function(fruit) {
  console.log(fruit);
});
// Output:
// "apple"
// "banana"
// "cherry"

```

This method doesn't return anything; it's mainly used for iterating over arrays.

.map()

.map(): creates a new array by applying a provided function to each element of an existing array. It does not change the original array; instead, it returns a new array where each element is the result of the function applied to the corresponding element in the original array.



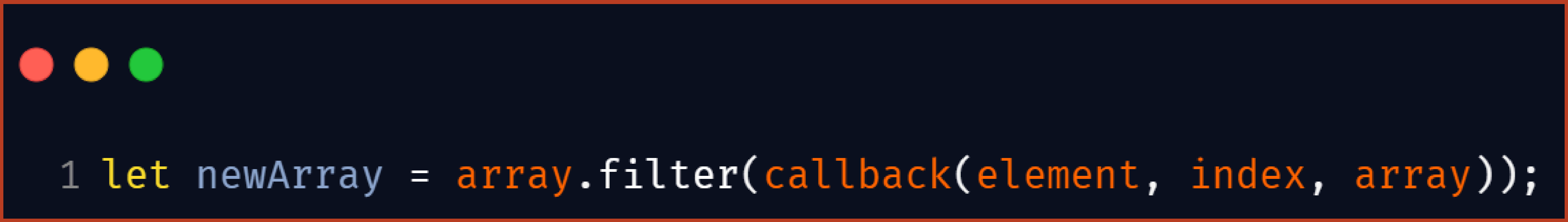
```
let numbers = [1, 2, 3];
let doubled = numbers.map(function(number) {
  return number * 2;
});
console.log(doubled); // [2, 4, 6]
```

Returns a new array, leaving the original array unchanged. This is particularly useful when you need to transform every element in an array.

kodemaven-portfolio.vercel.app

.filter()

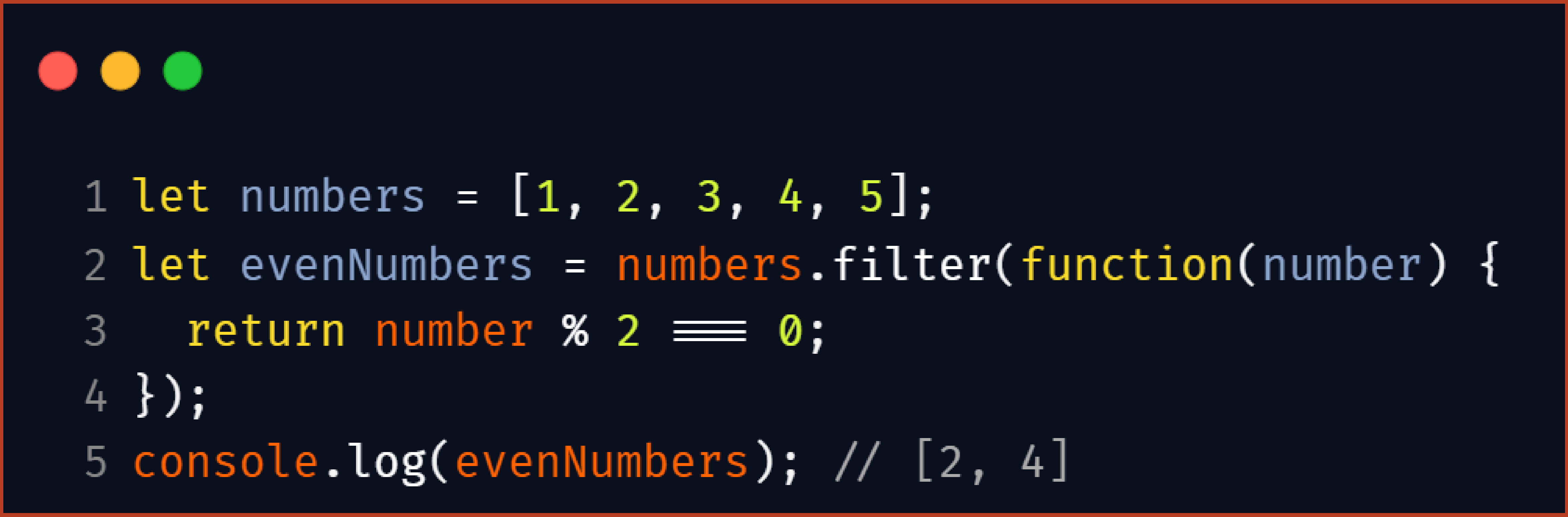
.filter(): Creates a new array filled with elements that pass a test provided by a function. It iterates over each element in the original array, applying the test (callback function) to each element.



