Array Methods...

1. values()
2. length()
3. reverse()
4. sort()
5. at()
6. fill()
7. from()
8. join()
9. toString()
10. pop()
11. forEach()
12. shift()
13. copyWithin()
14. push()
15. unshift()
16. concat()
17. some()
18. splice()
19. flat()
20. lastIndexOf()
21. of()
22. every()
23. slice()
24. flatMap()
25. findIndex()
26. find()
27. inculdes()
28. entries()
29. reduceRight()
30. reduce()
31. isArray()
32. filter()
33. keys()
34. map()

values(): This method returns an iterator that provides the values for each index in the array. It takes no arguments.

```
const arr = ['apple', 'banana', 'cherry']
const iterator = arr.values();

for (const value of iterator) {
  console.log(value);
} // Output: apple banana cherry
```

length(): This property returns the length of the array.

```
const arr = ['apple', 'banana', 'cherry']
console.log(arr.length); // Output: 3
```

reverse(): This method reverses the order of the elements in the array.

```javascript
const arr = ['apple', 'banana', 'cherry'];
arr.reverse();
console.log(arr); // Output: ['cherry', 'banana', 'apple']
```

sort(): This method sorts the elements of an array in place and returns the sorted array. It can take an optional compare function as an argument.

```javascript
const arr = ['banana', 'apple', 'cherry'];
arr.sort();
console.log(arr); // Output: ['apple', 'banana', 'cherry']
```

at(): This method returns the element at the specified index in the array. It takes one argument: the index.

```
const arr = ['apple', 'banana', 'cherry'];
console.log(arr.at(1)); // Output: 'banana'
```

fill(): This method fills all the elements of an array from a start index to an end index with a static value. It can take up to three arguments: the value to fill with, the start index, and the end index.

```
const arr = ['apple', 'banana', 'cherry'];
arr.fill('orange', 1, 2);
console.log(arr); // Output: ['apple', 'orange', 'cherry']
```

»»

from(): This method creates a new array from an array-like object or an iterable object. It can take up to two arguments: the object to convert to an array, and a mapping function to apply to each element of the new array.

```
const obj = { 0: 'apple', 1: 'banana', 2: 'cherry', length: 3 }
const arr = Array.from(obj);
console.log(arr); // Output: ['apple', 'banana', 'cherry']
```

join(): This method joins all the elements of an array into a string using a specified separator. It takes one optional argument: the separator to use.

```
const arr = ['apple', 'banana', 'cherry'];
const str = arr.join(', ');
console.log(str); // Output: 'apple, banana, cherry'
```

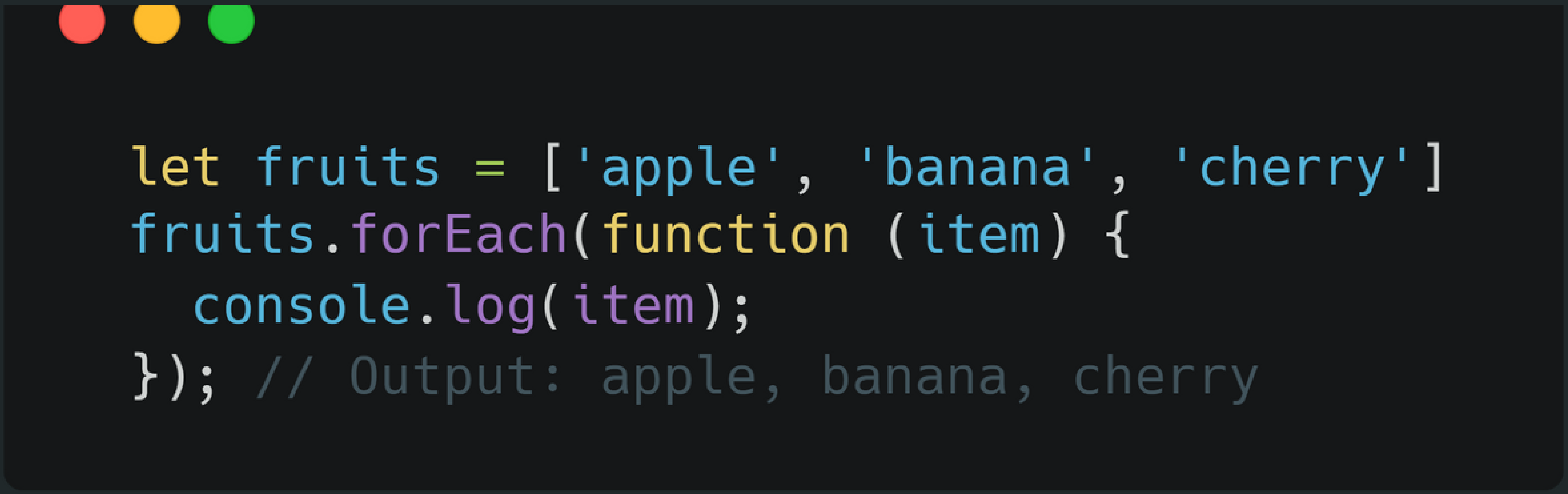
toString(): This method returns a string representing the array and its elements.

```
const arr = ['apple', 'banana', 'cherry'];
const str = arr.toString();
console.log(str); // Output: 'apple,banana,cherry'
```

pop(): This method removes the last element from an array and returns that element.

