JavaScript Arrays

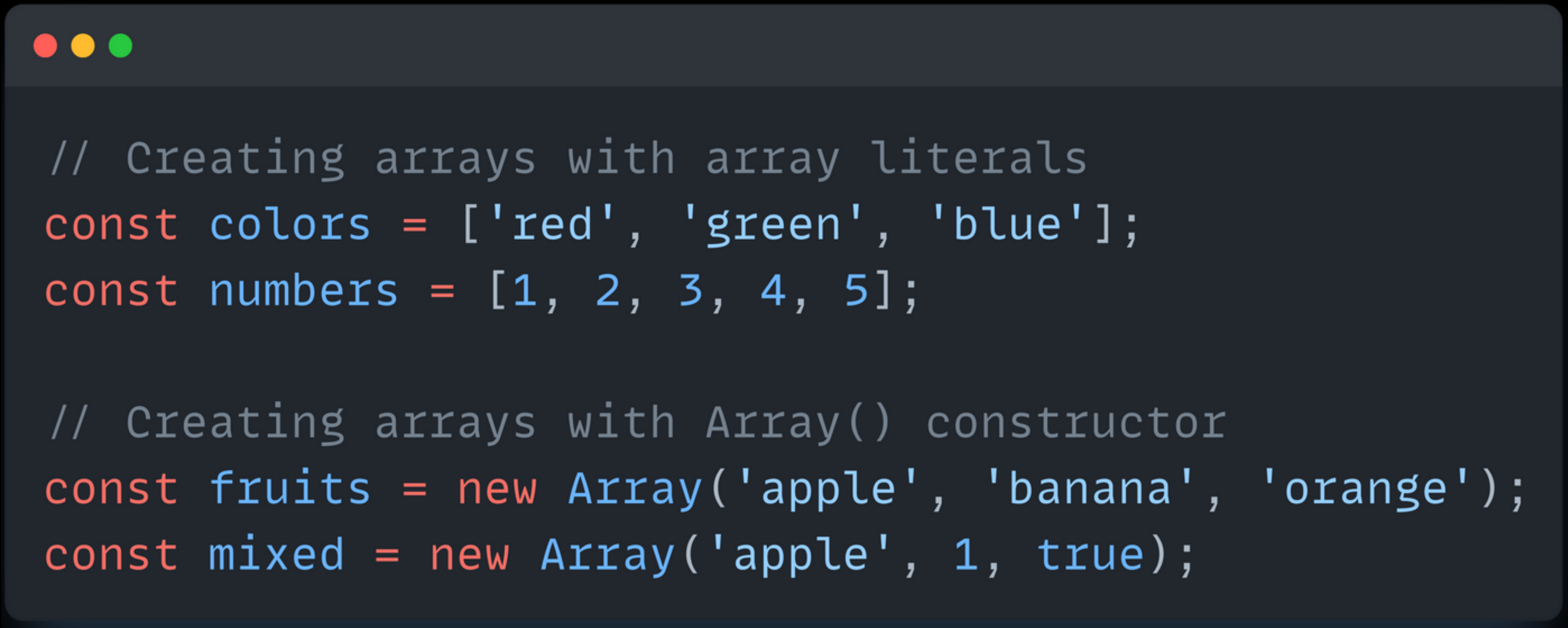
[ ]

TechGlobal School

Swipe

JavaScript Arrays | Introduction

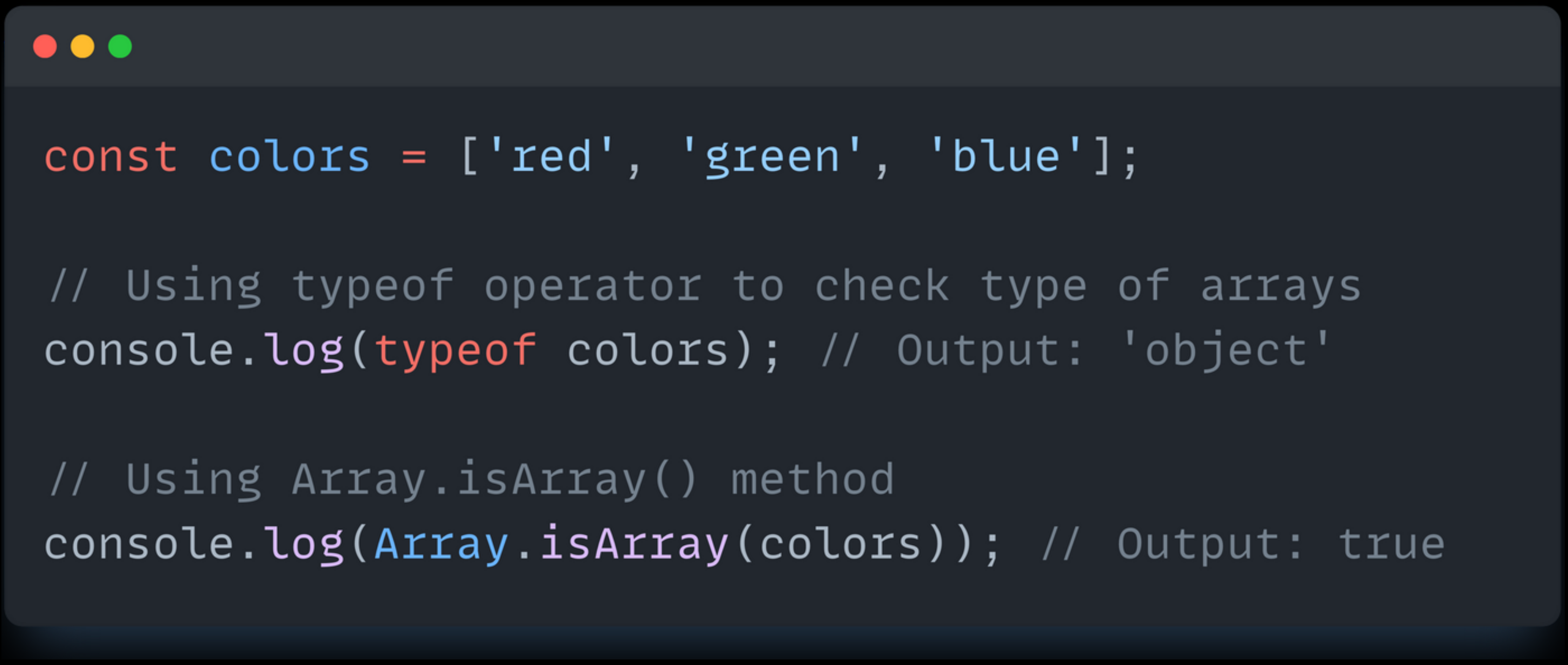
- JavaScript arrays are special types of objects used to store multiple values in a single variable.
- They are one of the most commonly used data structures in JavaScript and provide a convenient way to organize and manipulate collections of data.
- We can create arrays using array literals [ ] or the Array() constructor.



```javascript
// Creating arrays with array literals
const colors = ['red', 'green', 'blue'];
const numbers = [1, 2, 3, 4, 5];

// Creating arrays with Array() constructor
const fruits = new Array('apple', 'banana', 'orange');
const mixed = new Array('apple', 1, true);
```

JavaScript Arrays | typeof vs isArray()

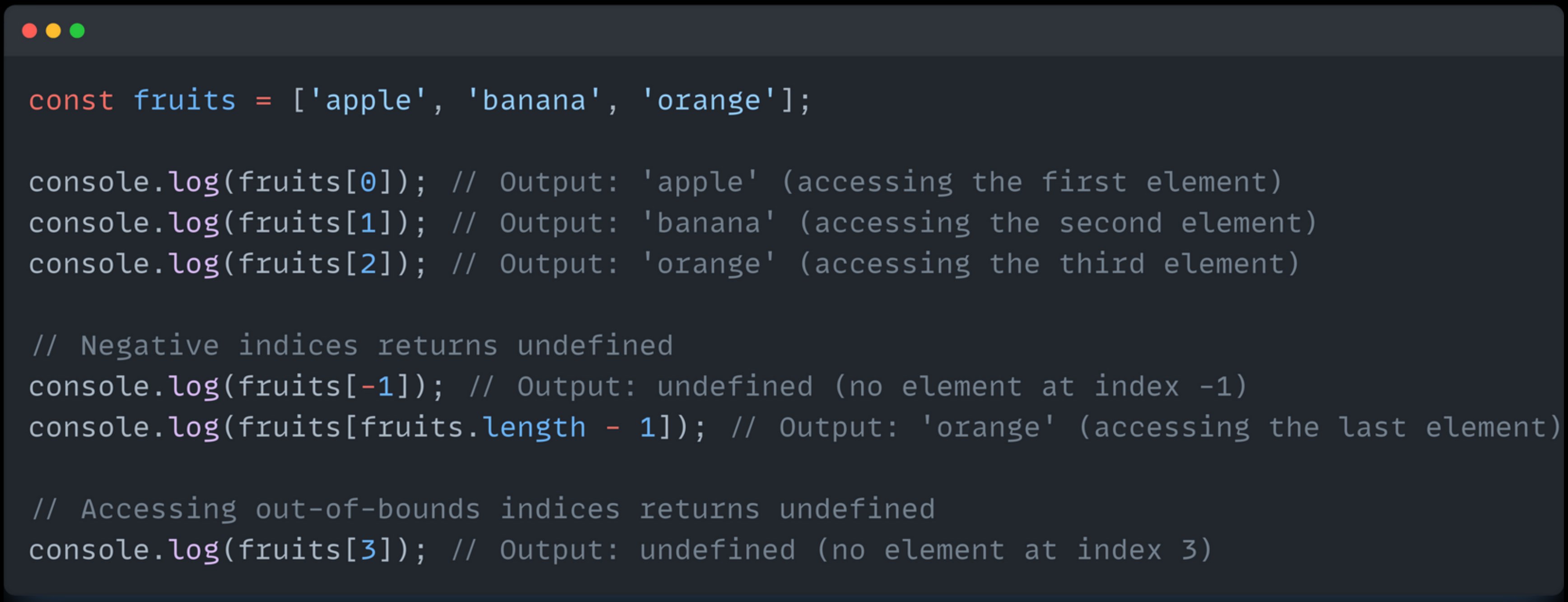
- We can use the typeof operator to check if a variable is an array. However, typeof returns 'object' for arrays, as arrays are objects in JavaScript.



```javascript
const colors = ['red', 'green', 'blue'];

// Using typeof operator to check type of arrays
console.log(typeof colors); // Output: 'object'

// Using Array.isArray() method
console.log(Array.isArray(colors)); // Output: true
```

JavaScript Arrays | accessing elements

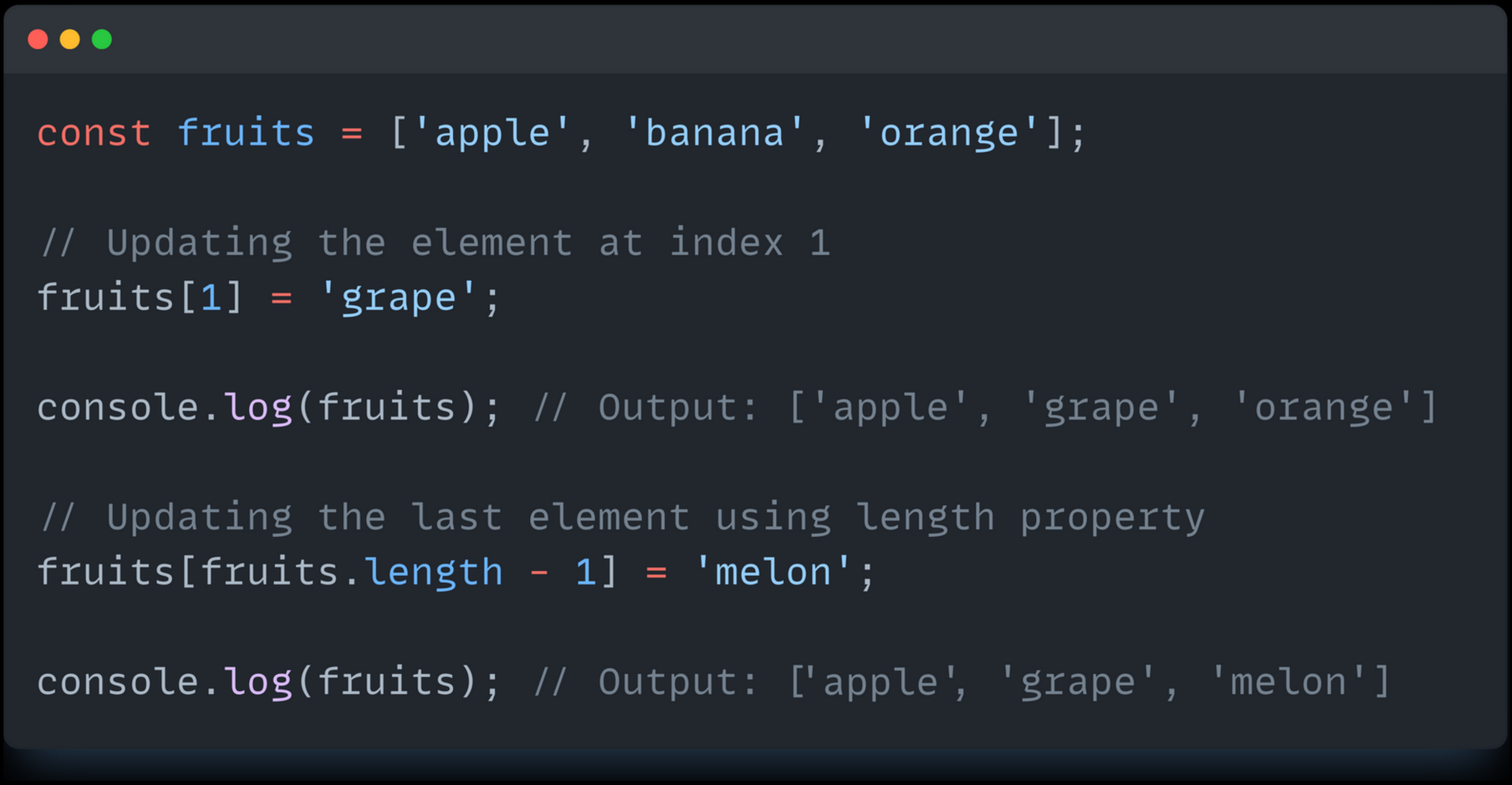
- In JavaScript, we can access array elements using square brackets [] notation.
- The square brackets contain the index of the element you want to access. Array indexing starts at 0, so the first element of an array has an index of 0, the second element has an index of 1, and so on.



```javascript
const fruits = ['apple', 'banana', 'orange'];

console.log(fruits[0]); // Output: 'apple' (accessing the first element)
console.log(fruits[1]); // Output: 'banana' (accessing the second element)
console.log(fruits[2]); // Output: 'orange' (accessing the third element)

// Negative indices returns undefined
console.log(fruits[-1]); // Output: undefined (no element at index -1)
console.log(fruits[fruits.length - 1]); // Output: 'orange' (accessing the last element)

// Accessing out-of-bounds indices returns undefined
console.log(fruits[3]); // Output: undefined (no element at index 3)
```

JavaScript Arrays | updating elements

▼ In JavaScript, we can update array elements using square brackets [] notation to access the element at a specific index, and then assigning a new value to that element.

