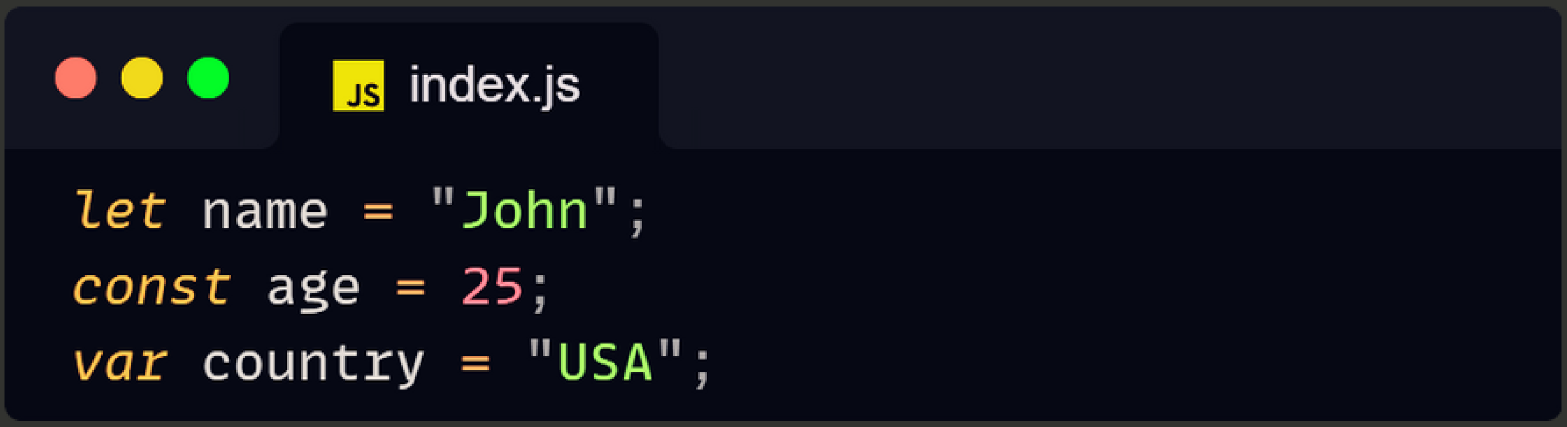
JavaScript 101

Must-Know Concepts

Beginner's Guide

1. Variables and Data Types

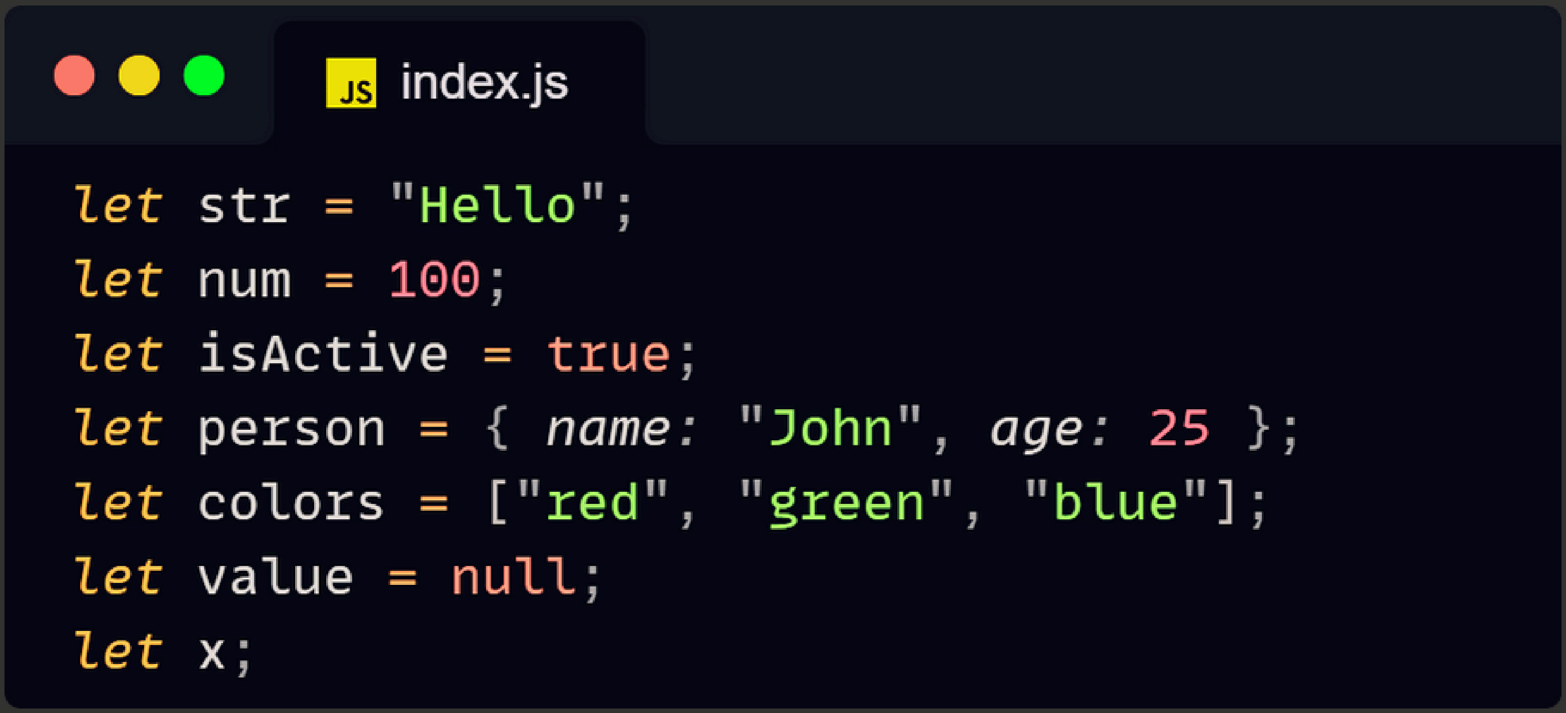
- **Variables:**



```javascript
let name = "John";
const age = 25;
var country = "USA";
```