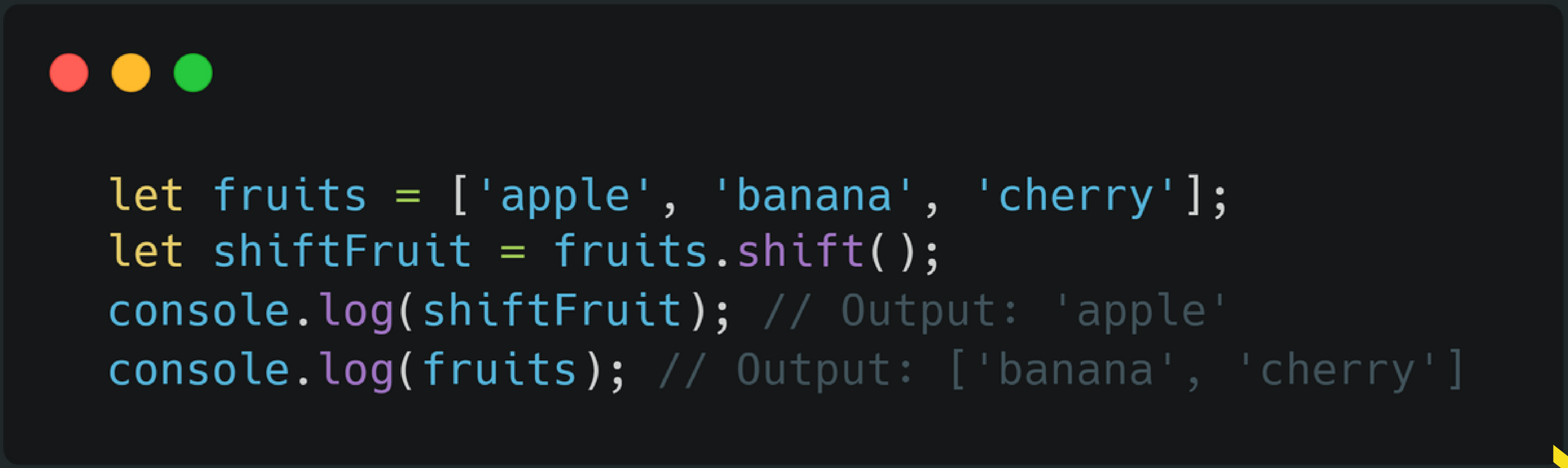
```
const arr = ['apple', 'banana', 'cherry'];
const last = arr.pop();
console.log(last); // Output: 'cherry'
console.log(arr); // Output: ['apple', 'banana']
```

forEach() method executes a provided function once for each array element. It doesn't return anything, it just executes the callback function on each element of the array.



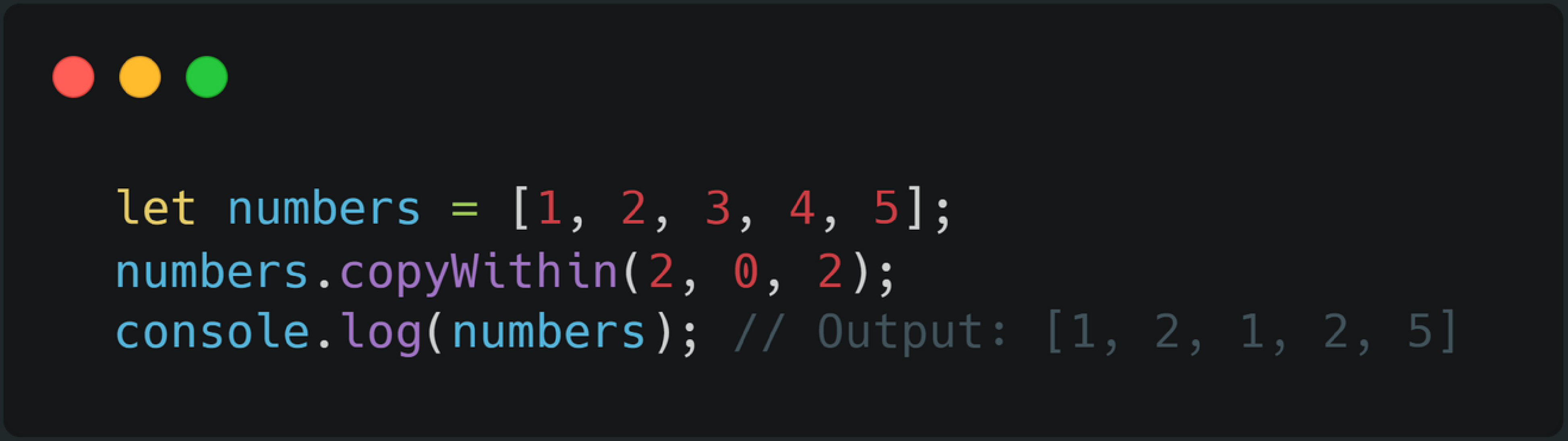
```
let fruits = ['apple', 'banana', 'cherry']
fruits.forEach(function (item) {
  console.log(item);
}); // Output: apple, banana, cherry
```

shift() method removes the first element from an array and returns that removed element. This method changes the length of the array.



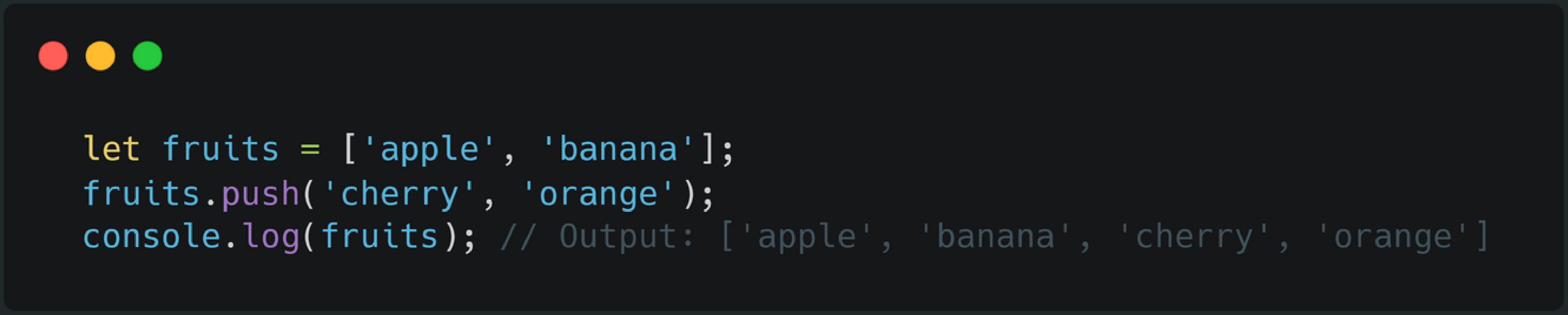
```
let fruits = ['apple', 'banana', 'cherry'];
let shiftFruit = fruits.shift();
console.log(shiftFruit); // Output: 'apple'
console.log(fruits); // Output: ['banana', 'cherry']
```

copyWithin() method shallow copies part of an array to another location in the same array and returns the modified array without modifying its length.**Syntax** .copyWithin(target, start, end)



```javascript
let numbers = [1, 2, 3, 4, 5];
numbers.copyWithin(2, 0, 2);
console.log(numbers); // Output: [1, 2, 1, 2, 5]
```

push() method 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('cherry', 'orange');
console.log(fruits); // Output: ['apple', 'banana', 'cherry', 'orange']
```

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