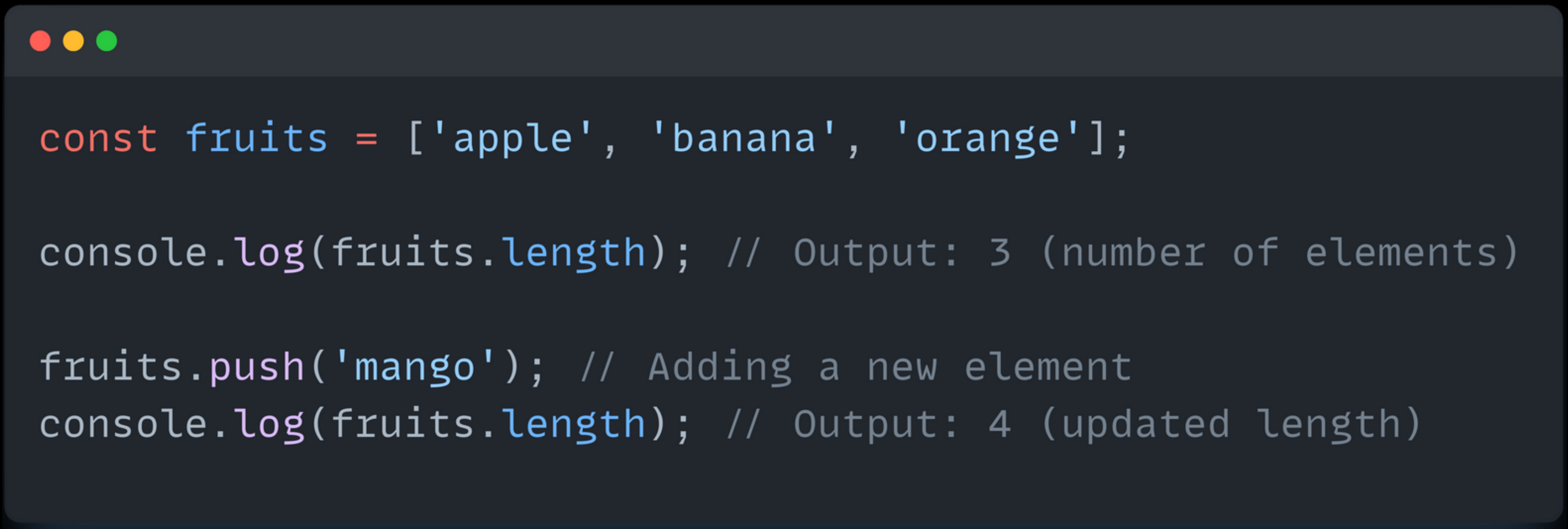


```javascript
const fruits = ['apple', 'banana', 'orange'];

// Updating the element at index 1
fruits[1] = 'grape';

console.log(fruits); // Output: ['apple', 'grape', 'orange']

// Updating the last element using length property
fruits[fruits.length - 1] = 'melon';

console.log(fruits); // Output: ['apple', 'grape', 'melon']
```

TechGlobal School

JavaScript Arrays | length property

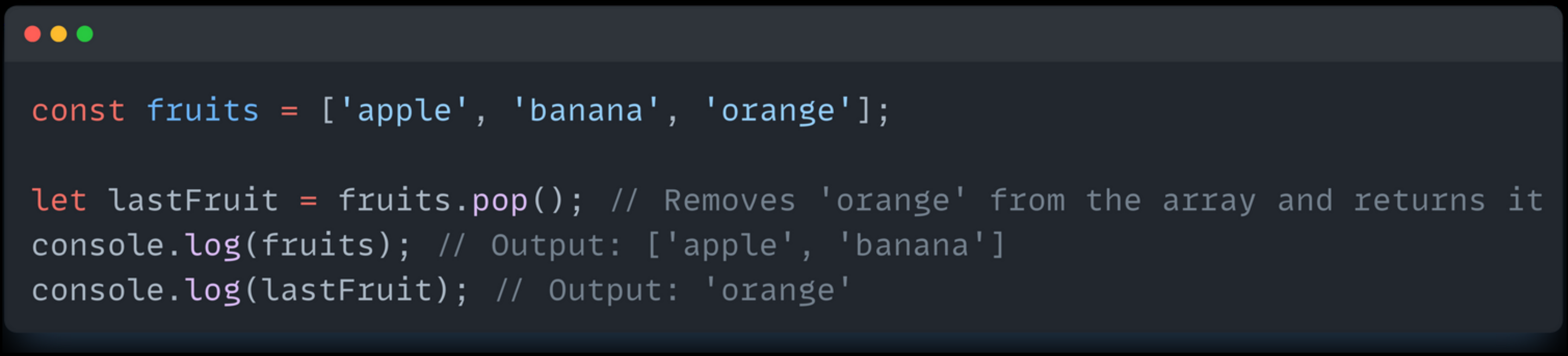
- In JavaScript, the length property of an array returns the number of elements in the array.
- It is a property that automatically updates whenever elements are added to or removed from the array.



```javascript
const fruits = ['apple', 'banana', 'orange'];

console.log(fruits.length); // Output: 3 (number of elements)

fruits.push('mango'); // Adding a new element
console.log(fruits.length); // Output: 4 (updated length)
```

JavaScript Arrays | pop() method

- The pop() method removes the last element from an array and returns that element.
- It mutates the original array by removing the last element.
- If the array is empty (length is 0), pop() returns undefined.
- Syntax: array.pop()



```javascript
const fruits = ['apple', 'banana', 'orange'];

let lastFruit = fruits.pop(); // Removes 'orange' from the array and returns it
console.log(fruits); // Output: ['apple', 'banana']
console.log(lastFruit); // Output: 'orange'
```

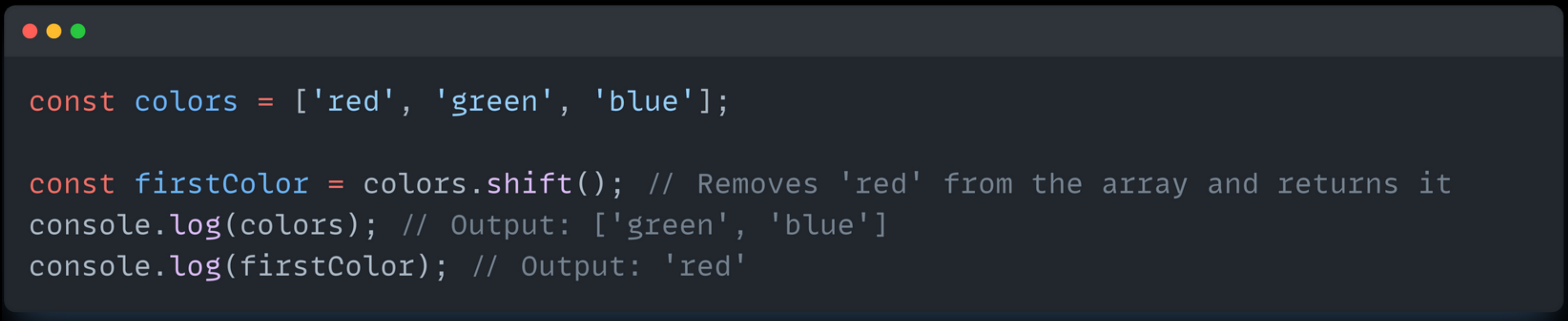
JavaScript Arrays | push() method

- The push() method adds one or more elements to the end of an array and returns the new length of the array.
- It mutates the original array by adding elements to the end.
- The elements are appended in the order they appear in the arguments list.
- Syntax: array.push(element1, ..., elementN)

```
const fruits = ['apple', 'banana'];

let newLength = fruits.push('orange', 'kiwi'); // Adds 'orange' and 'kiwi' to the end
console.log(fruits); // Output: ['apple', 'banana', 'orange', 'kiwi']
console.log(newLength); // Output: 4 (new length of the array)
```

JavaScript Arrays | shift() method

- The shift() method removes the first element from an array and returns that element.
- It mutates the original array by removing the first element and shifting all subsequent elements one position to the left.
- If the array is empty (length is 0), shift() returns undefined.
- Syntax: array.shift()



```javascript
const colors = ['red', 'green', 'blue'];

const firstColor = colors.shift(); // Removes 'red' from the array and returns it
console.log(colors); // Output: ['green', 'blue']
console.log(firstColor); // Output: 'red'
```

TechGlobal School

JavaScript Arrays | unshift() method

- The unshift() method adds one or more elements to the beginning of an array and returns the new length of the array.
- It mutates the original array by adding elements to the beginning and shifting all existing elements to the right.
- The elements are prepended in the order they appear in the arguments list.
- Syntax: array.unshift(element1, ..., elementN)

```
const numbers = [3, 4, 5];

const newLength = numbers.unshift(1, 2); // Adds 1 and 2 to the beginning of the array
console.log(numbers); // Output: [1, 2, 3, 4, 5]
console.log(newLength); // Output: 5 (new length of the array)
```

TechGlobal School

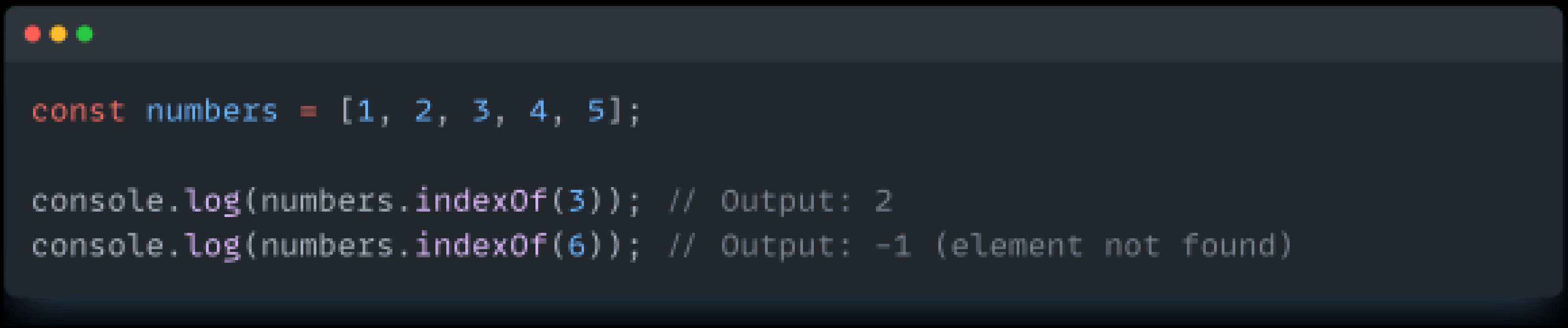
JavaScript Arrays | splice() method

- The splice() method changes the contents of an array by removing or replacing existing elements and/or adding new elements in place.
- It modifies the original array and returns an array containing the removed elements, if any.
- Syntax: array.splice(startIndex, deleteCount, item1, item2, ...)
  - startIndex: The index at which to start modifying the array. deleteCount: The number of elements to remove starting from the startIndex.
  - item1, item2, ...: Elements to add to the array, starting at the startIndex.

```
const numbers = [1, 2, 3, 4, 5];

// Remove elements starting from index 2 (3 and 4)
const removedElements = numbers.splice(2, 2);
console.log(numbers); // Output: [1, 2, 5]
console.log(removedElements); // Output: [3, 4]

// Add elements at index 2 (6 and 7) without removing any elements
numbers.splice(2, 0, 6, 7);
console.log(numbers); // Output: [1, 2, 6, 7, 5]

// Replace element at index 3 (7 with 8)
numbers.splice(3, 1, 8);
console.log(numbers); // Output: [1, 2, 6, 8, 5]
```