- **let** allows reassignment, so you can change the value later.
- **const** makes the value immutable, meaning you cannot reassign this variable.
- **var** is function-scoped, but it's best to avoid using it in modern JS.

- **Data Types :**



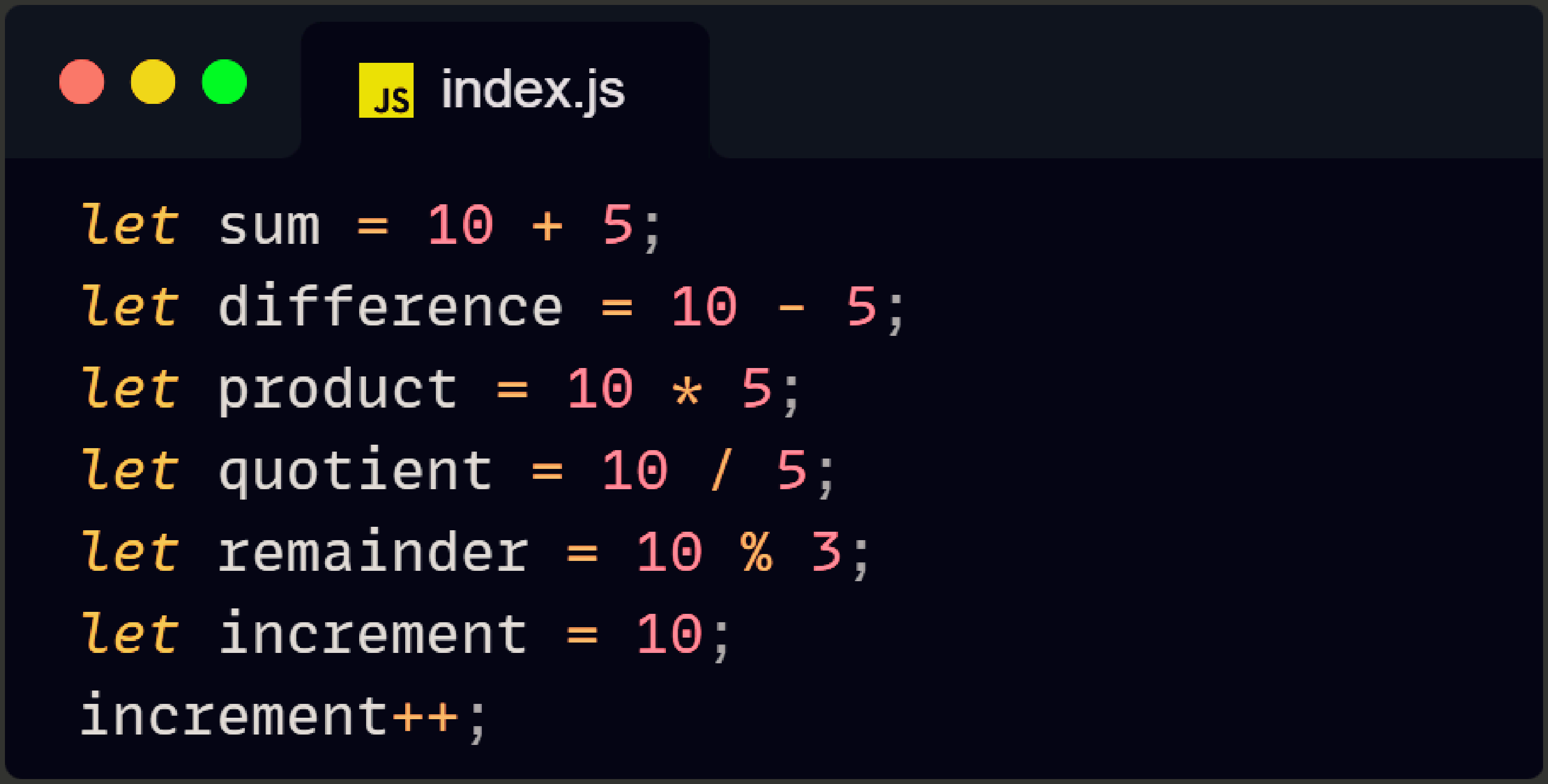
```
let str = "Hello";
let num = 100;
let isActive = true;
let person = { name: "John", age: 25 };
let colors = ["red", "green", "blue"];
let value = null;
let x;
```