```
let fruits = ['cherry', 'orange'];
fruits.unshift('apple', 'banana');
console.log(fruits); // Output: ['apple', 'banana', 'cherry', 'orange']
```

concat() method is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

```
let fruits = ['apple', 'banana'];
let moreFruits = ['cherry', 'orange'];
let allFruits = fruits.concat(moreFruits);
console.log(allFruits); // Output: ['apple', 'banana', 'cherry', 'orange']
```

splice() method changes the contents of an array by removing or replacing existing elements and/or adding new elements in place.

```
  const fruits = ['apple', 'banana', 'cherry', 'orange'];
  //Syntax : arr.splice(start, deleteCount, item1, ..., itemN)
  fruits.splice(2, 1, 'mango', 'kiwi');
  console.log(fruits); // Output: [ 'apple', 'banana', 'mango', 'kiwi', 'orange'
]
```

flat() This method creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.

```
const numbers = [1, [2, [3]], 4];
const flatNumbers = numbers.flat(Infinity);
console.log(flatNumbers); // Output: [1, 2, 3, 4]
```

lastIndexOf() This method returns the last index at which a given element can be found in the array.

```
const numbers = [1, 2, 3, 4, 5, 3];
const lastIndex = numbers.lastIndexOf(3);
console.log(lastIndex); // Output: 5
```

indexOf(): This method returns the index of the first occurrence of a specified element in an array. If the element is not present, it returns -1.

```
const arr = [5, 10, 15, 20];
const index = arr.indexOf(10);
console.log(index); // 1
```

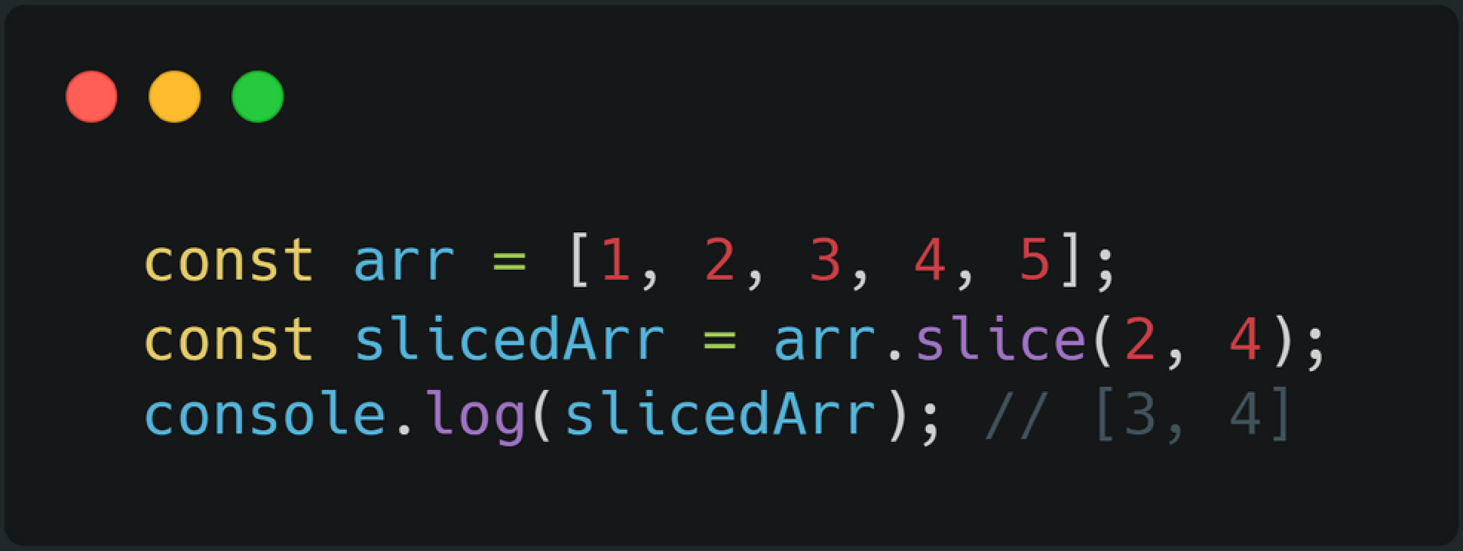
of(): This method creates a new array instance with a variable number of arguments, regardless of number or type of the arguments.