JavaScript Arrays | indexOf() method

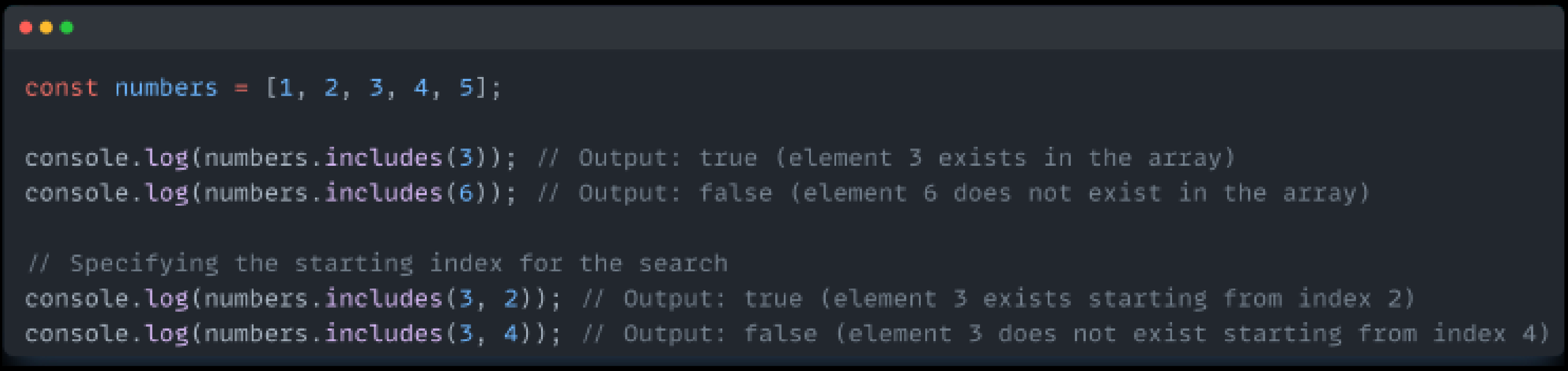
- The indexOf() method searches the array for the specified element from the beginning (index 0) to the end.
- It returns the index of the first occurrence of the specified element in the array.
- If the element is not found, indexOf() returns -1.
- Syntax: array.indexOf(searchElement, startIndex)
  - searchElement: The element to search for in the array.
  - startIndex (optional): The index at which to start the search. If omitted, the search starts from index 0.



```
const numbers = [1, 2, 3, 4, 5];

console.log(numbers.indexOf(3)); // Output: 2
console.log(numbers.indexOf(6)); // Output: -1 (element not found)
```

JavaScript Arrays | lastIndexOf() method

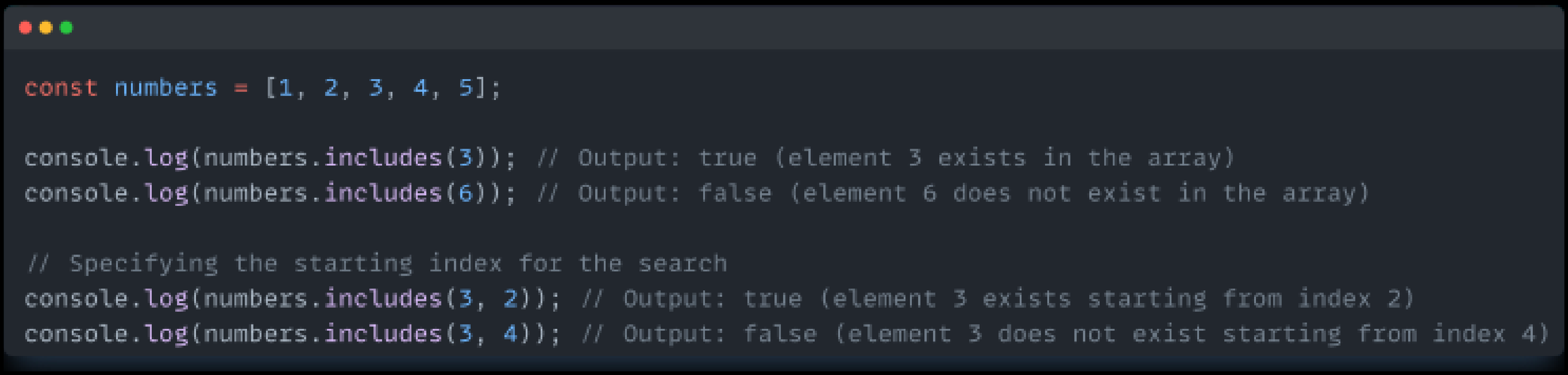
- The lastIndexOf() method searches the array for the specified element from the end to the beginning.
- It returns the index of the first occurrence of the specified element in the array.
- If the element is not found, lastIndexOf() returns -1.
- Syntax: array.lastIndexOf(searchElement, startIndex)
  - searchElement: The element to search for in the array.
  - startIndex (optional): The index at which to start the search. If omitted, the search starts from the last element of the array.



```
const numbers = [1, 2, 3, 4, 5];

console.log(numbers.includes(3)); // Output: true (element 3 exists in the array)
console.log(numbers.includes(6)); // Output: false (element 6 does not exist in the array)

// Specifying the starting index for the search
console.log(numbers.includes(3, 2)); // Output: true (element 3 exists starting from index 2)
console.log(numbers.includes(3, 4)); // Output: false (element 3 does not exist starting from index 4)
```

JavaScript Arrays | includes() method

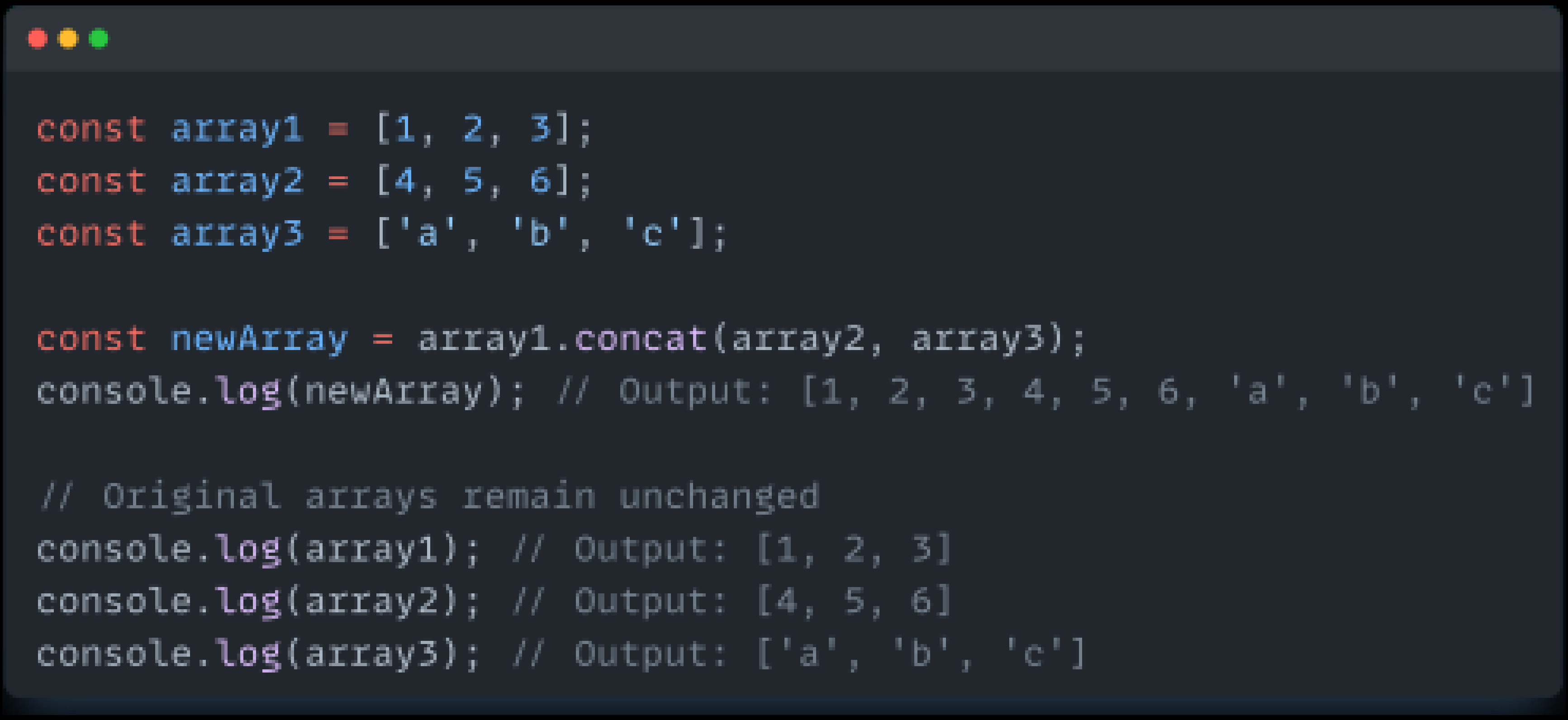
- The includes() method checks whether a specified element is present in the array.
- It returns true if the specified element is found in the array, and false otherwise.
- Syntax: array.includes(searchElement, fromIndex)
  - searchElement: The element to search for in the array.
  - fromIndex (optional): The index at which to start the search.



```javascript
const numbers = [1, 2, 3, 4, 5];

console.log(numbers.includes(3)); // Output: true (element 3 exists in the array)
console.log(numbers.includes(6)); // Output: false (element 6 does not exist in the array)

// Specifying the starting index for the search
console.log(numbers.includes(3, 2)); // Output: true (element 3 exists starting from index 2)
console.log(numbers.includes(3, 4)); // Output: false (element 3 does not exist starting from index 4)
```

JavaScript Arrays | concat() method

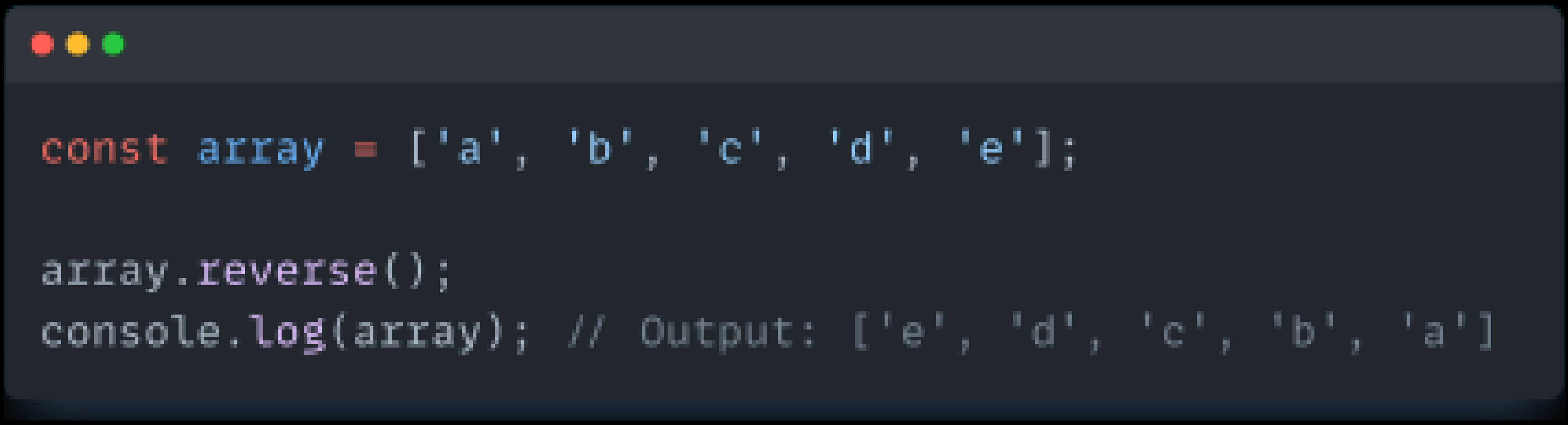
- The concat() method is used to merge two or more arrays, creating a new array that contains the elements of the original arrays.
- It does not modify the existing arrays but instead returns a new array with the combined elements.
- Syntax: array.concat(array1, array2, ..., arrayN)
    - array1, array2, ..., arrayN: Arrays or values to concatenate to the original array. These can be arrays, values, or a combination of both.



```
const array1 = [1, 2, 3];
const array2 = [4, 5, 6];
const array3 = ['a', 'b', 'c'];

const newArray = array1.concat(array2, array3);
console.log(newArray); // Output: [1, 2, 3, 4, 5, 6, 'a', 'b', 'c']

// Original arrays remain unchanged
console.log(array1); // Output: [1, 2, 3]
console.log(array2); // Output: [4, 5, 6]
console.log(array3); // Output: ['a', 'b', 'c']
```

JavaScript Arrays | reverse() method

- The reverse() method is used to reverse the order of elements in an array.
- It modifies the original array in place and returns the reversed array.
- It returns the reversed array, with the elements rearranged in the opposite order.
- Syntax: array.reverse()