- **str**: A string type used to store text values.
- **num**: A number type used to represent integers or floating-point numbers.
- **isActive**: A boolean type that can either be true or false.
- **person**: An object used to store key-value pairs.
- **colors**: An array representing an ordered list of values.
- **value**: Represents the intentional absence of any value or object.
- **x**: Undefined is the default value of a variable that hasn't been assigned any value.

Note:
The variable name does not determine its type; it's just a label.
The type is defined by the value assigned to it.

2. Operators

- **Arithmetic Operators :**



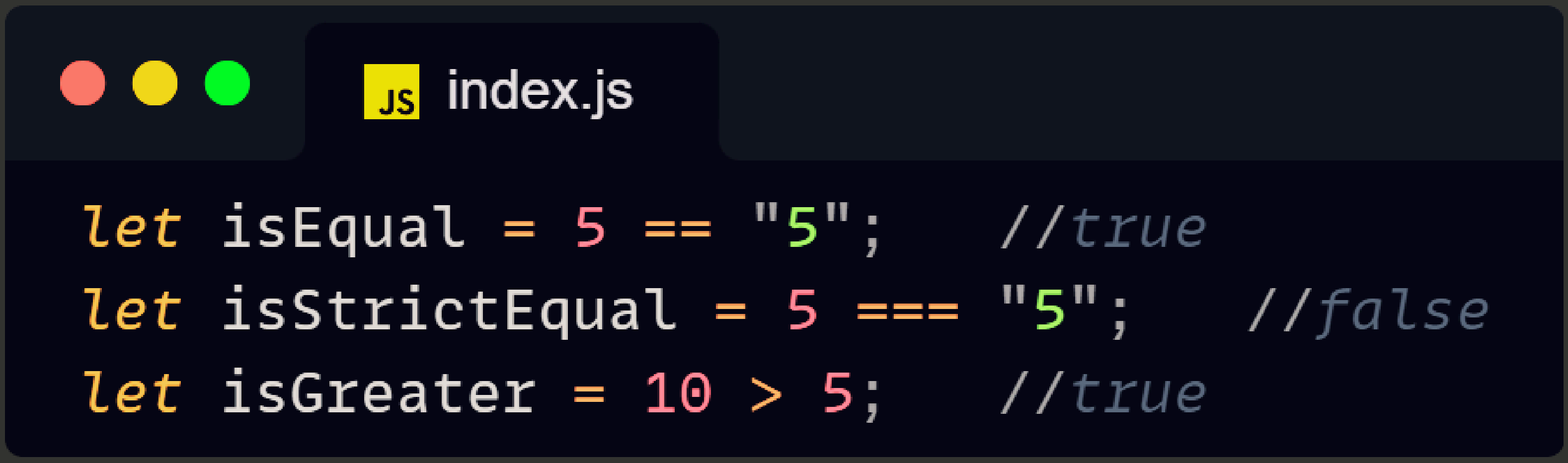
index.js

```javascript
let sum = 10 + 5;
let difference = 10 - 5;
let product = 10 * 5;
let quotient = 10 / 5;
let remainder = 10 % 3;
let increment = 10;
increment++;
```

- **sum**: The result of addition (10 + 5), which is 15.
- **difference**: The result of subtraction (10 - 5), which is 5.
- **product**: The result of multiplication (10 * 5), which is 50.
- **quotient**: The result of division (10 / 5), which is 2.
- **remainder**: The remainder of the division (10 % 3), which is 1.
- **increment**: Starts at 10 and is increased by 1 using the increment operator (++), resulting in

Note:
The variable name is independent of the operation performed or its result; it's just a label to store the value.