```
let newArray = array.filter(callback(element, index, array));
```

- callback: A function that is called for each element in the array. It takes up to three arguments:
- element: The current element being processed in the array.
- index (optional): The index of the current element being processed.
- array (optional): The array filter() was called upon.
- newArray: The new array containing the elements that passed the test.

Example:



```javascript
let numbers = [1, 2, 3, 4, 5];
let evenNumbers = numbers.filter(function(number) {
  return number % 2 === 0;
});
console.log(evenNumbers); // [2, 4]
```

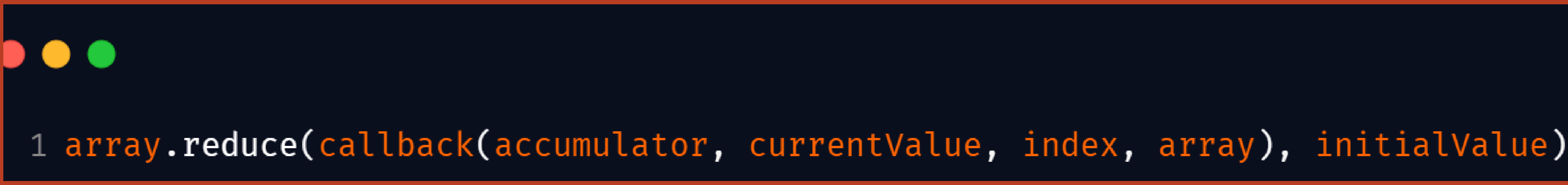
In this example, the filter() method is used to create a new array, evenNumbers, containing only the even numbers from the original numbers array.

The callback function checks if each number is divisible by 2 (number % 2 === 0).

If the condition is true, the number is included in the evenNumbers array.

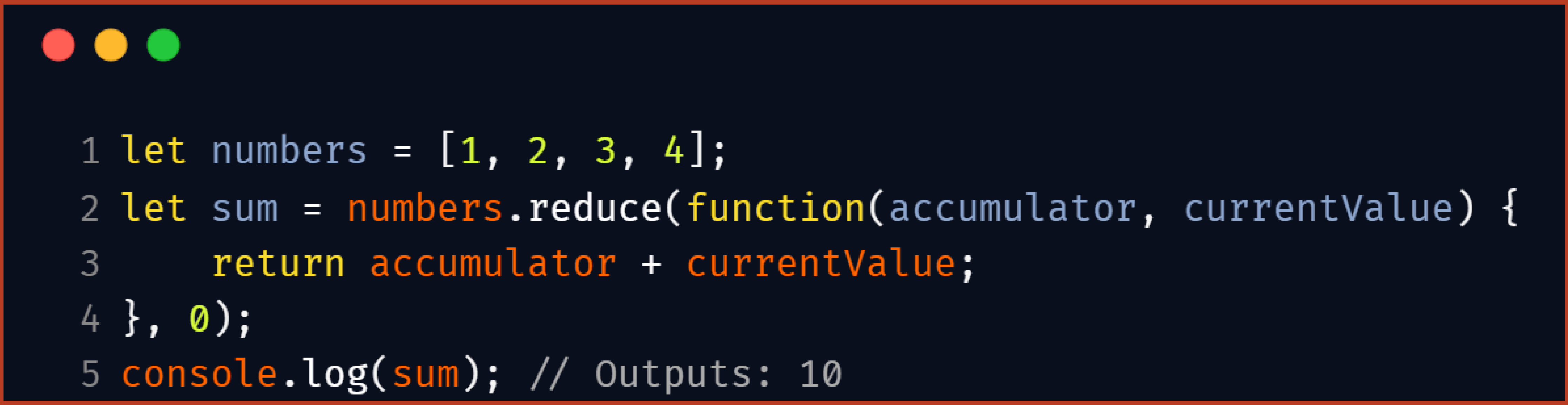
.reduce()

.reduce(): Executes a reducer function on each element of the array, resulting in a single output value. Syntax:



```
array.reduce(callback(accumulator, currentValue, index, array), initialValue)
```

Example:



```
let numbers = [1, 2, 3, 4];
let sum = numbers.reduce(function(accumulator, currentValue) {
    return accumulator + currentValue;
}, 0);
console.log(sum); // Outputs: 10
```

The reduce() method sums up all the elements in the numbers array. The accumulator starts at 0 (provided as the initialValue), and each element is added to this accumulator until all elements have been processed.

.find() & .findIndex()

.find(): Returns the value of the first element in the array that satisfies the provided testing function. If no values satisfy the testing function, undefined is returned.

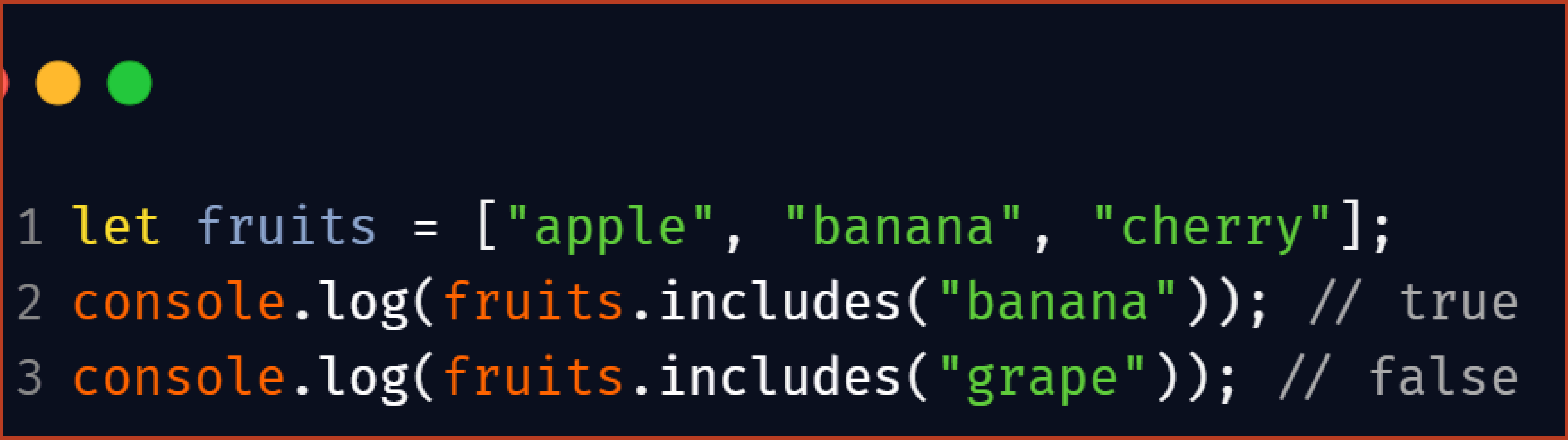
```javascript
let numbers = [1, 2, 3, 4];
let firstEven = numbers.find(function(number) {
  return number % 2 === 0;
});
console.log(firstEven); // 2
```

.findIndex(): Returns the index of the first element in the array that satisfies the provided testing function. If no values satisfy the testing function, -1 is returned.