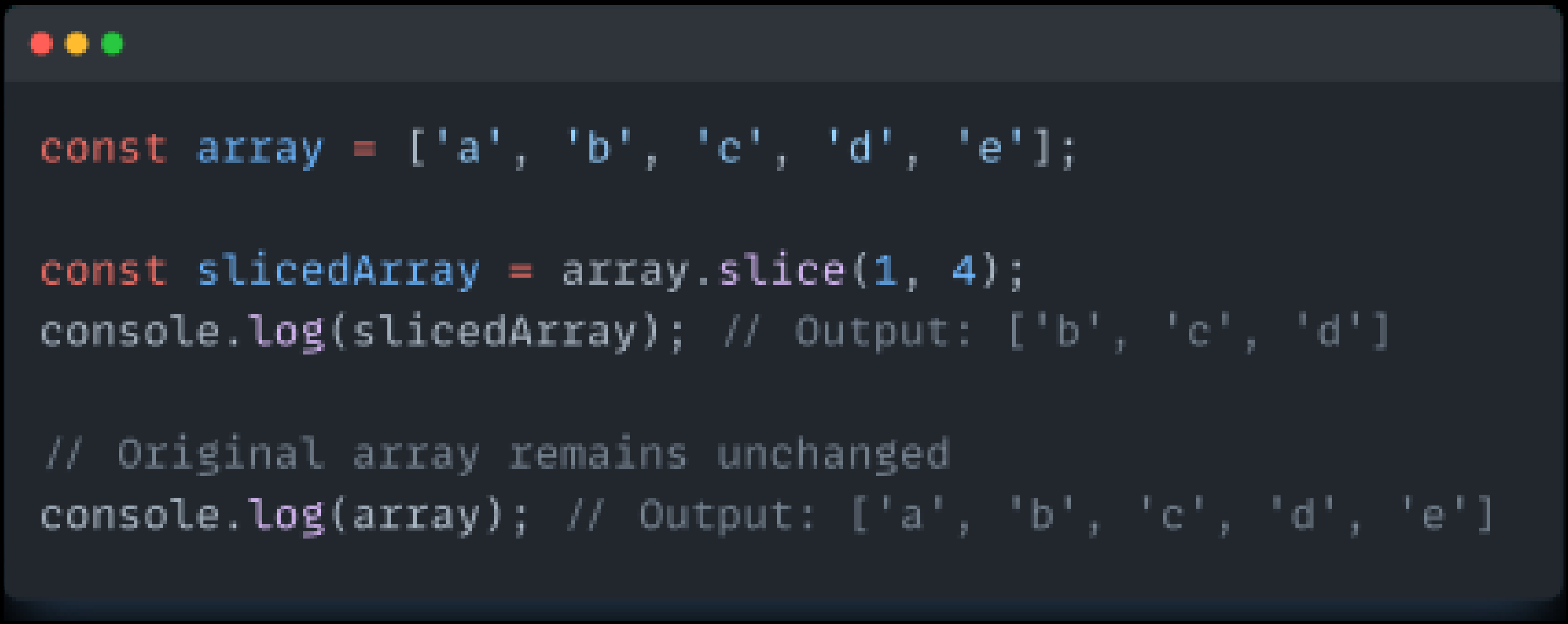


```
const array = ['a', 'b', 'c', 'd', 'e'];

array.reverse();
console.log(array); // Output: ['e', 'd', 'c', 'b', 'a']
```

TechGlobal School

JavaScript Arrays | slice() method

- The slice() method is used to extract a portion of an array into a new array, without modifying the original array.
- It takes two optional parameters: start and end, which specify the beginning and end of the slice, respectively.
- It returns a new array containing the extracted elements.
- The original array remains unchanged.
- Syntax: array.slice(start, end)
  - start (optional): The index at which to begin the extraction. If omitted, slice() starts from index 0. If negative, it counts backward from the end of the array.
  - end (optional): The index before which to end the extraction. slice() extracts up to but does not include the end index. If omitted or greater than the length of the array, slice() extracts to the end of the array. If negative, it counts backward from the end of the array.

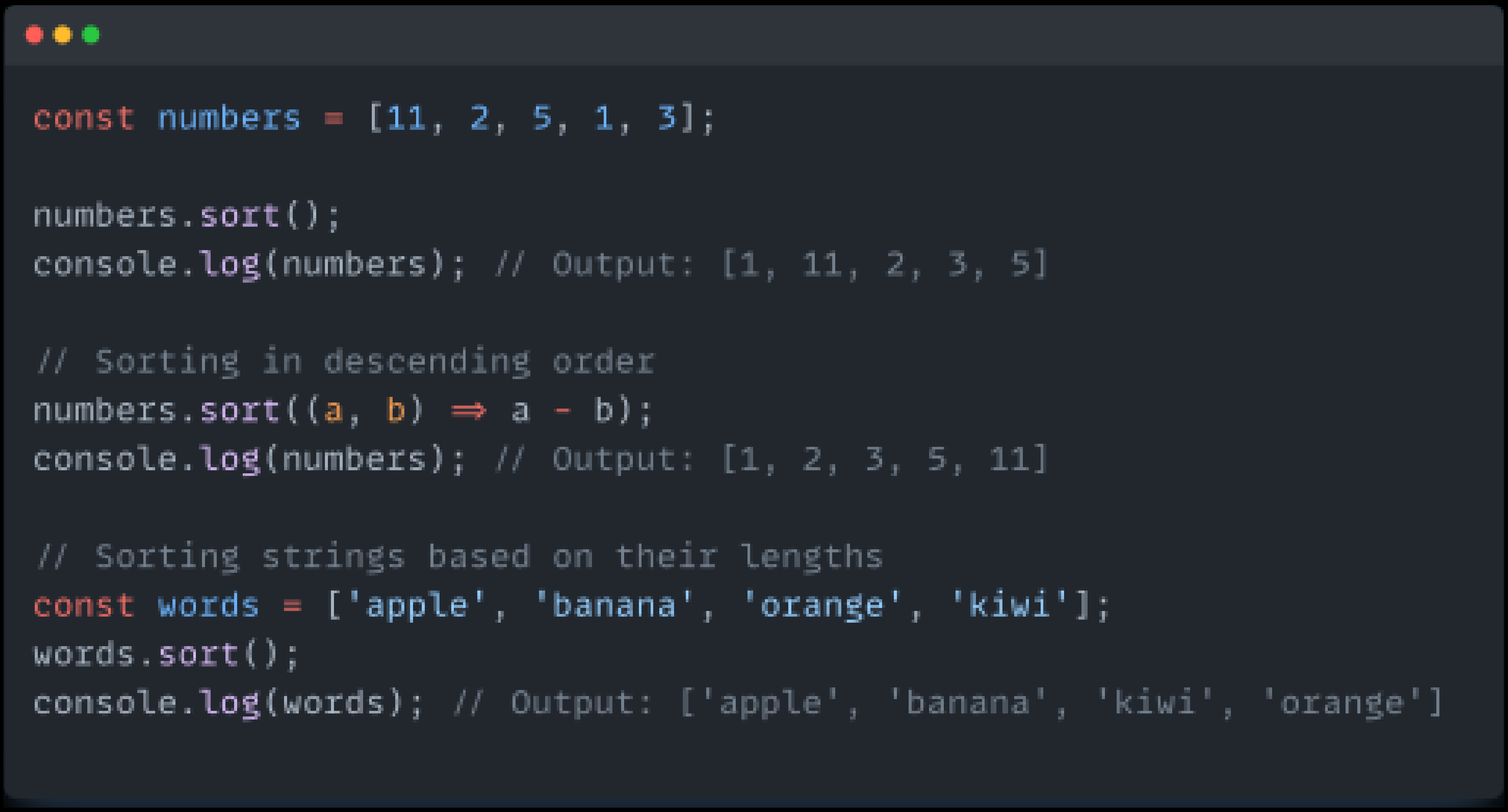


```
const array = ['a', 'b', 'c', 'd', 'e'];

const slicedArray = array.slice(1, 4);
console.log(slicedArray); // Output: ['b', 'c', 'd']

// Original array remains unchanged
console.log(array); // Output: ['a', 'b', 'c', 'd', 'e']
```

JavaScript Arrays | sort() method

- The sort() method sorts the elements of an array in place and returns the sorted array.
- It modifies the original array and does not create a new array.

- Syntax: array.sort(compareFunction), where compareFunction is optional.
  - compareFunction (optional): A function that defines the sort order. If omitted, the array elements are sorted based on their string representations. If provided, the function should return a negative value if the first argument should come before the second, a positive value if the second argument should come before the first, or zero if the two elements are equal.

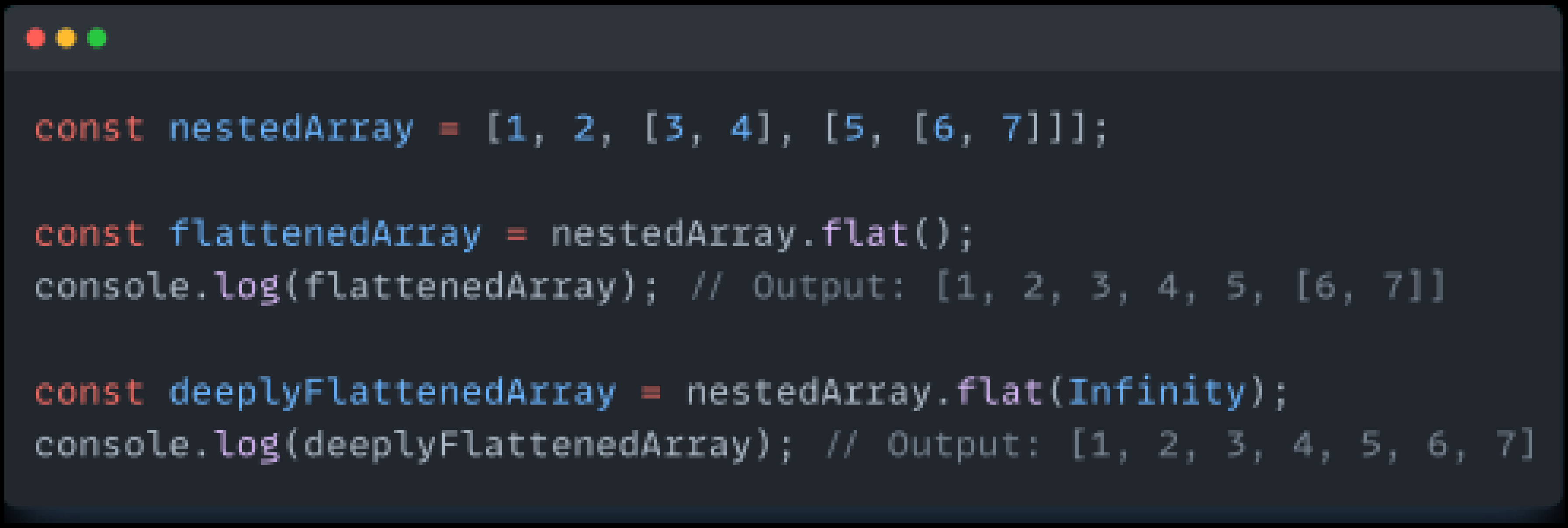


```javascript
const numbers = [11, 2, 5, 1, 3];

numbers.sort();
console.log(numbers); // Output: [1, 11, 2, 3, 5]

// Sorting in descending order
numbers.sort((a, b) => a - b);
console.log(numbers); // Output: [1, 2, 3, 5, 11]

// Sorting strings based on their lengths
const words = ['apple', 'banana', 'orange', 'kiwi'];
words.sort();
console.log(words); // Output: ['apple', 'banana', 'kiwi', 'orange']
```

TechGlobal School

JavaScript Arrays | flat() method

- The flat() method creates a new array with all sub-array elements concatenated into it recursively up to the specified depth. It flattens the nested array structure.
- It returns a new array that is flattened to the specified depth.
- Syntax: array.flat(depth), where depth is optional.
  - depth (optional): The depth level specifying how deep nested arrays should be flattened. The default value is 1. If depth is Infinity, all nested arrays will be flattened.