- **Comparison Operators :**

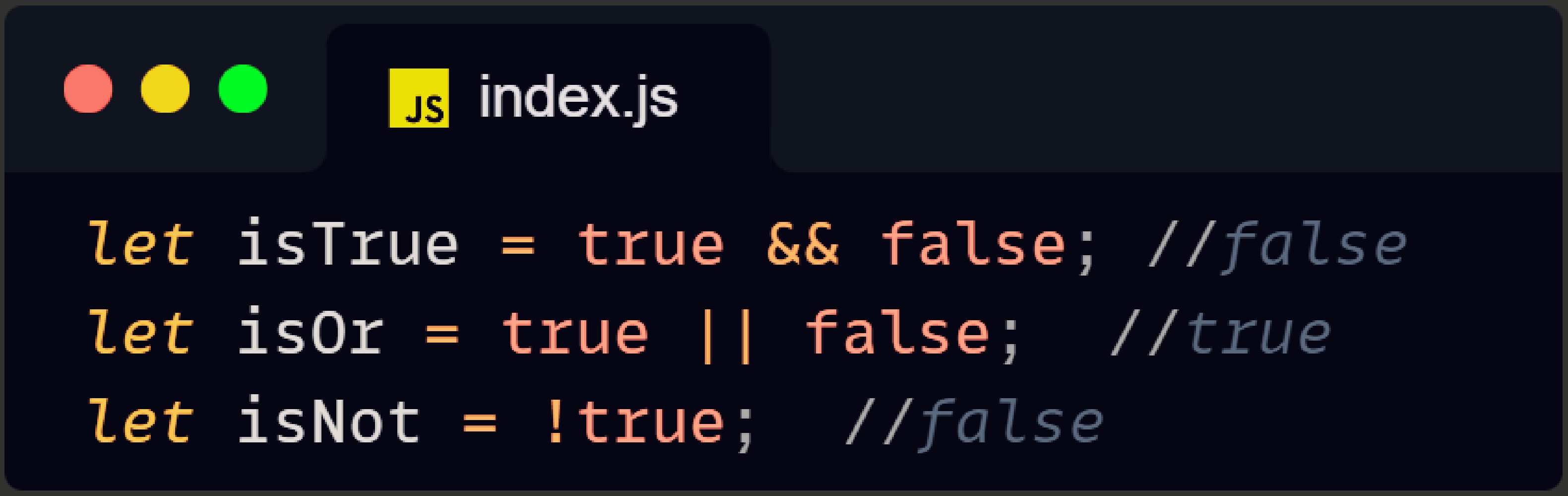


```javascript
let isEqual = 5 == "5";    //true
let isStrictEqual = 5 === "5";    //false
let isGreater = 10 > 5;    //true
```

- **isEqual**: Uses the equality operator (==) to compare values only, so 5 == "5" returns true.
- **isStrictEqual**: Uses the strict equality operator (===) to compare both value and type, so 5 === "5" returns false.
- **isGreater**: Uses the greater-than operator (>) to check if 10 is greater than 5, returning true.

Note:
The variable name does not affect the behavior of the comparison operators; it's simply a label to store the result of the operation.