```javascript
let firstEvenIndex = numbers.findIndex(function(number) {
  return number % 2 === 0;
});
console.log(firstEvenIndex); // 1
```

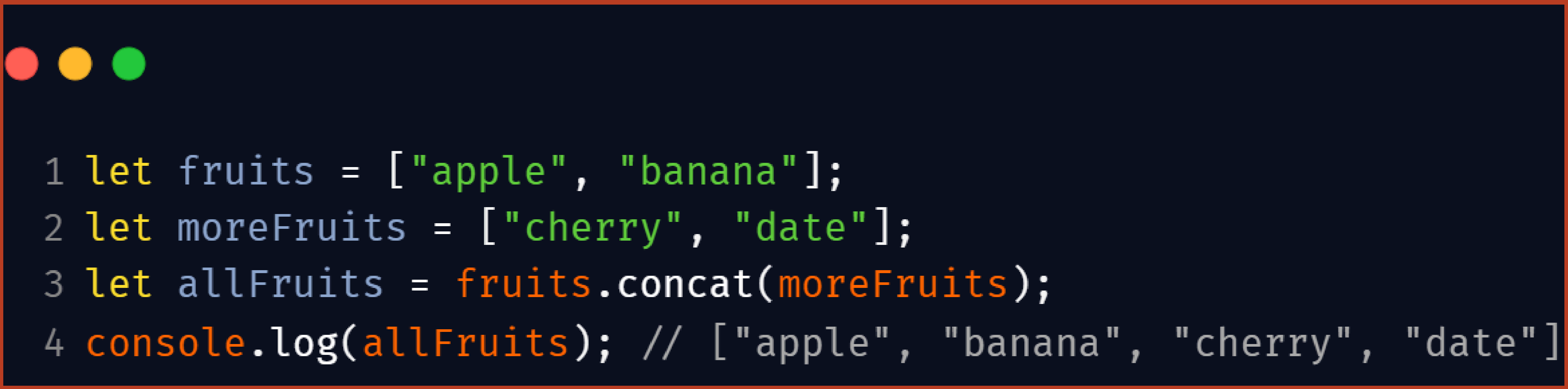
.includes() & .concat()

.includes(): Determines whether an array includes a certain value among its entries, returning true or false.



```javascript
let fruits = ["apple", "banana", "cherry"];
console.log(fruits.includes("banana")); // true
console.log(fruits.includes("grape")); // false
```

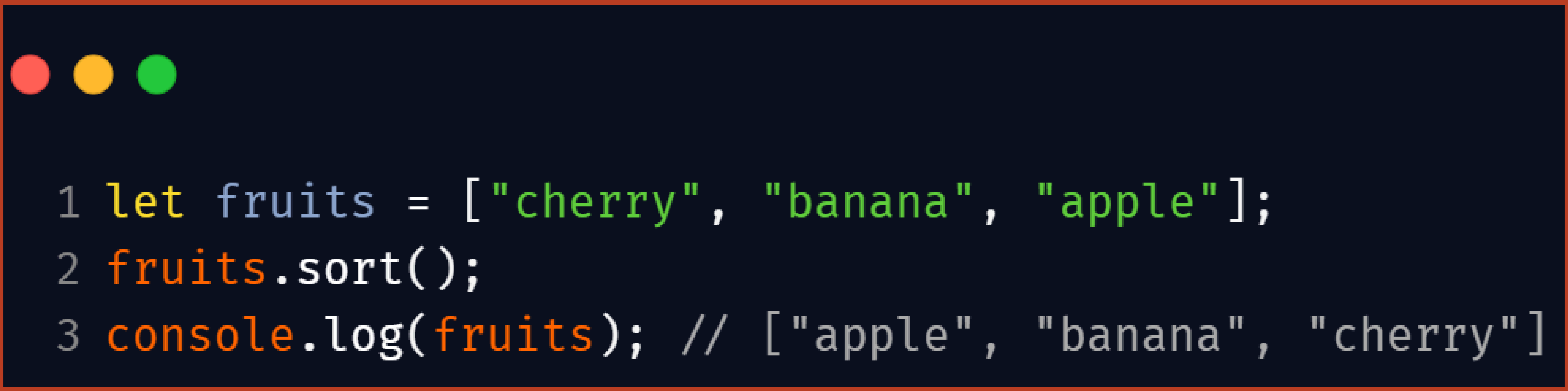
.concat(): Merges two or more arrays into one. It does not change the existing arrays but instead returns a new array.