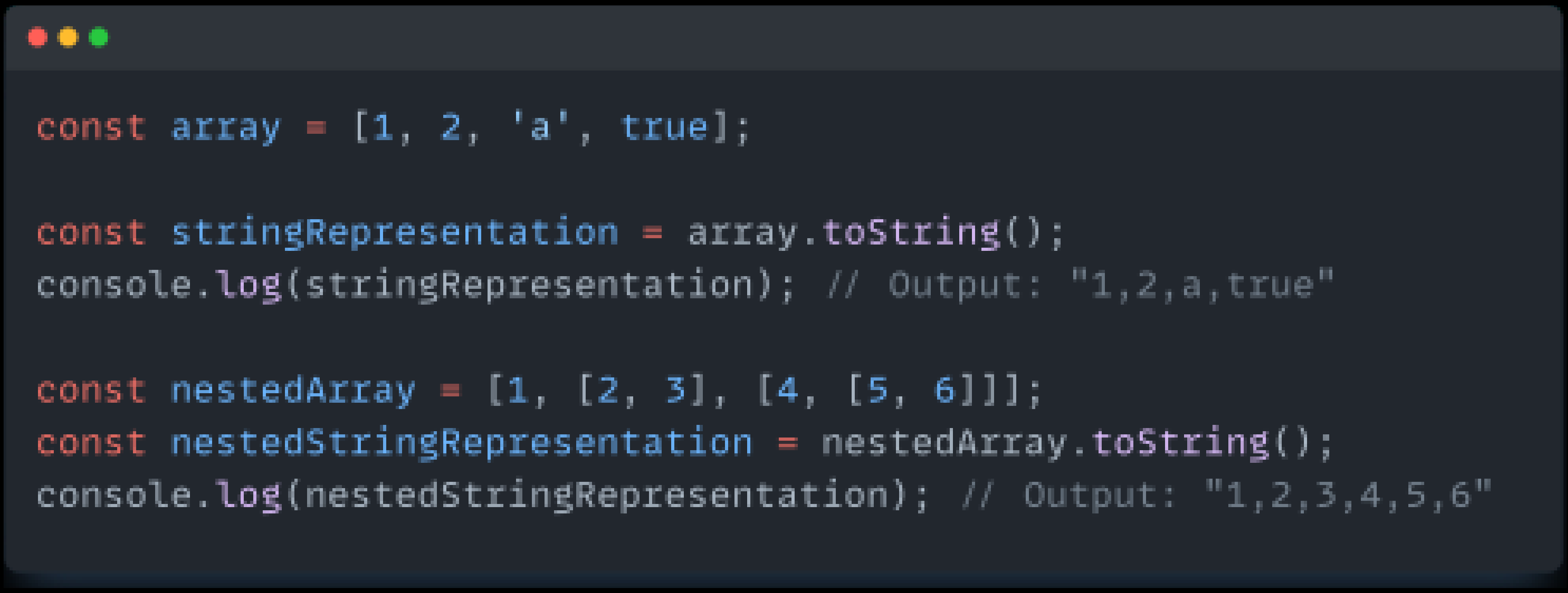


```
const nestedArray = [1, 2, [3, 4], [5, [6, 7]]];

const flattenedArray = nestedArray.flat();
console.log(flattenedArray); // Output: [1, 2, 3, 4, 5, [6, 7]]

const deeplyFlattenedArray = nestedArray.flat(Infinity);
console.log(deeplyFlattenedArray); // Output: [1, 2, 3, 4, 5, 6, 7]
```

TechGlobal School

JavaScript Arrays | toString() method

- The toString() method converts an array to a string representation, where each element is converted to a string and separated by commas. If an element is undefined, it is converted to the string "undefined". If an element is null, it is converted to the string "null".
- It returns a string representation of the array.
- Syntax: array.toString()

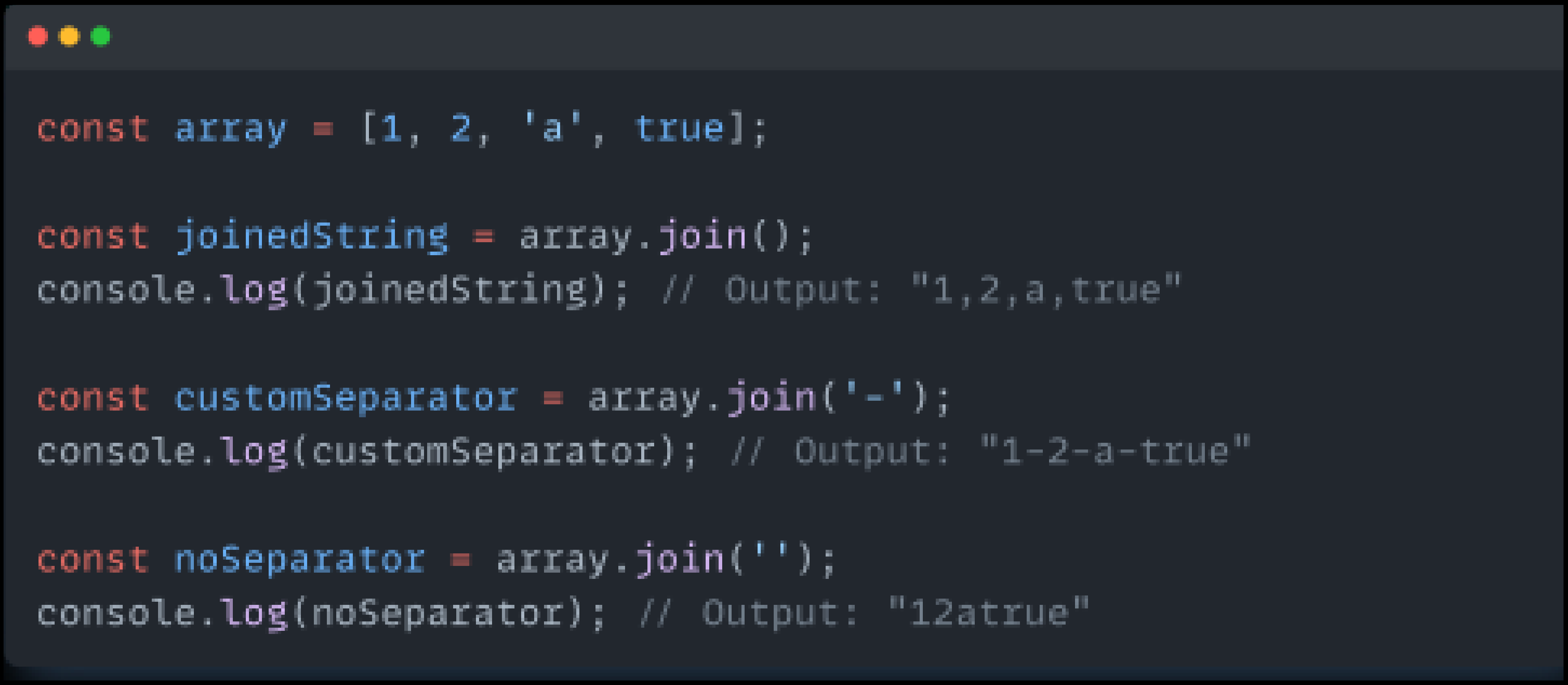


```
const array = [1, 2, 'a', true];

const stringRepresentation = array.toString();
console.log(stringRepresentation); // Output: "1,2,a,true"

const nestedArray = [1, [2, 3], [4, [5, 6]]];
const nestedStringRepresentation = nestedArray.toString();
console.log(nestedStringRepresentation); // Output: "1,2,3,4,5,6"
```

TechGlobal School

JavaScript Arrays | join() method

- The join() method joins all elements of an array into a single string. Each element is converted to a string and concatenated together, separated by the specified separator string.
- It returns a string representing the joined array elements.
- Syntax: array.join(separator)
  - separator (optional): The string used to separate the elements of the array in the resulting string. If omitted, the elements are separated by commas. If separator is an empty string (""), the elements are joined without any characters between them.

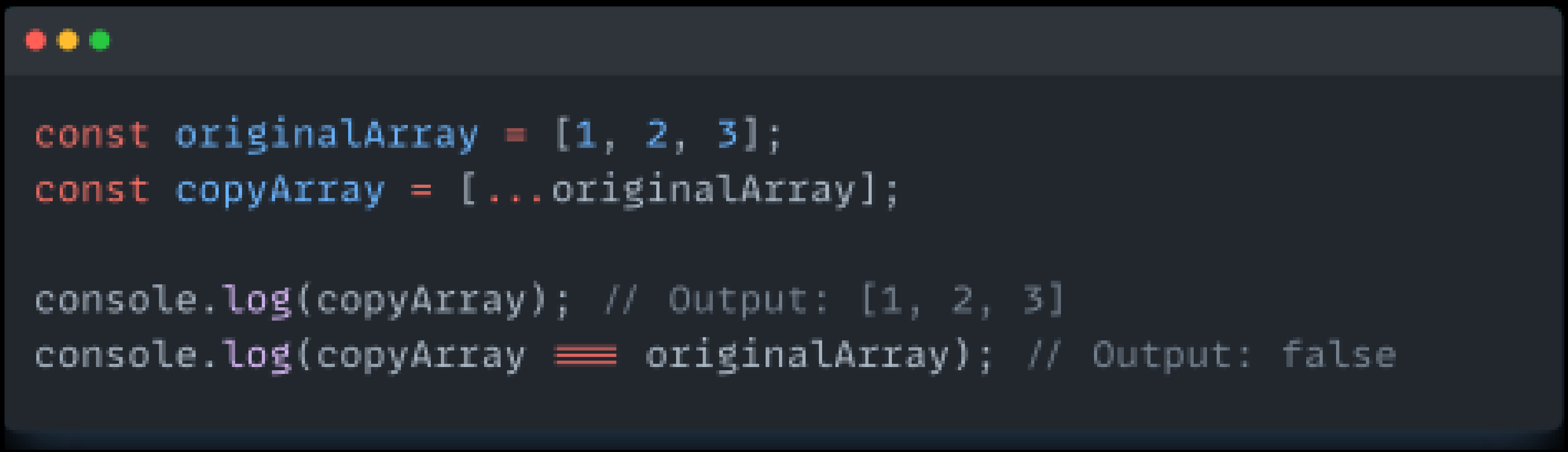


```
const array = [1, 2, 'a', true];

const joinedString = array.join();
console.log(joinedString); // Output: "1,2,a,true"

const customSeparator = array.join('-');
console.log(customSeparator); // Output: "1-2-a-true"

const noSeparator = array.join('');
console.log(noSeparator); // Output: "12atrue"
```

TechGlobal School

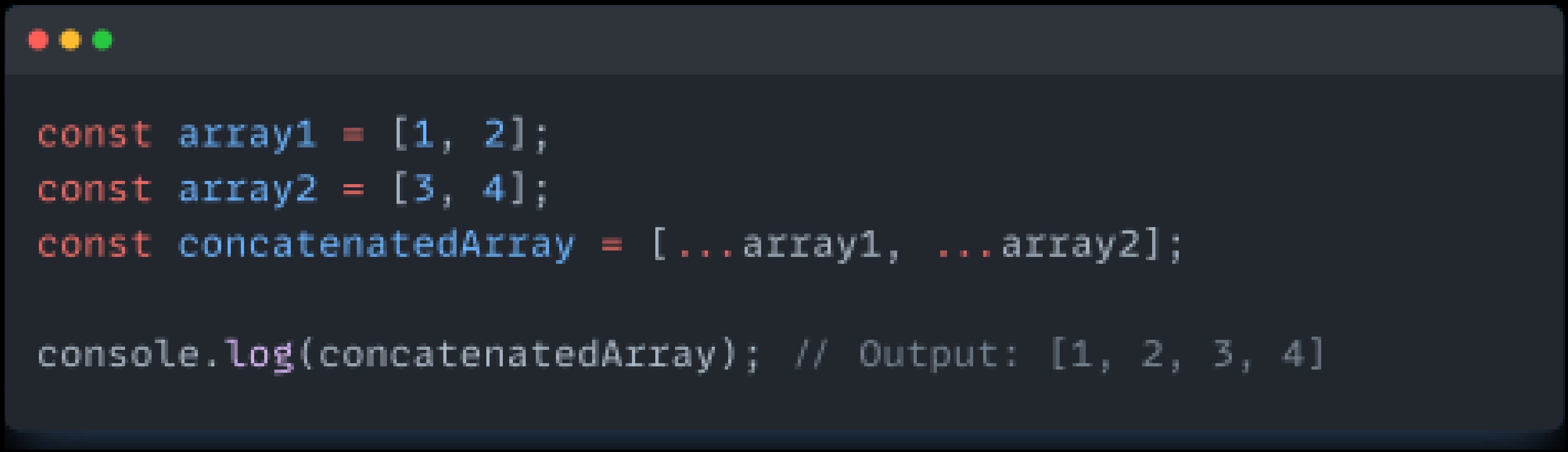
JavaScript Arrays | Spread Operator

- The spread operator (...) is a feature introduced in ES6 that allows an iterable, such as an array, to be expanded into individual elements.
- When used with arrays, the spread operator can be used for various operations, including creating shallow copies of arrays, concatenating arrays, and passing array elements as arguments to functions.
- The spread operator can be used to create shallow copies of arrays. This means that a new array is created, and the elements of the original array are copied into the new array.



```javascript
const originalArray = [1, 2, 3];
const copyArray = [...originalArray];

console.log(copyArray); // Output: [1, 2, 3]
console.log(copyArray === originalArray); // Output: false
```

- The spread operator can be used to concatenate multiple arrays into a single array.



```javascript
const array1 = [1, 2];
const array2 = [3, 4];
const concatenatedArray = [...array1, ...array2];

console.log(concatenatedArray); // Output: [1, 2, 3, 4]
```

JavaScript Arrays | Spread Operator

▼ The spread operator can be used to pass individual array elements as arguments to a function.