- **Logical Operators :**



index.js

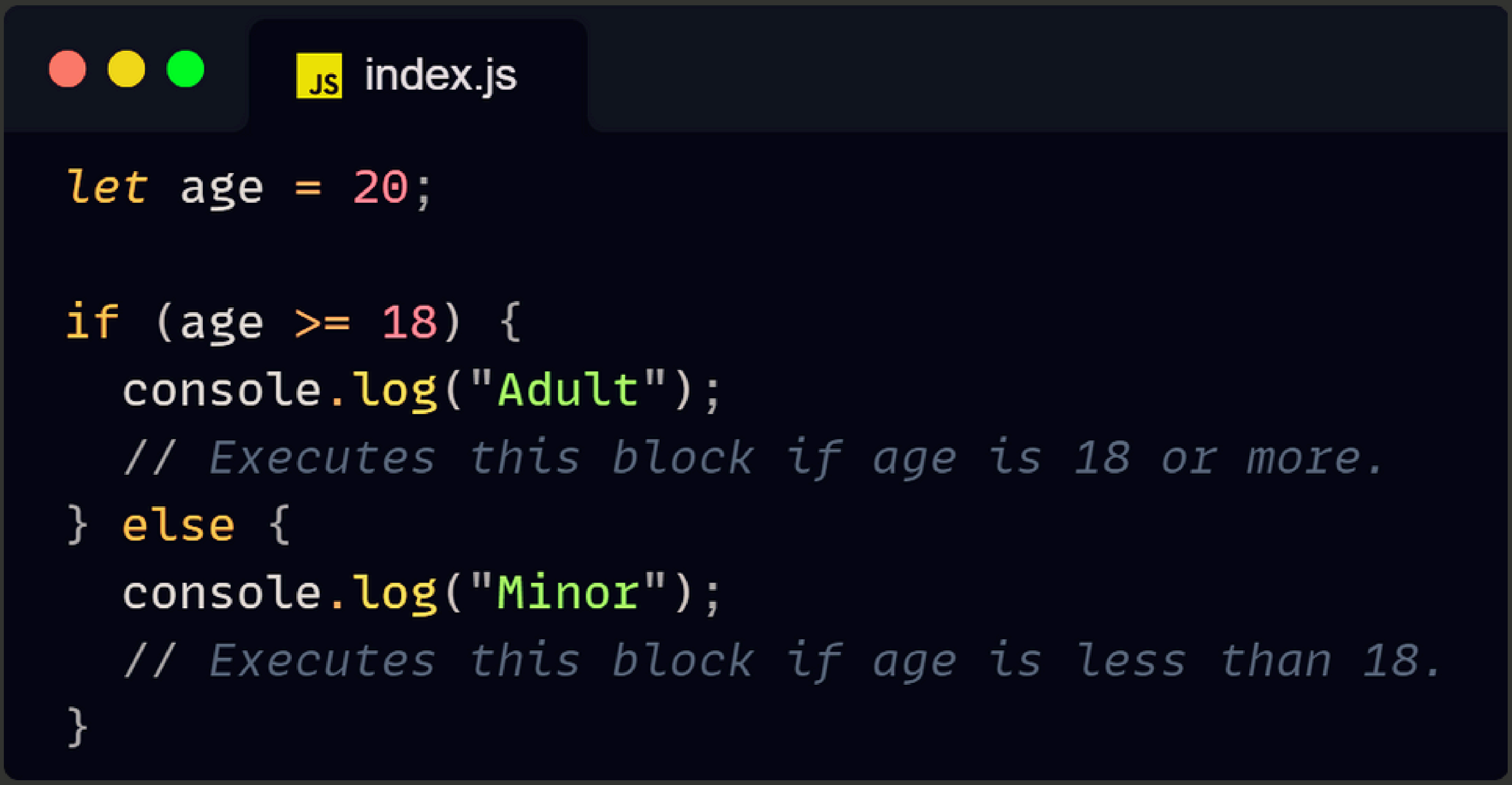
```
let isTrue = true && false; //false
let isOr = true || false;  //true
let isNot = !true;  //false
```

- **isTrue**: Uses the logical AND (&&) operator, which returns false because both conditions must be true.
- **isOr**: Uses the logical OR (||) operator, which returns true because at least one condition is true.
- **isNot**: Uses the logical NOT (!) operator, which negates the value, turning true into false.

Note:
The variable names are unrelated to the logical operations; they are simply labels for storing the result.

3. Control Flow

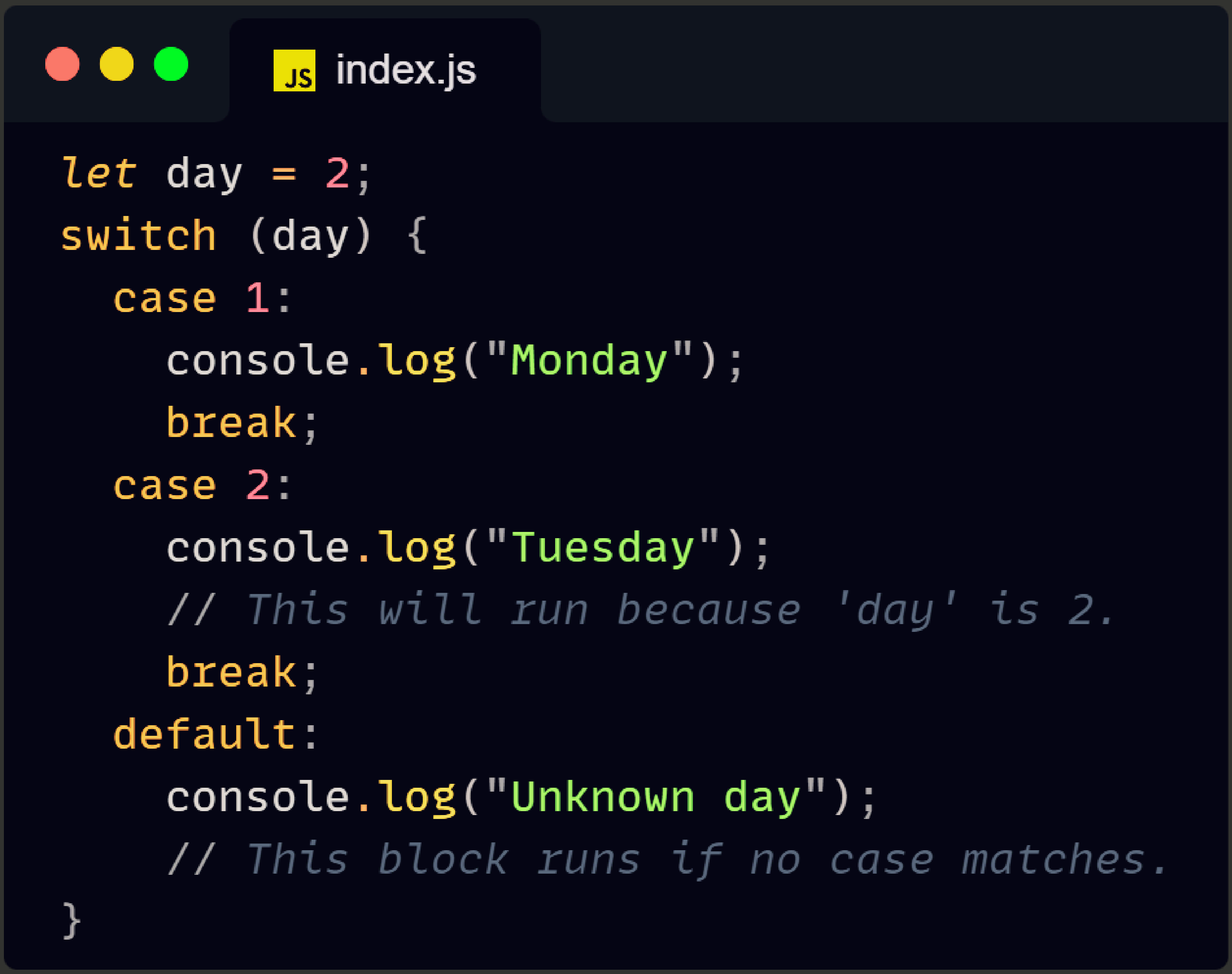
- **If-Else Statements :**



```javascript
let age = 20;

if (age >= 18) {
  console.log("Adult");
  // Executes this block if age is 18 or more.
} else {
  console.log("Minor");
  // Executes this block if age is less than 18.
}
```