```javascript
const arr = Array.of(1, 2, 3, "four", true);
console.log(arr); // [1, 2, 3, "four", true]
```

every(): This method checks if all elements in an array pass a test (provided as a function). It returns true if all elements pass the test; otherwise, it returns false.

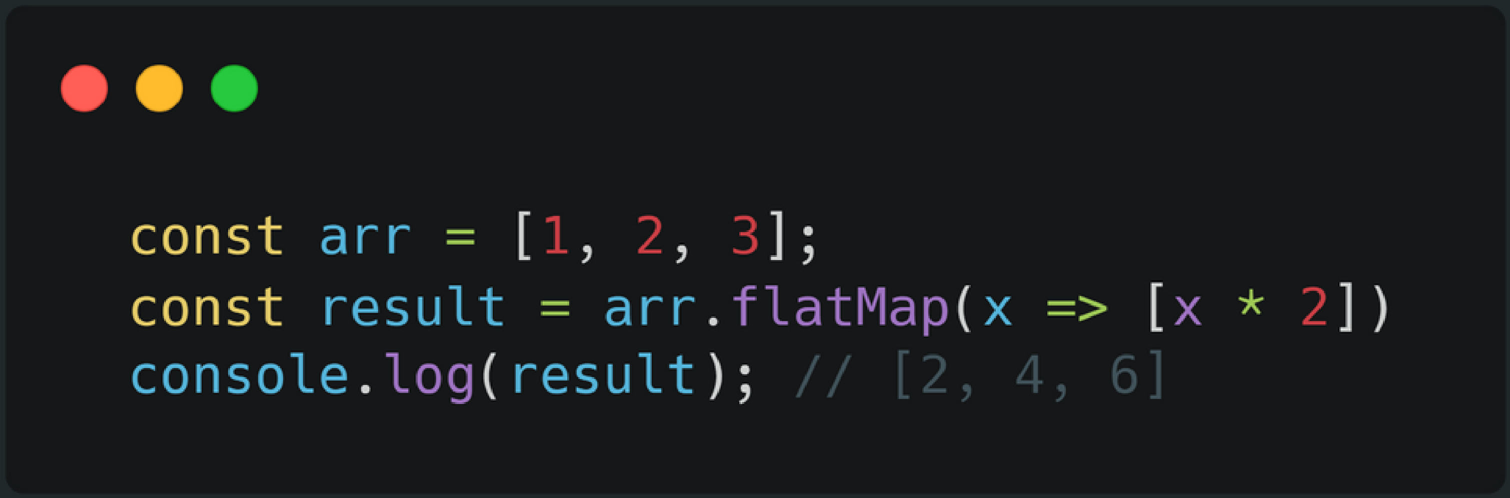
```javascript
const arr = [2, 4, 6, 8];
const isEven = (num) => num % 2 === 0;
const result = arr.every(isEven);
console.log(result); // true
```

slice(): This method returns a shallow copy of a portion of an array into a new array object selected from begin to end (end not included).



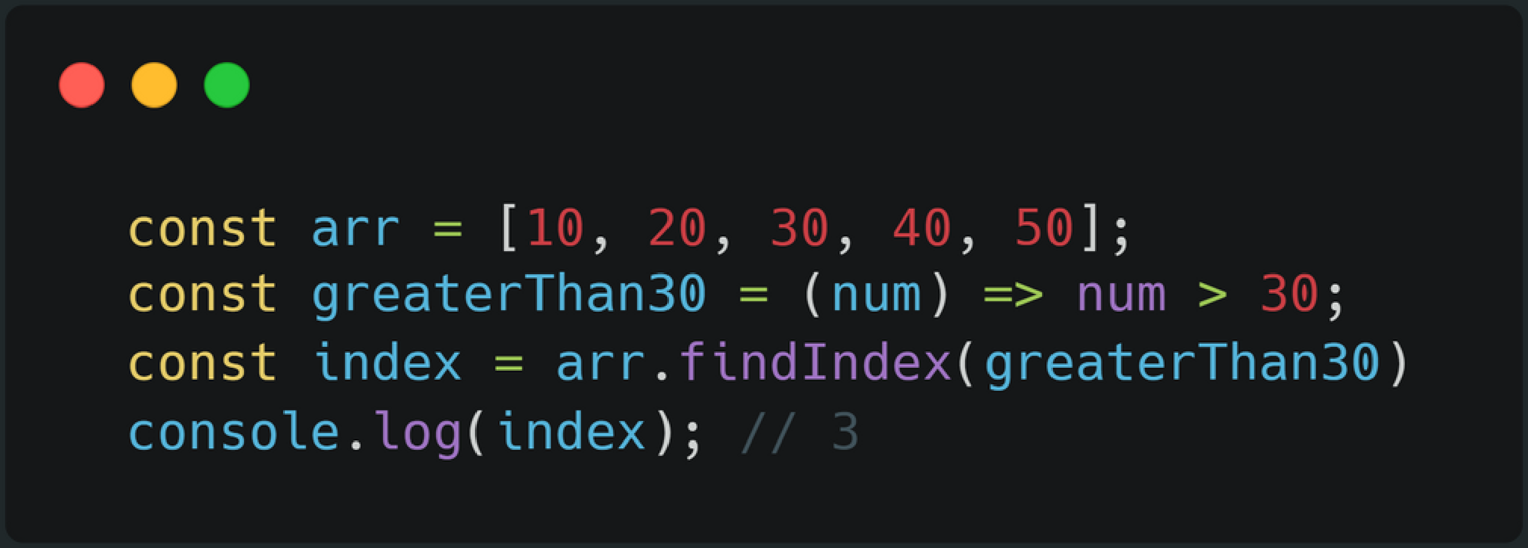
```
const arr = [1, 2, 3, 4, 5];
const slicedArr = arr.slice(2, 4);
console.log(slicedArr); // [3, 4]
```

flatMap(): This method maps each element using a mapping function, then flattens the result into a new array.