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

function sum(a, b, c) {
    return a + b + c;
}

const result = sum(...numbers);
console.log(result); // Output: 6
```

▼ The spread operator can be used to add new elements to an array, either at the beginning or end.

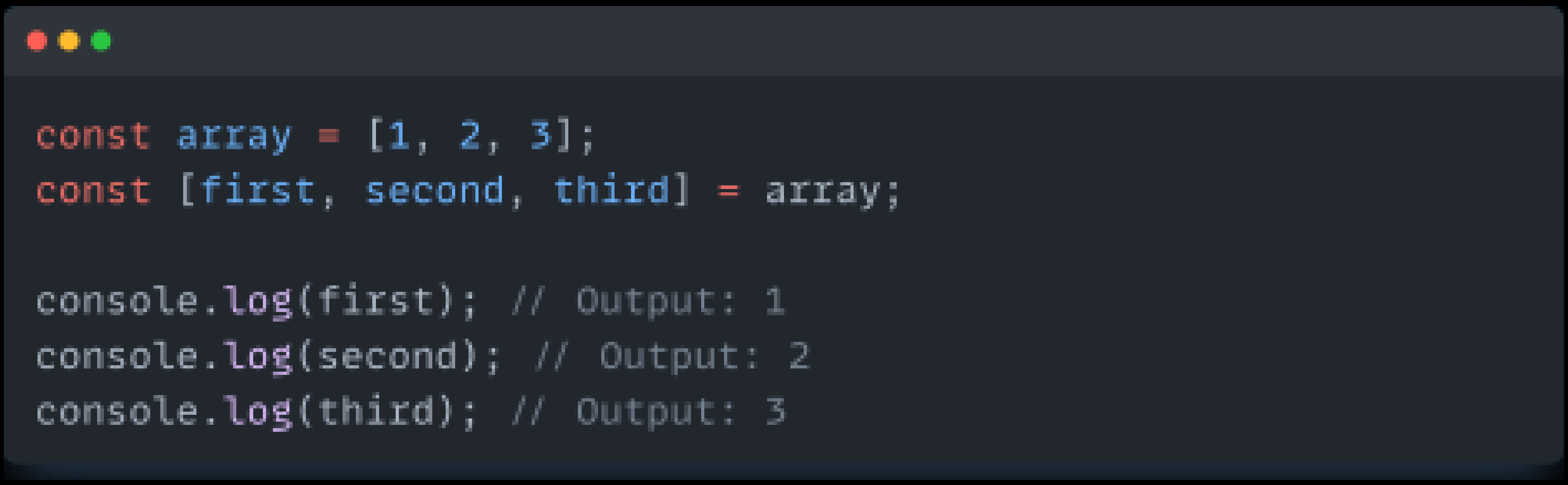
```javascript
const array = [1, 2, 3];
const newArray1 = [0, ...array]; // Add element at the beginning
const newArray2 = [...array, 4]; // Add element at the end

console.log(newArray1); // Output: [0, 1, 2, 3]
console.log(newArray2); // Output: [1, 2, 3, 4]
```

▼ BONUS: Use spread operator find min or max number in an array.

```javascript
const numbers = [3, 7, 2, 8, 5];

const maxNumber = Math.max(...numbers);
console.log("Maximum number:", maxNumber); // Output: Maximum number: 8

const minNumber = Math.min(...numbers);
console.log("Minimum number:", minNumber); // Output: Minimum number: 2
```

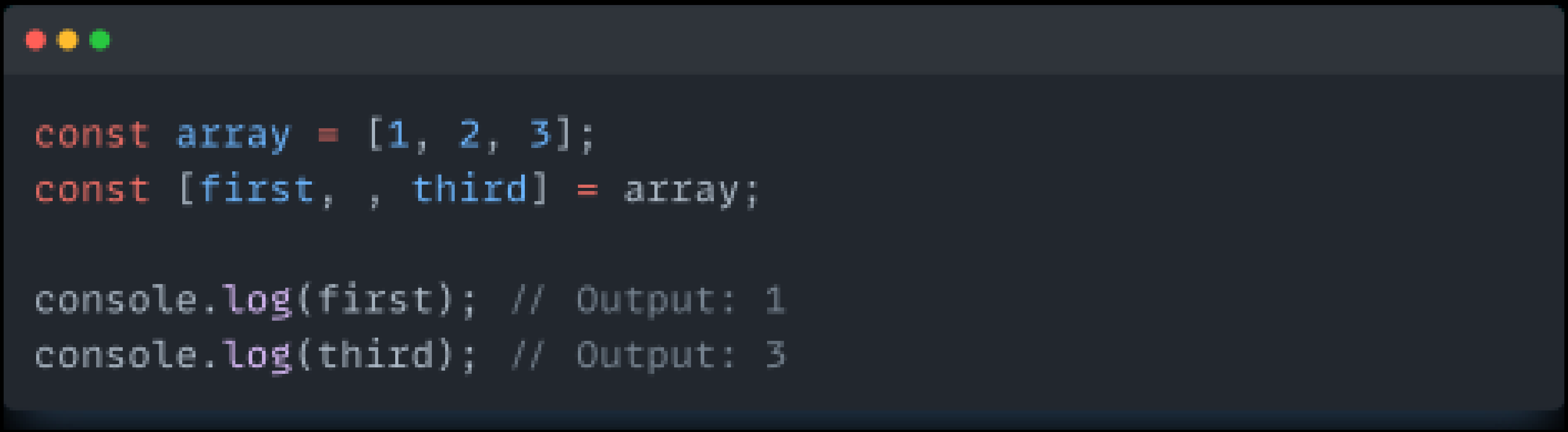
JavaScript Arrays | Destructuring

- Array destructuring is a feature introduced in ES6 that allows you to extract values from arrays and assign them to variables in a concise and readable way.
- It provides a convenient syntax for unpacking array elements into separate variables.
- Array destructuring uses square brackets ([]) on the left-hand side of an assignment to indicate that the values should be extracted from the array.



```javascript
const array = [1, 2, 3];
const [first, second, third] = array;

console.log(first); // Output: 1
console.log(second); // Output: 2
console.log(third); // Output: 3
```

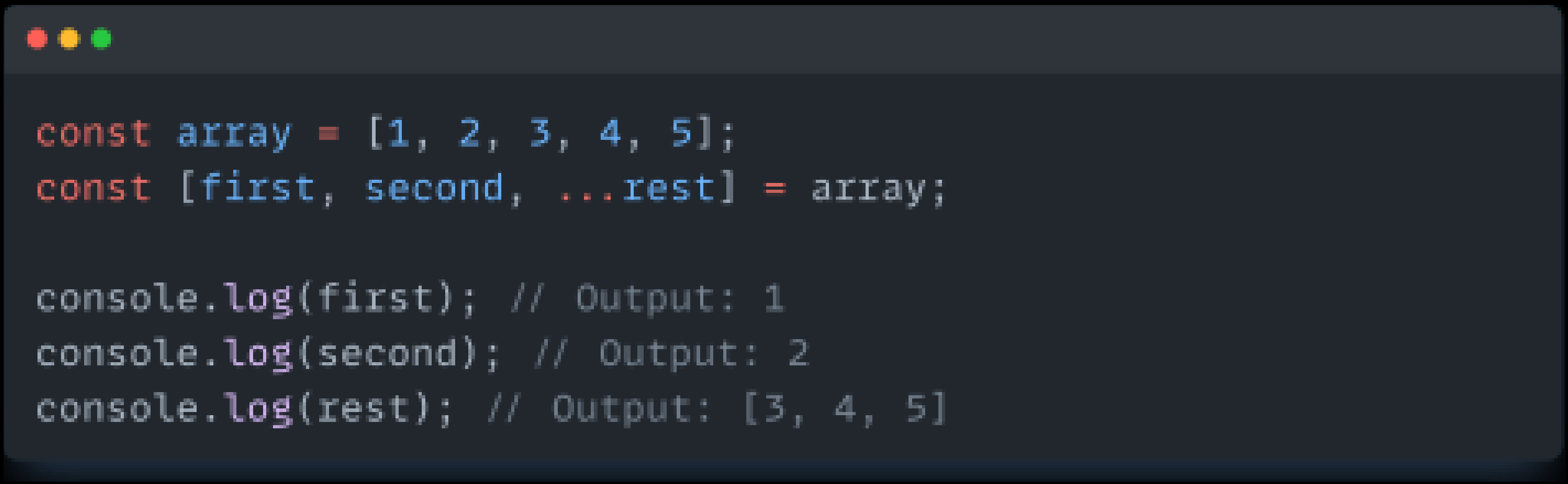
- You can skip elements in the array by omitting the corresponding variable name in the destructuring assignment.



```javascript
const array = [1, 2, 3];
const [first, , third] = array;

console.log(first); // Output: 1
console.log(third); // Output: 3
```

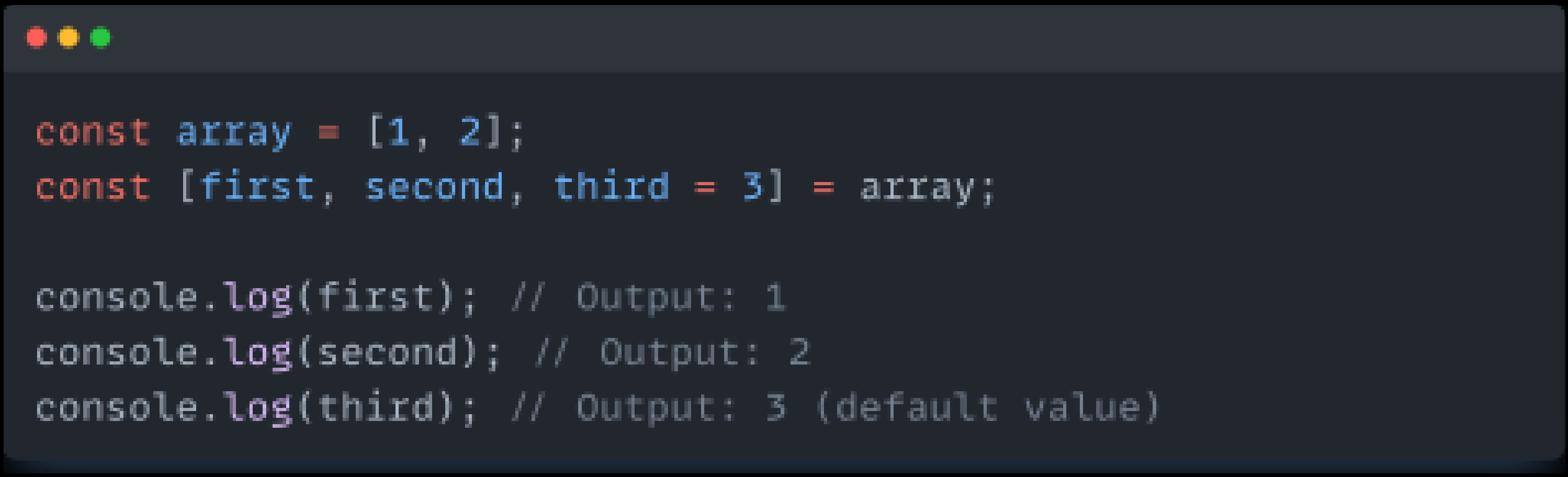
JavaScript Arrays | Destructuring

- The rest syntax (...) can be used to capture remaining elements of an array into a single variable.



```
const array = [1, 2, 3, 4, 5];
const [first, second, ...rest] = array;

console.log(first); // Output: 1
console.log(second); // Output: 2
console.log(rest); // Output: [3, 4, 5]
```

- You can provide default values for variables in case the corresponding array element is undefined.

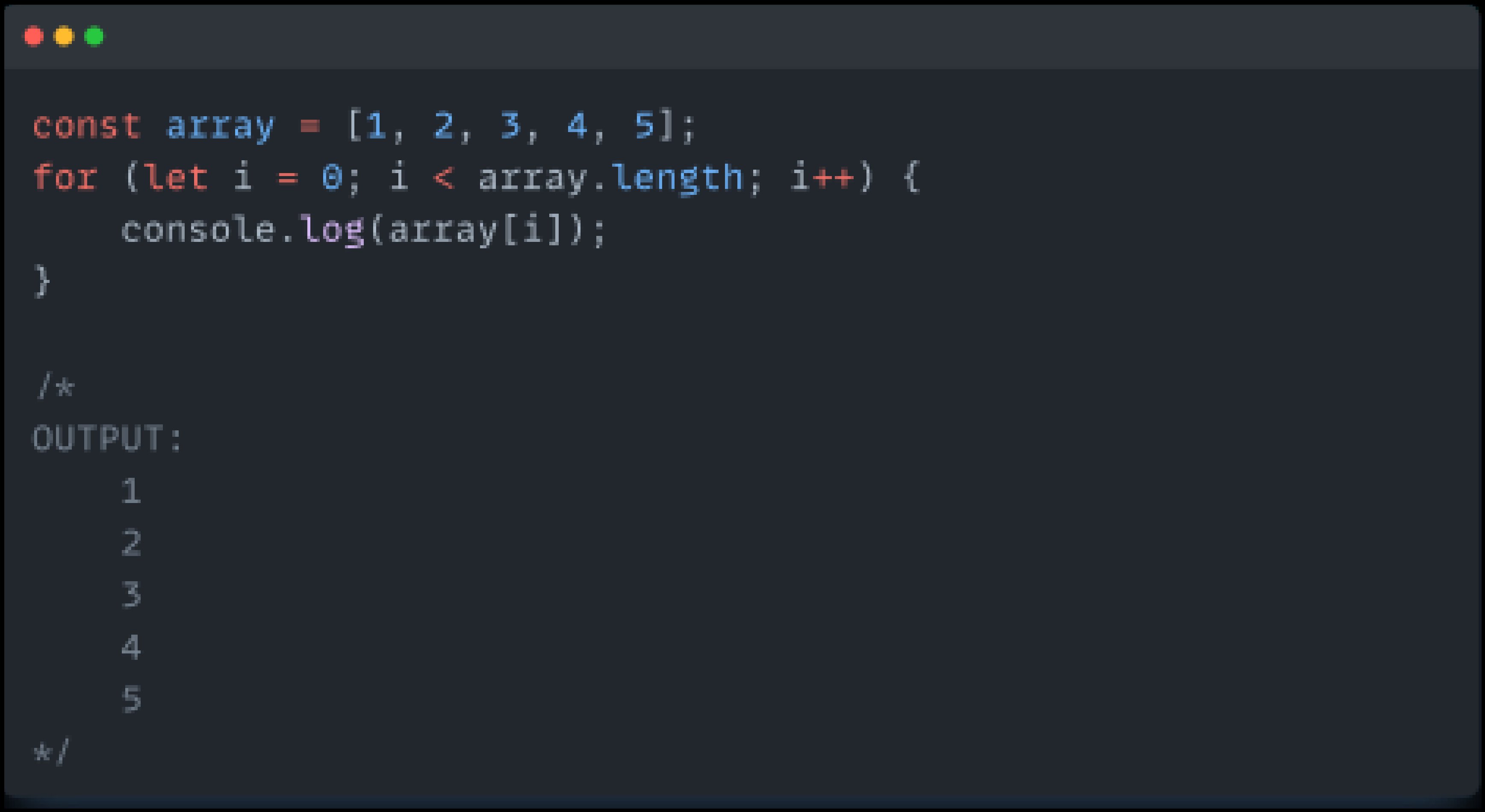