Explanation:
The if...else statement checks a condition (age >= 18). If the condition is true, the first block of code runs (console.log("Adult")). If it's false, the second block runs (console.log("Minor")).

- **Switch Statements**



index.js

```
let day = 2;
switch (day) {
  case 1:
    console.log("Monday");
    break;
  case 2:
    console.log("Tuesday");
    // This will run because 'day' is 2.
    break;
  default:
    console.log("Unknown day");
    // This block runs if no case matches.
}
```

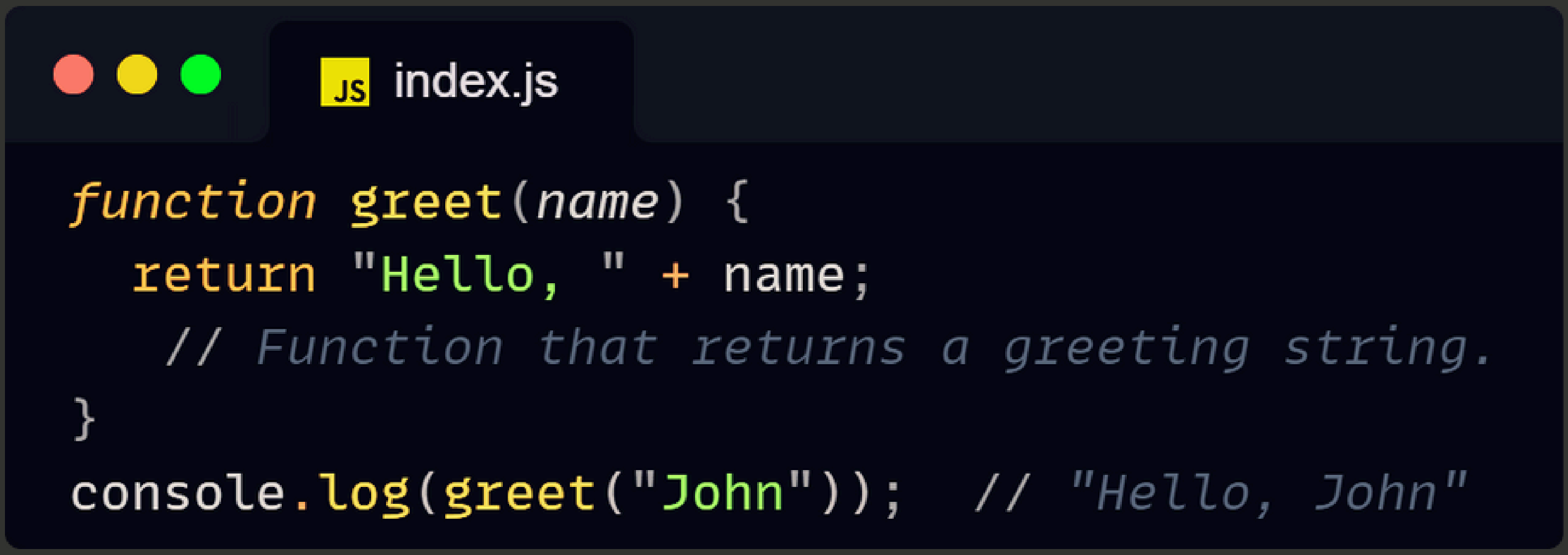
Explanation:

The switch statement checks the value of day.