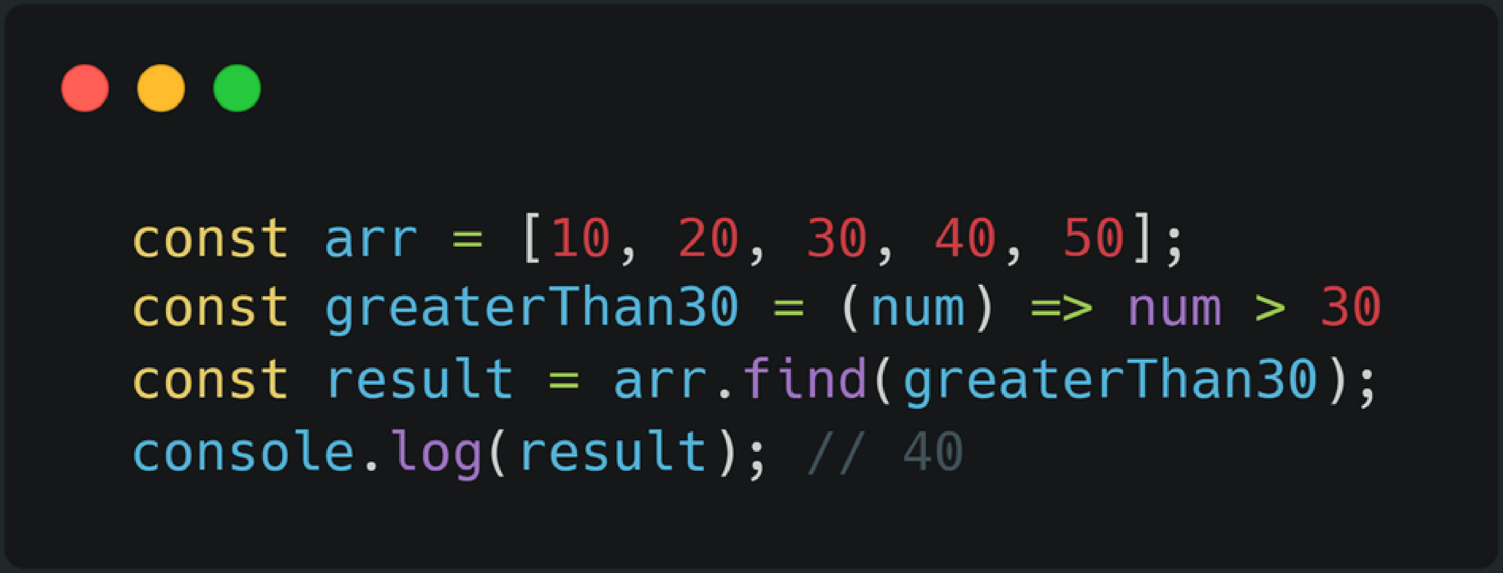
```
const arr = [1, 2, 3];
const result = arr.flatMap(x => [x * 2])
console.log(result); // [2, 4, 6]
```

findIndex(): This method returns the index of the first element in an array that passes a test (provided as a function). If no element passes the test, it returns -1.



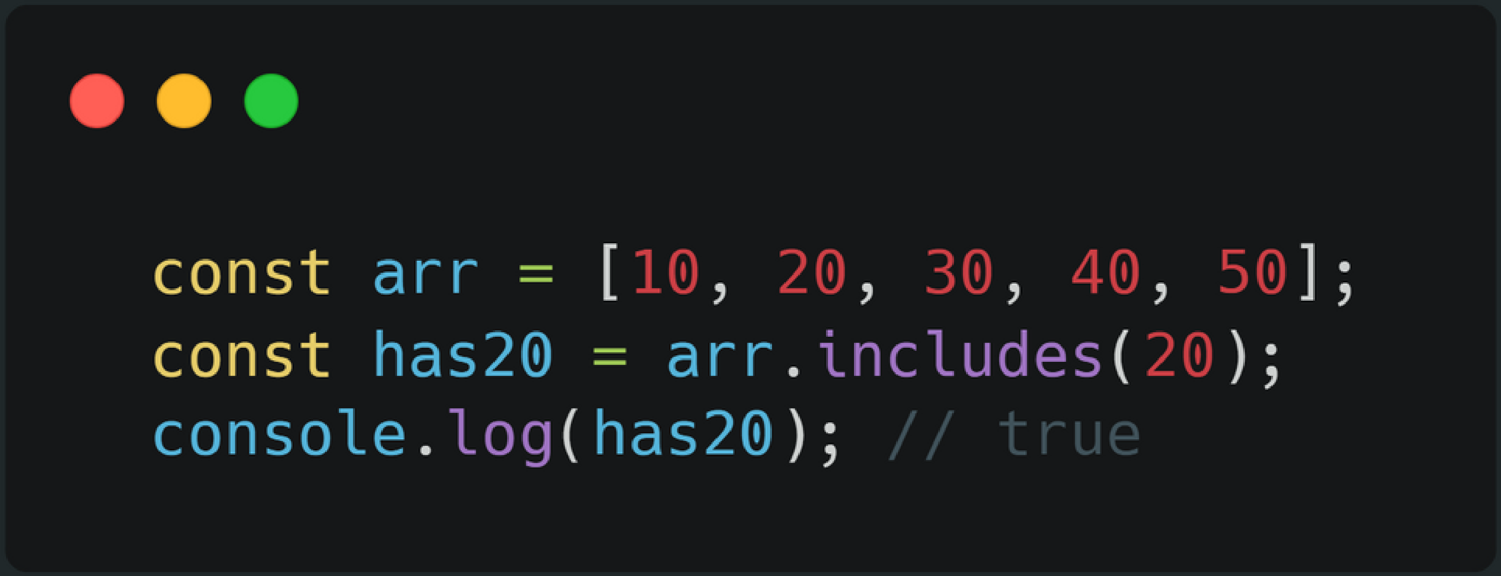
```javascript
const arr = [10, 20, 30, 40, 50];
const greaterThan30 = (num) => num > 30;
const index = arr.findIndex(greaterThan30)
console.log(index); // 3
```

find(): This method returns the value of the first element in an array that passes a test (provided as a function). If no element passes the test, it returns undefined.



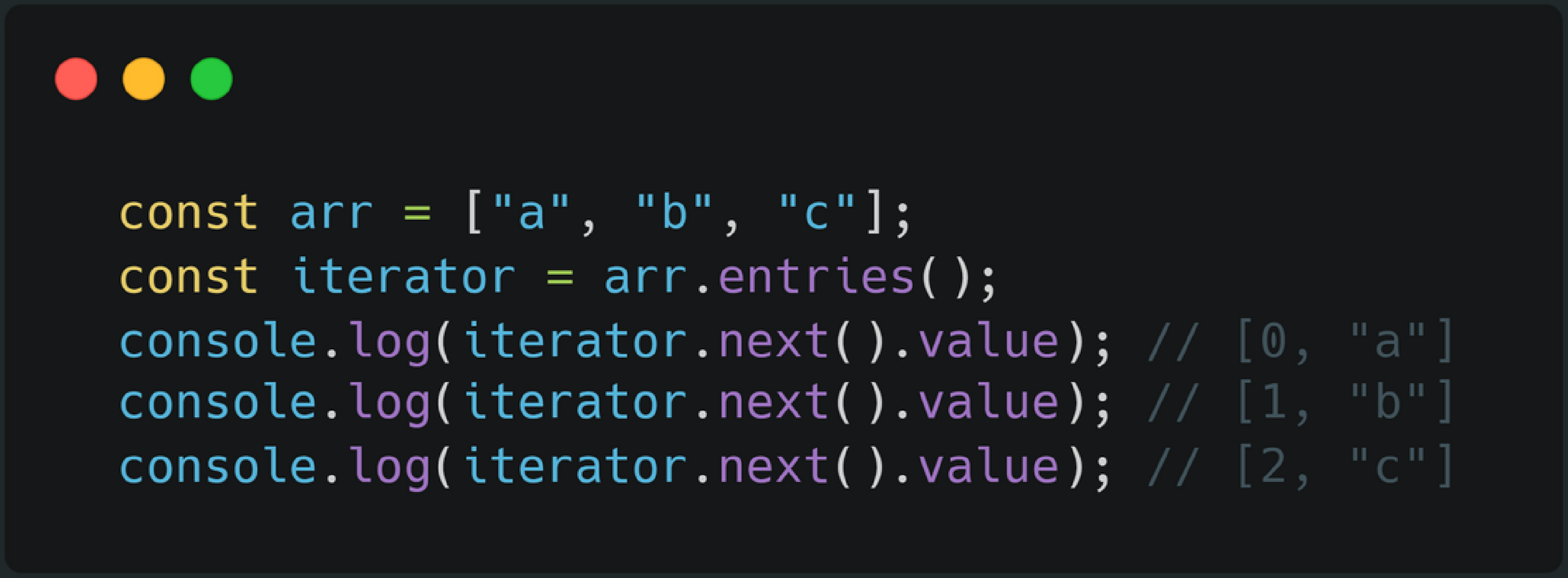
```javascript
const arr = [10, 20, 30, 40, 50];
const greaterThan30 = (num) => num > 30
const result = arr.find(greaterThan30);
console.log(result); // 40
```