```javascript
let fruits = ["apple", "banana"];
let moreFruits = ["cherry", "date"];
let allFruits = fruits.concat(moreFruits);
console.log(allFruits); // ["apple", "banana", "cherry", "date"]
```

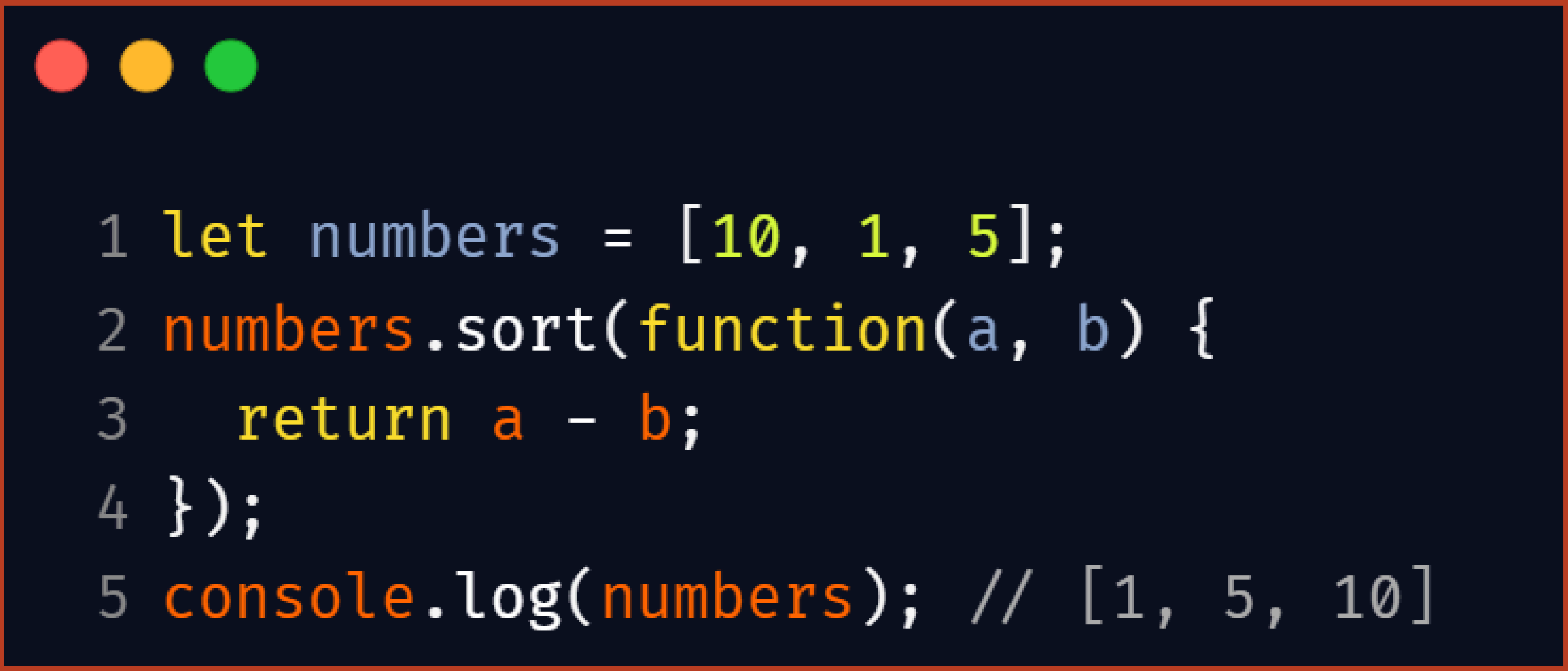
.sort()

.sort(): Sorts the elements of an array in place and returns the array. The default sort order is built upon converting the elements into strings, then comparing their sequences of UTF-16 code units values.