- If day matches 1, it prints "Monday".
- If day matches 2, it prints "Tuesday".
- If no case matches, the default block runs, printing "Unknown day".
- The break statement prevents the execution from continuing to the next case.

4. Functions

- **Function Declaration :**

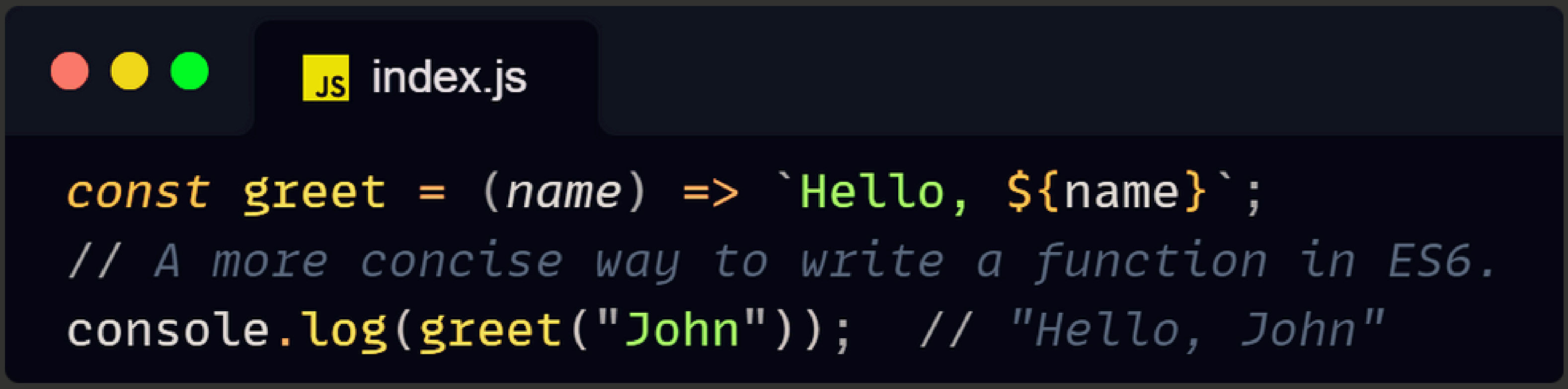


```javascript
function greet(name) {
  return "Hello, " + name;
   // Function that returns a greeting string.
}
console.log(greet("John"));  // "Hello, John"
```

Explanation:

- function greet(name): A function named greet that takes one parameter, name.
- return "Hello, " + name;: Combines the string "Hello, " with the value of name and returns the result.
- console.log(greet("John"));: Calls the greet function with "John" as an argument and logs the returned value ("Hello, John") to the console.

- **Arrow Functions**



index.js

```javascript
const greet = (name) => `Hello, ${name}`;
// A more concise way to write a function in ES6.
console.log(greet("John"));  // "Hello, John"
```

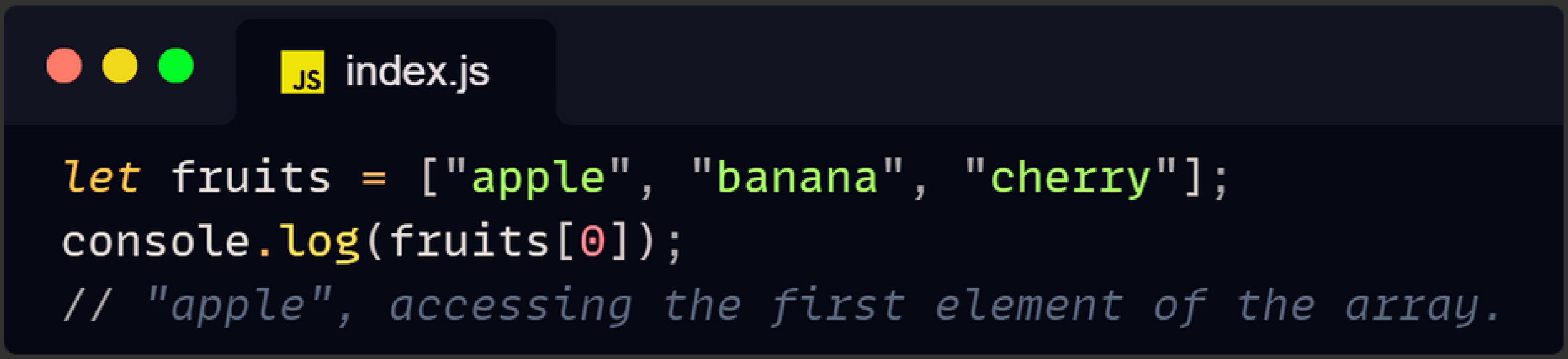
Explanation:

- const greet = (name) =>: Defines an arrow function named greet that takes one parameter, name.
- `Hello, ${name}`: Uses template literals (enclosed by backticks `) to insert the value of name into the string dynamically.
- console.log(greet("John"));: Calls the greet function with "John" as an argument and logs the returned value ("Hello, John") to the console.