includes(): This method determines whether an array includes a certain value among its entries, returning true or false as appropriate.



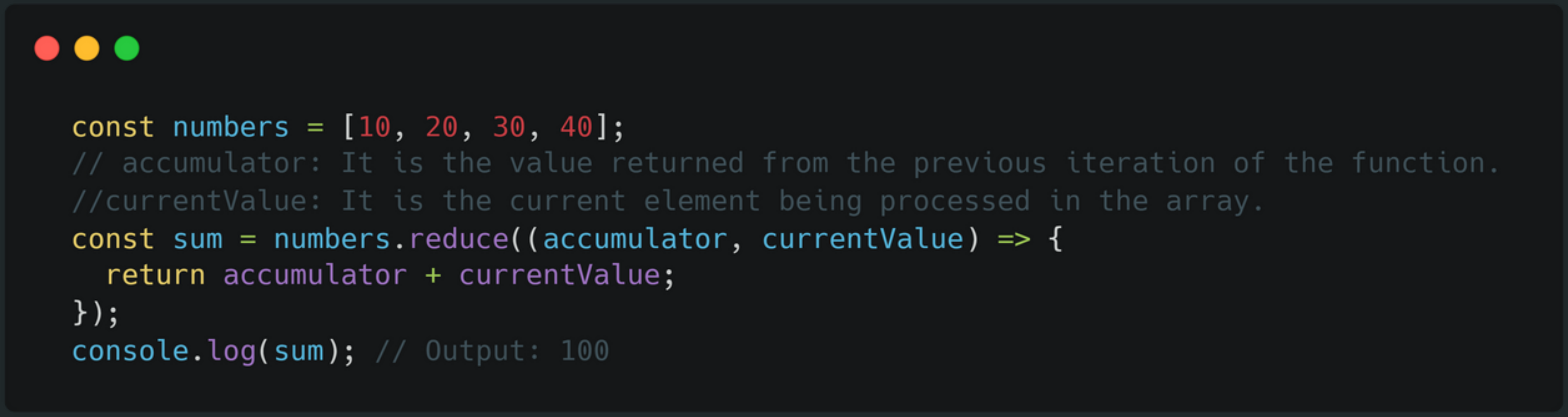
```javascript
const arr = [10, 20, 30, 40, 50];
const has20 = arr.includes(20);
console.log(has20); // true
```

entries(): This method returns a new Array Iterator object that contains the key/value pairs for each index in the array.



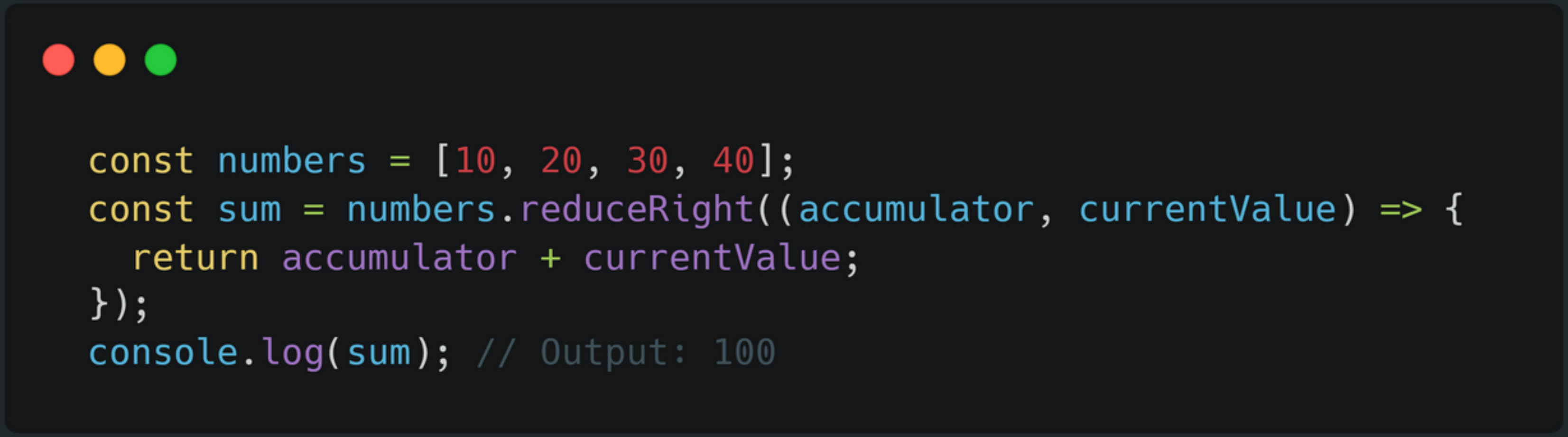
```javascript
const arr = ["a", "b", "c"];
const iterator = arr.entries();
console.log(iterator.next().value); // [0, "a"]
console.log(iterator.next().value); // [1, "b"]
console.log(iterator.next().value); // [2, "c"]
```

reduce(): This method applies a function to each element of an array and reduces the array to a single value.



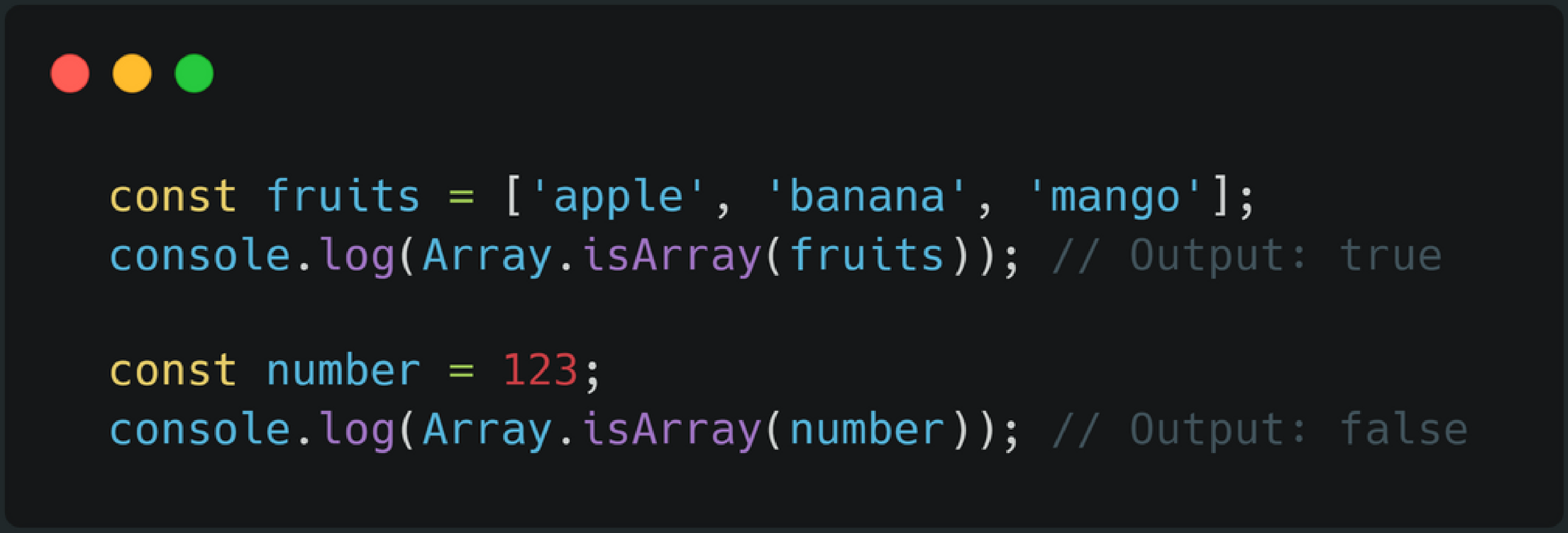
```
const numbers = [10, 20, 30, 40];