```javascript
let fruits = ["cherry", "banana", "apple"];
fruits.sort();
console.log(fruits); // ["apple", "banana", "cherry"]
```

To sort numbers correctly, you often need to provide a compare function:



```javascript
let numbers = [10, 1, 5];
numbers.sort(function(a, b) {
  return a - b;
});
console.log(numbers); // [1, 5, 10]
```

kodemaven-portfolio.vercel.app

.reverse(), .join(), .split()

.reverse(): Reverses the order of the elements in an array in place and returns the array.

```javascript
let numbers = [1, 2, 3];
numbers.reverse();
console.log(numbers); // [3, 2, 1]
```

.join(): Joins all elements of an array into a string and returns this string. You can specify a separator to insert between the elements.

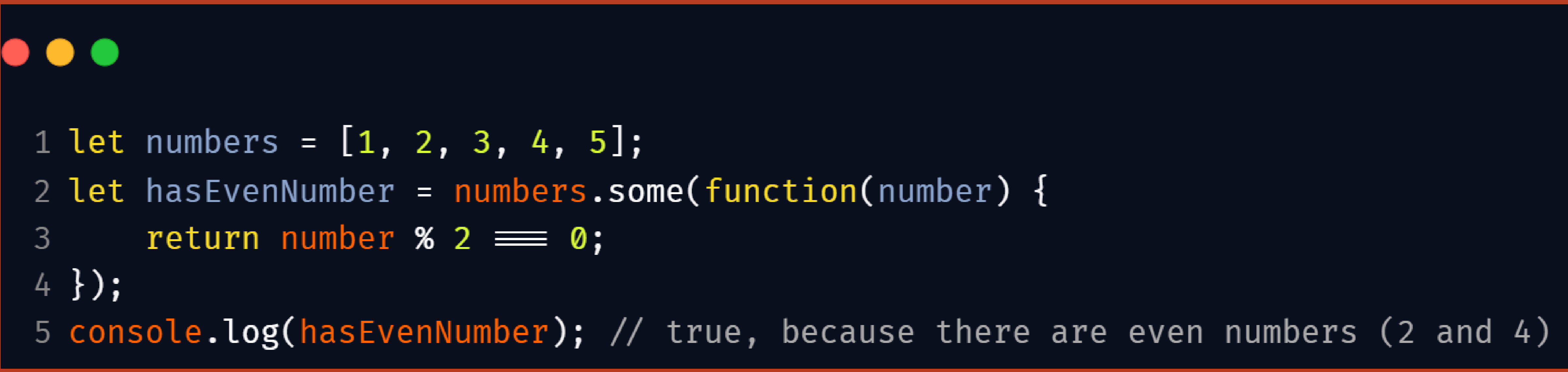
```javascript
let fruits = ["apple", "banana", "cherry"];
let fruitString = fruits.join(", ");
console.log(fruitString); // "apple, banana, cherry"
```

.split(): Splits a string into an array of substrings, and returns the new array. This method is the opposite of join().