```
const array = [1, 2];
const [first, second, third = 3] = array;

console.log(first); // Output: 1
console.log(second); // Output: 2
console.log(third); // Output: 3 (default value)
```

TechGlobal School

JavaScript Arrays | Iteration with for loop

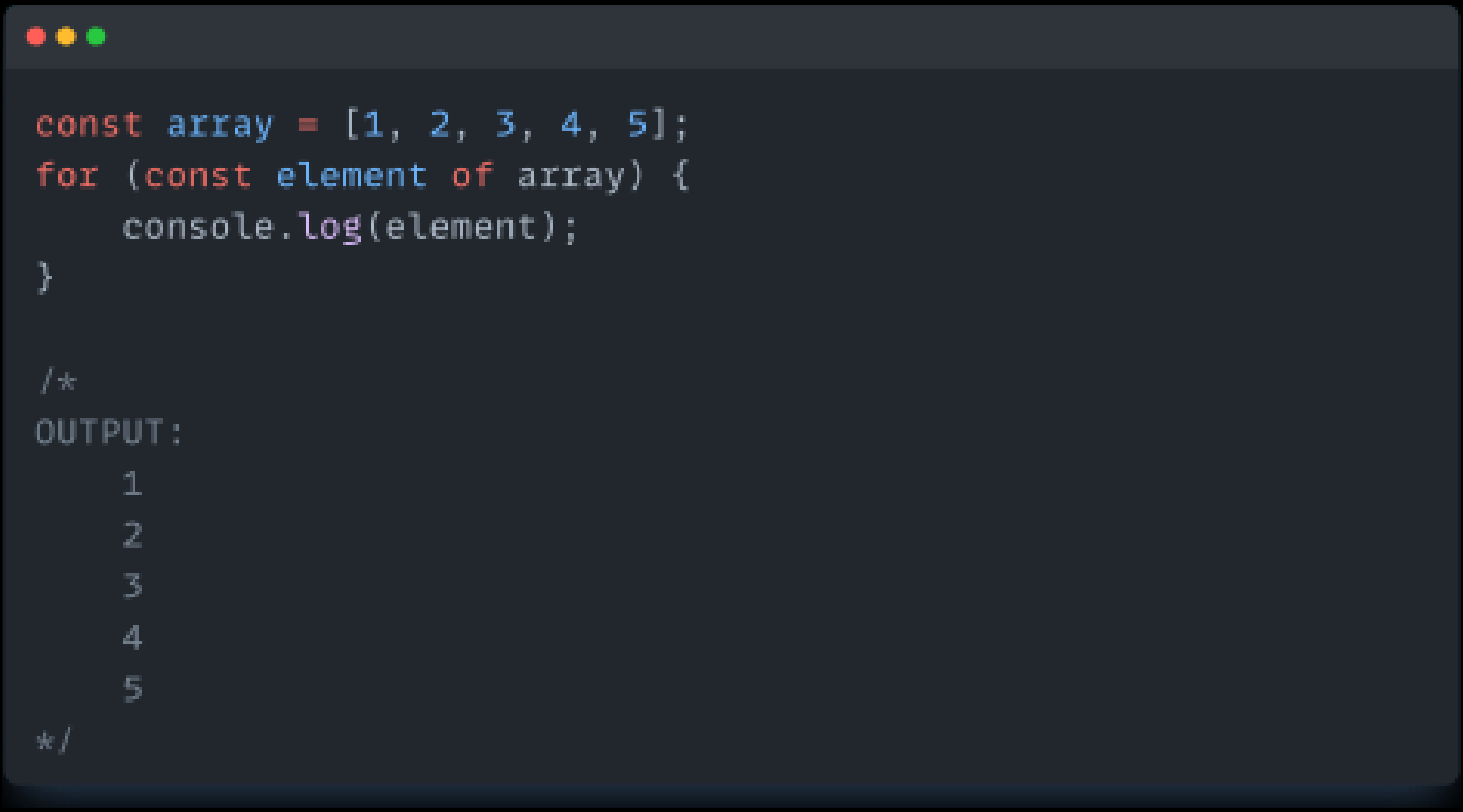
▼ We can use a traditional for loop to iterate over arrays by specifying the loop's initialization, condition, and iteration expressions.



```
const array = [1, 2, 3, 4, 5];
for (let i = 0; i < array.length; i++) {
    console.log(array[i]);
}

/*
OUTPUT:
    1
    2
    3
    4
    5
*/
```

JavaScript Arrays | Iteration with for...of loop

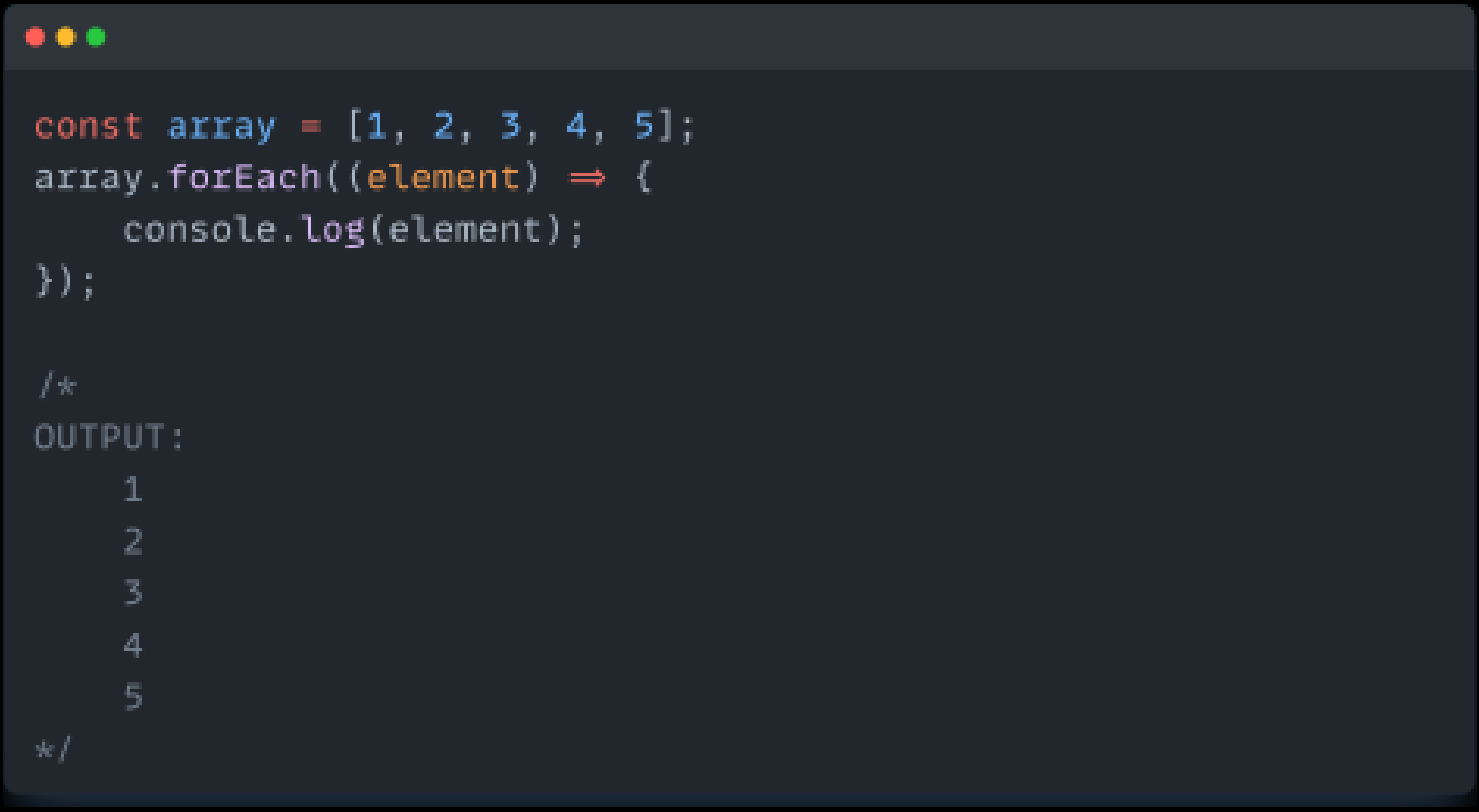
▼ The for...of loop is a modern syntax that allows you to iterate over the elements of an array directly without needing an index.



```
const array = [1, 2, 3, 4, 5];
for (const element of array) {
    console.log(element);
}

/*
OUTPUT:
    1
    2
    3
    4
    5
*/
```

JavaScript Arrays |
Iteration with forEach() method

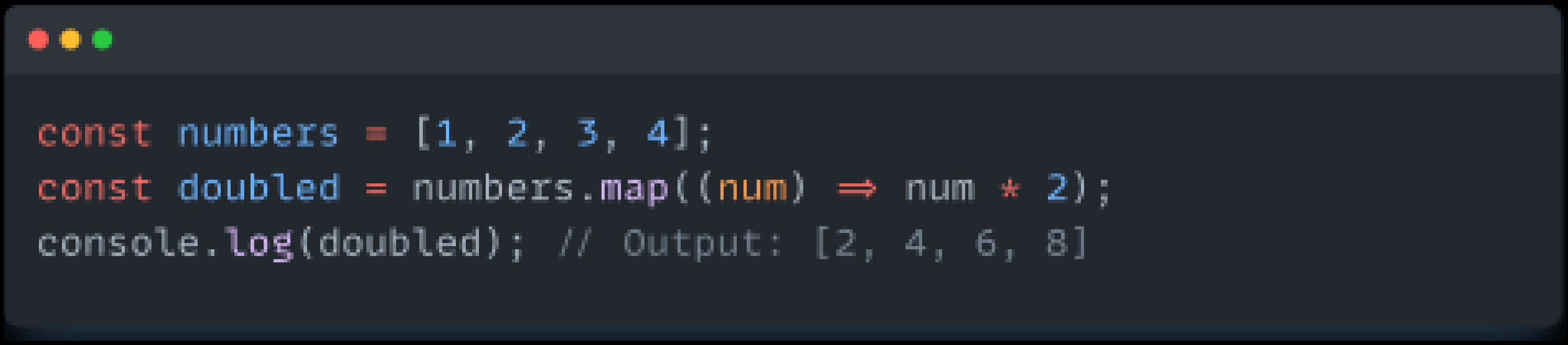
▼ The forEach() method is a built-in array method that executes a provided function once for each array element.



```
const array = [1, 2, 3, 4, 5];
array.forEach((element) => {
    console.log(element);
});

/*
OUTPUT:
    1
    2
    3
    4
    5
*/
```

JavaScript Arrays | map() method

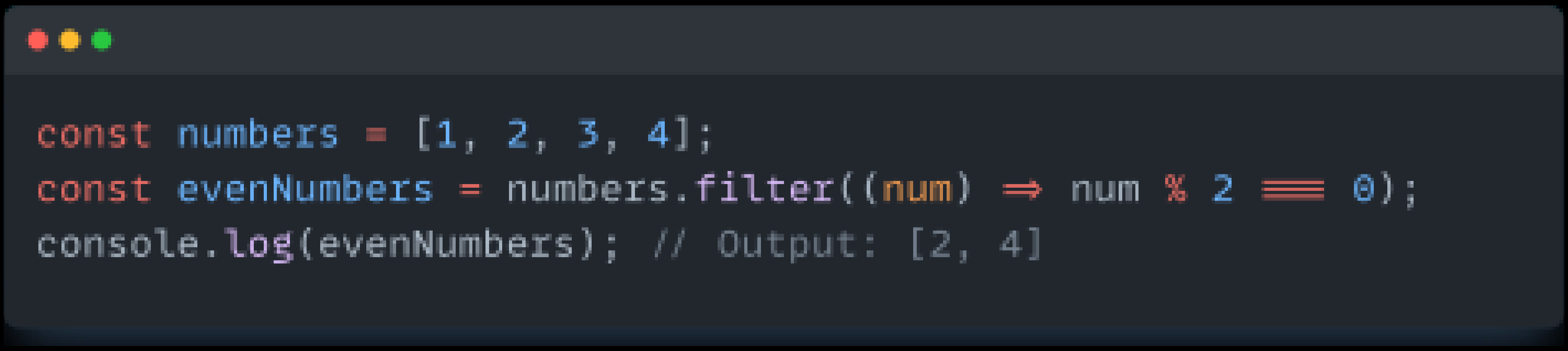
- The map() method creates a new array by applying a function to each element of the original array.
- It doesn't change the original array.
- It returns a new array with the same length as the original array, where each element is the result of applying the provided function to the corresponding element of the original array.
- Syntax: array.map(callback)
    - callback: A function to be called for each element in the array.