// accumulator: It is the value returned from the previous iteration of the function.
//currentValue: It is the current element being processed in the array.
const sum = numbers.reduce((accumulator, currentValue) => {
  return accumulator + currentValue;
});
console.log(sum); // Output: 100
```

reduceRight(): This method is similar to the reduce() method. However, it iterates over the array elements from right to left instead of from left to right.



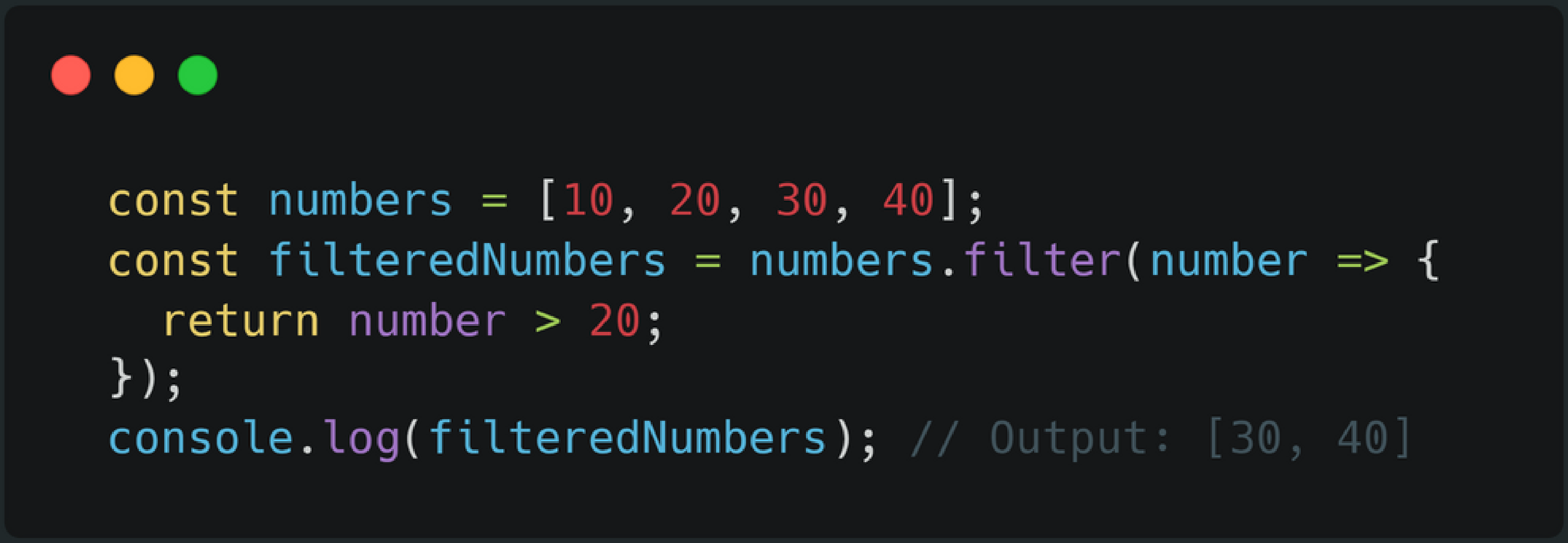
```
const numbers = [10, 20, 30, 40];
const sum = numbers.reduceRight((accumulator, currentValue) => {
  return accumulator + currentValue;
});
console.log(sum); // Output: 100
```

isArray(): This method determines whether the passed value is an array or not.



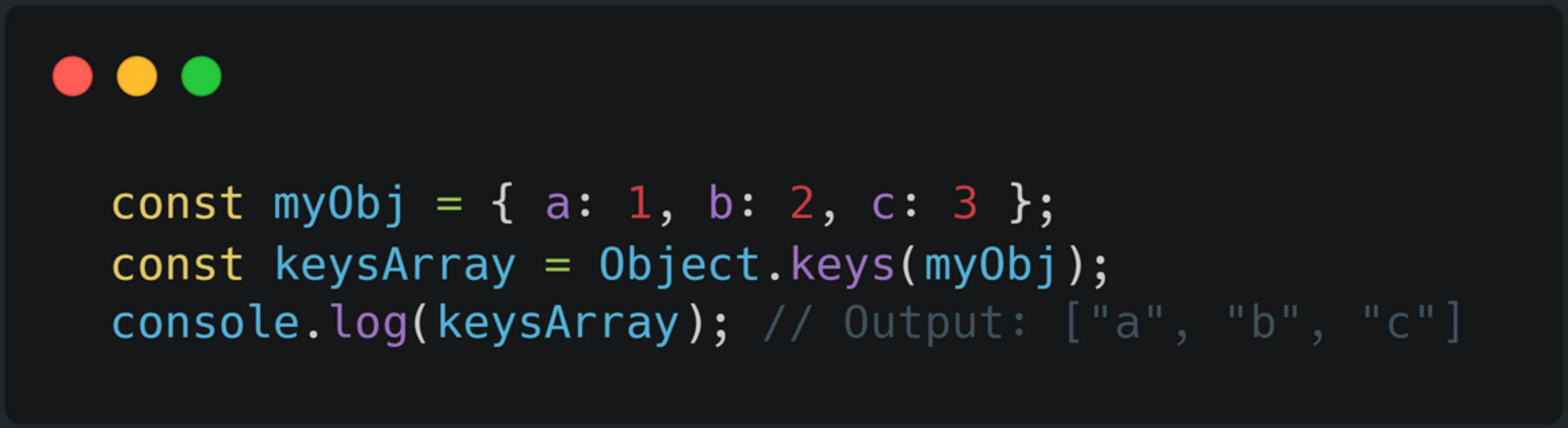
```
const fruits = ['apple', 'banana', 'mango'];
console.log(Array.isArray(fruits)); // Output: true