```javascript
let fruitString = "apple, banana, cherry";
let fruits = fruitString.split(", ");
console.log(fruits); // ["apple", "banana", "cherry"]
```

.some()

.some(): Checks if at least one element in the array passes a test (provided as a function). If it finds an element for which the provided function returns true, it immediately returns true and stops. Otherwise, it returns false.



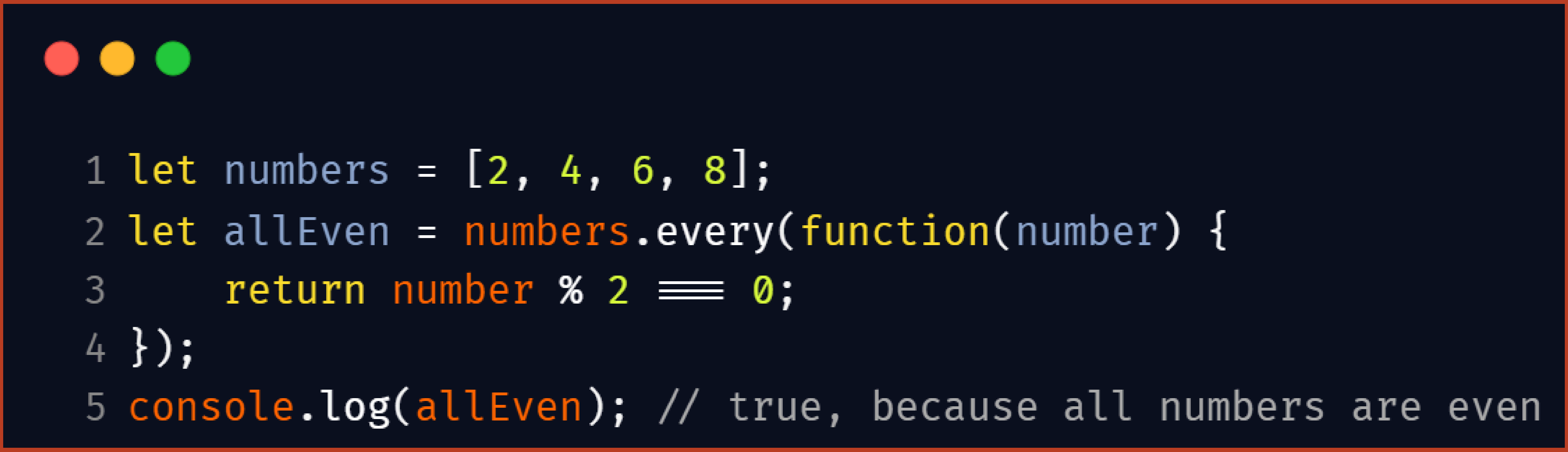
```
let numbers = [1, 2, 3, 4, 5];
let hasEvenNumber = numbers.some(function(number) {
    return number % 2 === 0;
});
console.log(hasEvenNumber); // true, because there are even numbers (2 and 4)
```

In this example, some() checks if there are any even numbers in the numbers array. Since 2 and 4 are even, it returns true.

This is particularly useful when you want to know if any element in an array meets a particular condition.

.every()

.every(): Checks if all elements in the array pass a test (provided as a function). If all elements pass the test, it returns true. If any element fails the test, it returns false and stops.