```
const numbers = [1, 2, 3, 4];
const doubled = numbers.map((num) ⇒ num * 2);
console.log(doubled); // Output: [2, 4, 6, 8]
```

JavaScript Arrays | filter() method

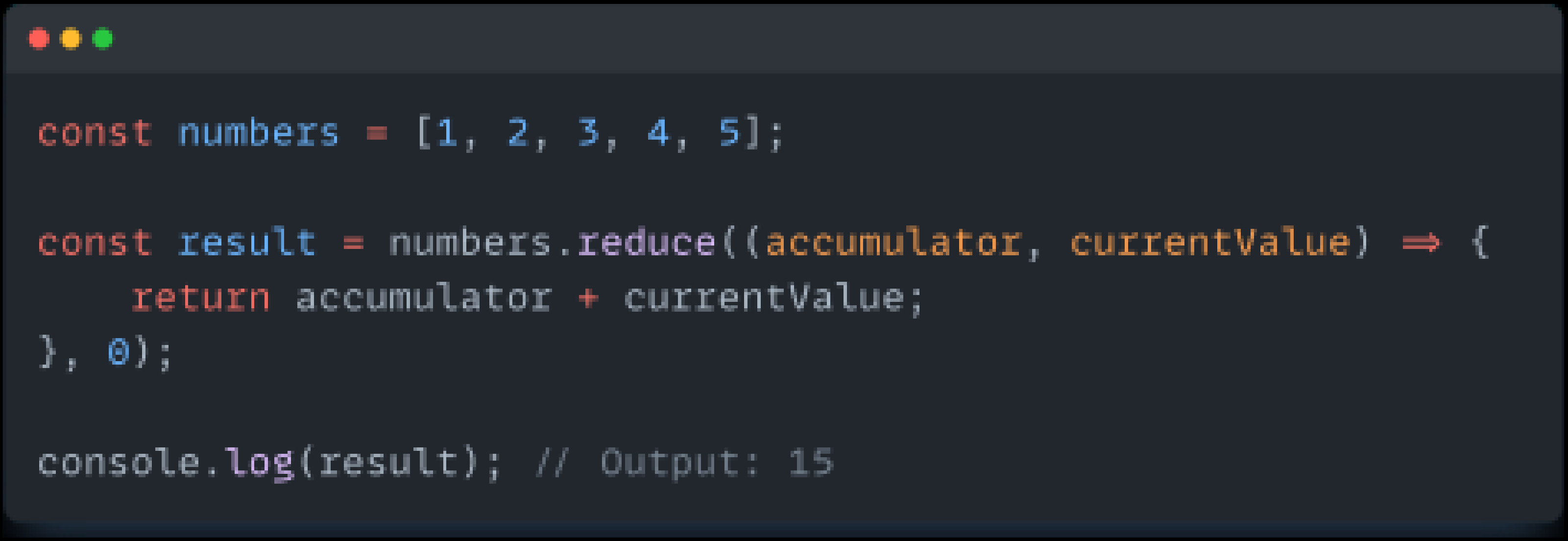
- The filter() method creates a new array with all elements that pass the test implemented by the provided function.
- It returns a new array containing only the elements of the original array that satisfy the condition specified in the callback function.
- It returns a new array with the same length as the original array, where each element is the result of applying the provided function to the corresponding element of the original array.
- Syntax: array.filter(callback)
  - callback: A function to test each element of the array.



```javascript
const numbers = [1, 2, 3, 4];
const evenNumbers = numbers.filter((num) => num % 2 === 0);
console.log(evenNumbers); // Output: [2, 4]
```

JavaScript Arrays | reduce() method

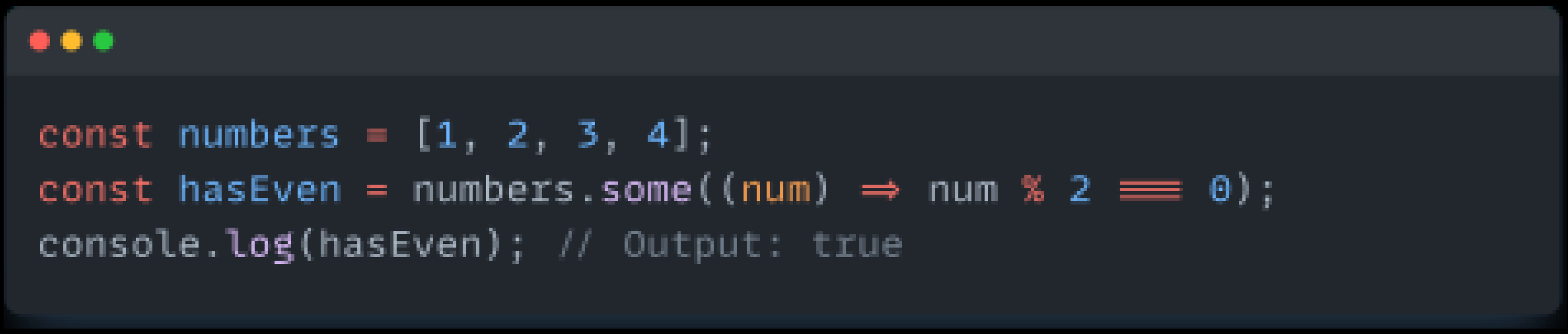
- The reduce() method applies a function against an accumulator and each element in the array to reduce it to a single value
- It returns a single value that is the result of applying the provided function to each element in the array.
- It doesn't change the original array.
- Syntax: array.reduce(callback, initialValue)
  - callback: A function to execute on each element in the array, taking four arguments:
    - accumulator: The accumulator accumulates the callback's return values. It is the accumulated value previously returned in the last invocation of the callback or the initialValue, if supplied.
    - currentValue: The current element being processed in the array.
    - index (optional): The index of the current element being processed in the array.
    - array (optional): The array reduce() was called upon.
    - The function should return the updated value of the accumulator.
  - initialValue (optional): A value to use as the first argument to the first call of the callback. If not provided, the first element of the array will be used as the initial value of the accumulator.



```
const numbers = [1, 2, 3, 4, 5];

const result = numbers.reduce((accumulator, currentValue) => {
    return accumulator + currentValue;
}, 0);

console.log(result); // Output: 15
```

JavaScript Arrays | some() method

- The some() method tests whether at least one element in the array passes the test implemented by the provided function.
- It returns true if at least one element in the array satisfies the testing function; otherwise, it returns false.
- It doesn't change the original array.
- Syntax:array.some(callback)
  - callback: A function to test each element of the array.

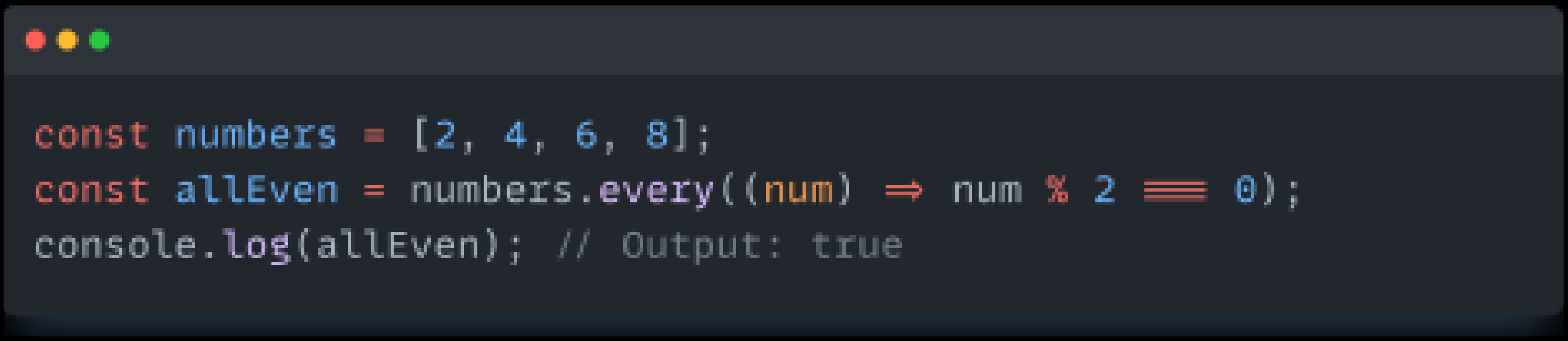


```
const numbers = [1, 2, 3, 4];
const hasEven = numbers.some((num) => num % 2 === 0);
console.log(hasEven); // Output: true
```

TechGlobal School

JavaScript Arrays | every() method

- The every() method tests whether all elements in the array pass the test implemented by the provided function.
- It returns true if all elements in the array satisfy the testing function; otherwise, it returns false.
- It doesn't change the original array.
- Syntax: array.every(callback)
  - callback: A function to test each element of the array.

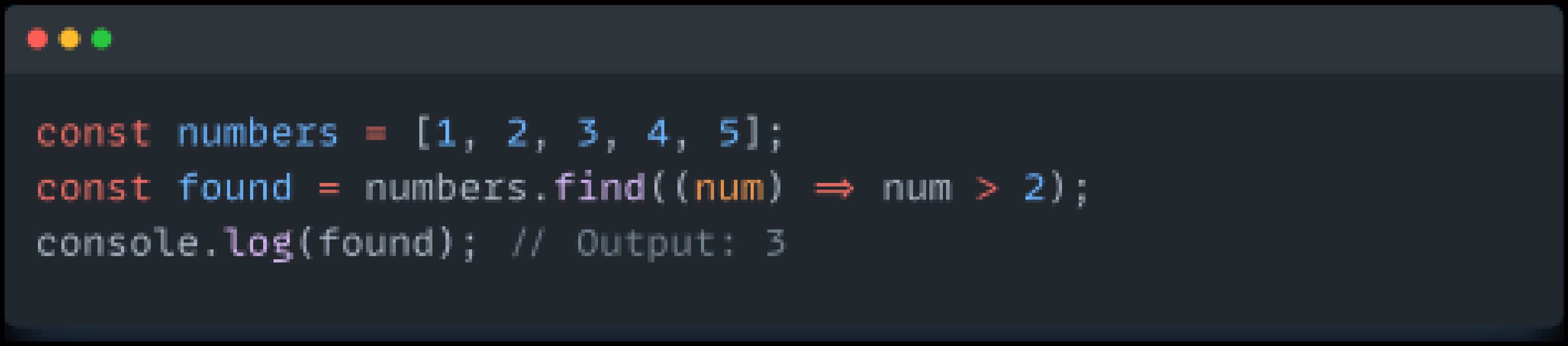


```javascript
const numbers = [2, 4, 6, 8];
const allEven = numbers.every((num) => num % 2 === 0);
console.log(allEven); // Output: true
```

TechGlobal School

JavaScript Arrays | find() method

- The find() method returns the value of the first element in the array that satisfies the provided testing function.
- It searches the array from left to right, and once an element passes the test, it stops searching and returns that element's value.
- It doesn't change the original array.
- It returns the value of the first element in the array that satisfies the testing function. If no such element is found, it returns undefined.
- Syntax:array.find(callback)
    - callback: A function to test each element of the array.

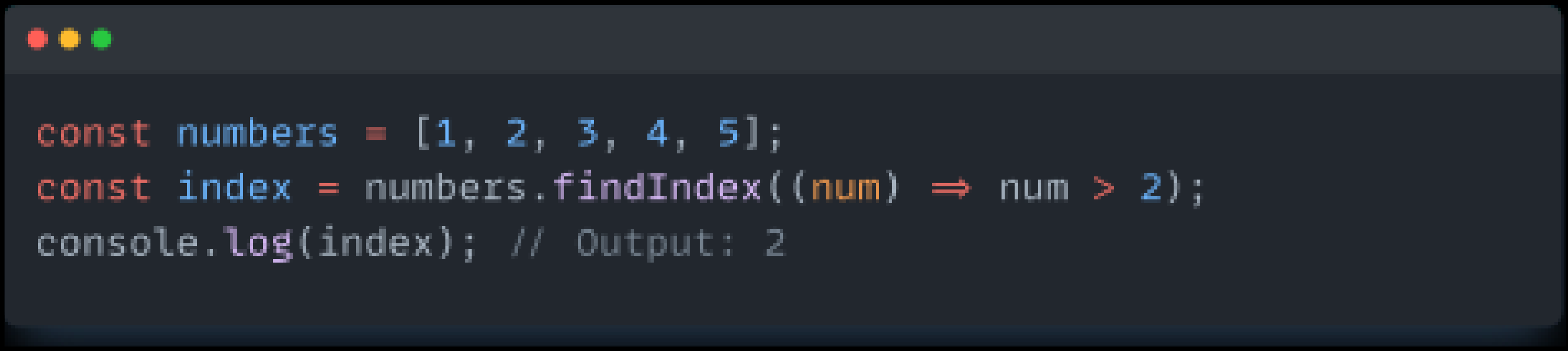


```
const numbers = [1, 2, 3, 4, 5];
const found = numbers.find((num) => num > 2);
console.log(found); // Output: 3
```

TechGlobal School

JavaScript Arrays | findIndex() method

- The findIndex() method returns the index of the first element in the array that satisfies the provided testing function.
- It searches the array from left to right, and once an element passes the test, it stops searching and returns the index of that element.
- It doesn't change the original array.
- It returns the index of the first element in the array that satisfies the testing function. If no such element is found, it returns -1.
- Syntax:array.findIndex(callback)
  - callback: A function to test each element of the array.



```
const numbers = [1, 2, 3, 4, 5];
const index = numbers.findIndex((num) => num > 2);
console.log(index); // Output: 2
```

Tech Global

Follow Us !

techglobal.school

techglobalschool

techglobalschool

techglobalschl

techglobalschool

www.techglobalschool.com