const number = 123;
console.log(Array.isArray(number)); // Output: false
```

filter(): This method creates a new array with all elements that pass the test implemented by the provided function.



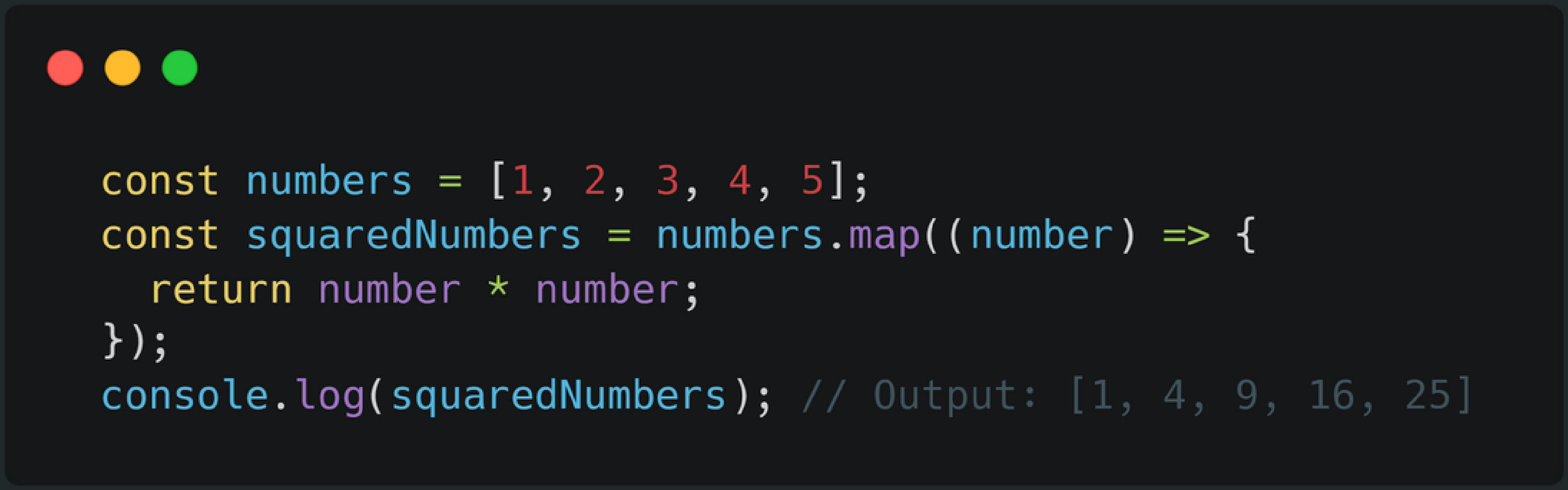
```
const numbers = [10, 20, 30, 40];
const filteredNumbers = numbers.filter(number => {
  return number > 20;
});
console.log(filteredNumbers); // Output: [30, 40]
```

keys(): This method returns an array containing the keys of the given object.



```
const myObj = { a: 1, b: 2, c: 3 };
const keysArray = Object.keys(myObj);
console.log(keysArray); // Output: ["a", "b", "c"]
```

map(): This method creates a new array with the results of calling a provided function on every element in the calling array.



```
const numbers = [1, 2, 3, 4, 5];
const squaredNumbers = numbers.map((number) => {
  return number * number;
});
console.log(squaredNumbers); // Output: [1, 4, 9, 16, 25]
```