```javascript
let numbers = [2, 4, 6, 8];
let allEven = numbers.every(function(number) {
    return number % 2 === 0;
});
console.log(allEven); // true, because all numbers are even
```

In this example, every() checks if all the numbers in the array are even. Since they are, it returns true.

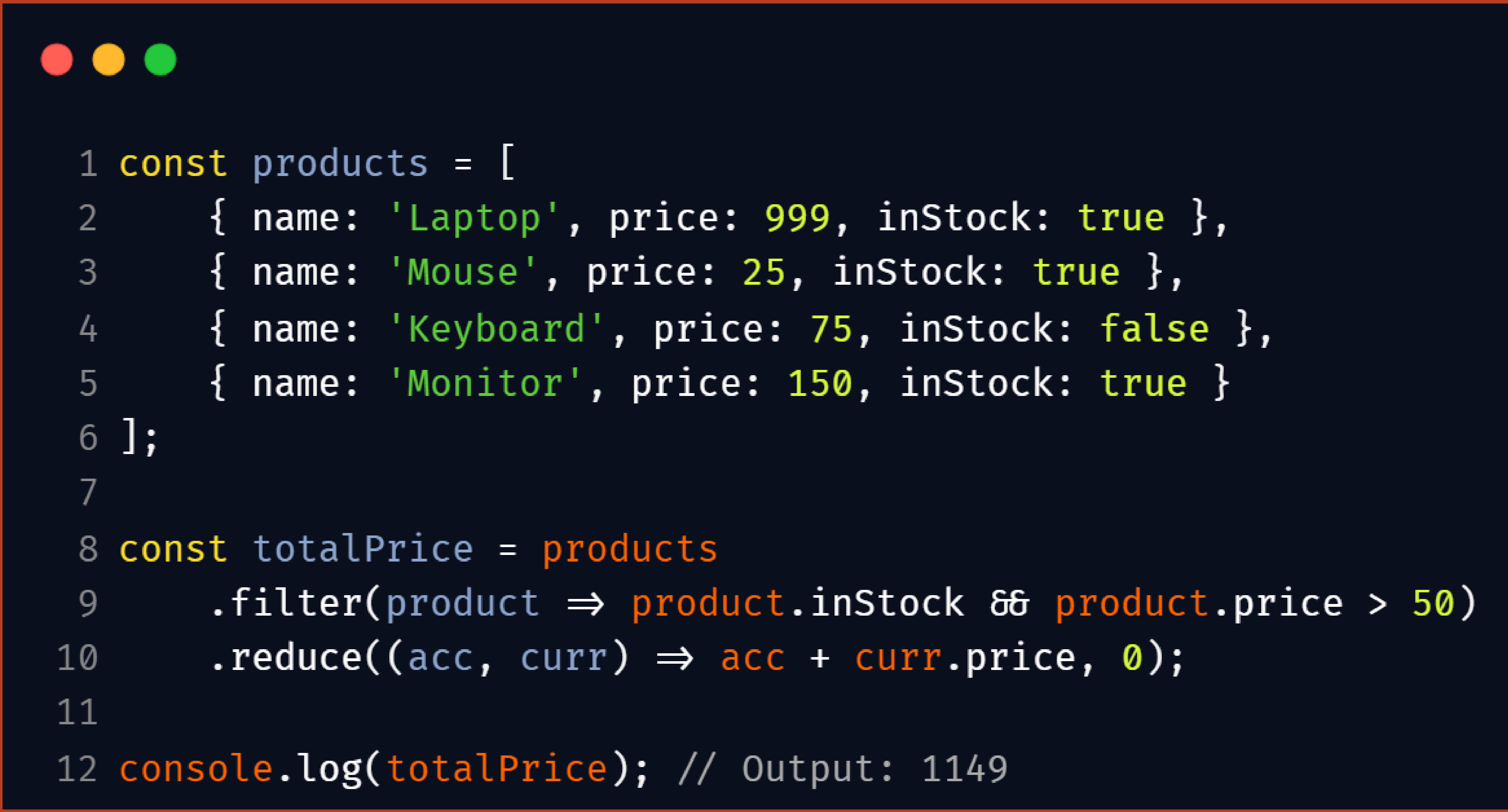
This is particularly useful when you want to know if all element in an array meets a particular condition.

Combining Array Methods for Powerful Data Manipulation

Array methods can be chained together to perform complex data manipulations with minimal code.

Here is an example:

You have an array named products that contains objects representing different products. Each product has a name, price, and inStock status.



```javascript
const products = [
    { name: 'Laptop', price: 999, inStock: true },
    { name: 'Mouse', price: 25, inStock: true },
    { name: 'Keyboard', price: 75, inStock: false },
    { name: 'Monitor', price: 150, inStock: true }
];

const totalPrice = products
    .filter(product => product.inStock && product.price > 50)
    .reduce((acc, curr) => acc + curr.price, 0);

console.log(totalPrice); // Output: 1149
```

Explanation:

- The filter() method is used to create a new array that only includes products that are both in stock (inStock: true) and have a price greater than $50 (price > 50).

- The reduce() method then takes the filtered array and sums up the prices of the remaining products. The acc variable accumulates the total price, starting at 0.

- The totalPrice variable will hold the sum of the prices of the products that passed the filter. In this case, the Laptop ($999) and Monitor ($150) are both in stock and cost more than $50, so their prices are added together.

I hope you found this material useful and helpful.

Remember to:

Like

Save for future reference

&

Share with your network, be helpful to someone 👌

Hi There!

Thank you for reading through

Did you enjoy this knowledge?

💼 Follow my LinkedIn page for more work-life balancing and Coding tips.

🌐 LinkedIn: Oluwakemi Oluwadahunsi