Arrow functions provide a shorter syntax for defining functions in ES6.

5. Arrays

- **Creating and Accessing Elements**

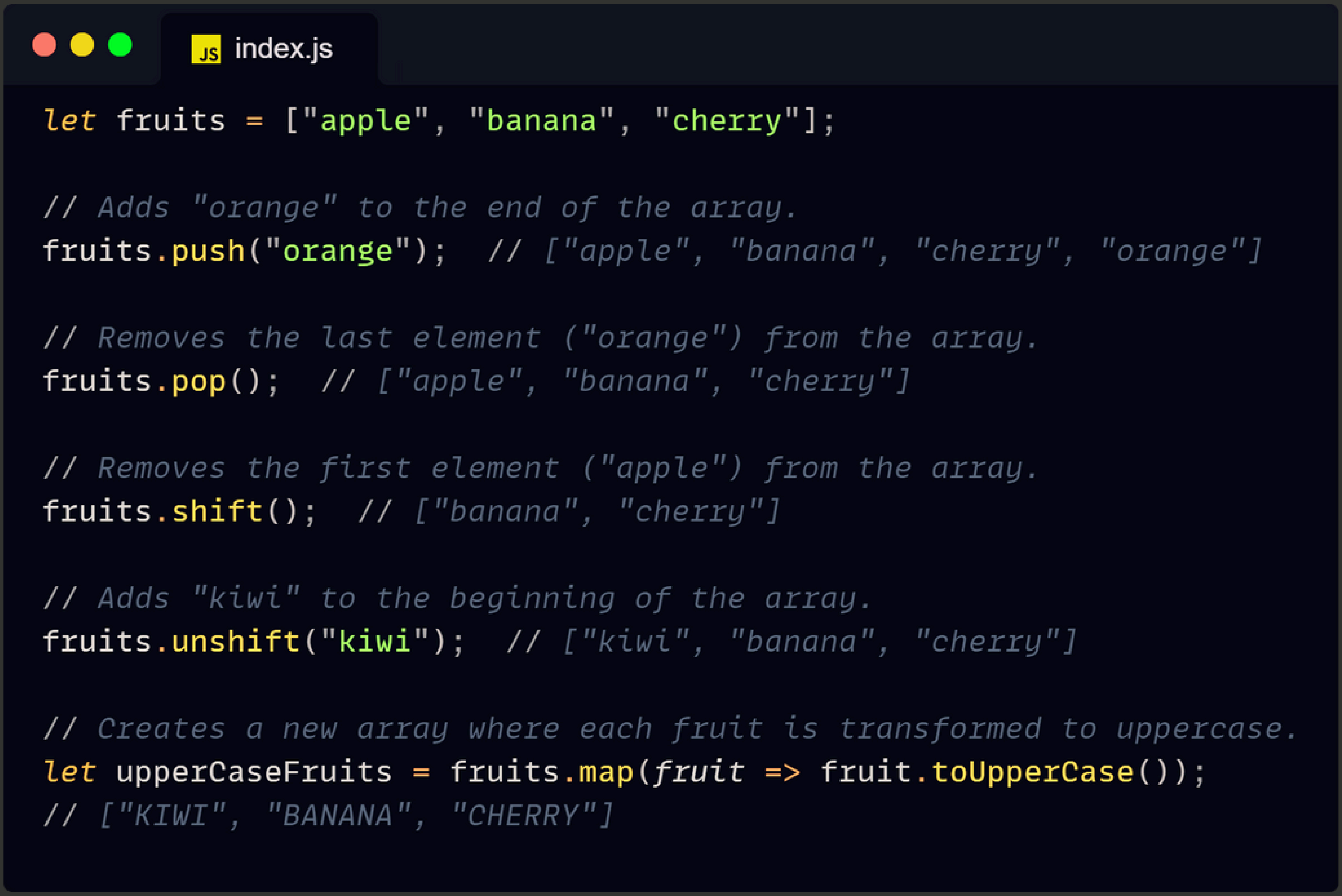

index.js

```javascript
let fruits = ["apple", "banana", "cherry"];
console.log(fruits[0]);
// "apple", accessing the first element of the array.
```

Explanation:

- let fruits = ["apple", "banana", "cherry"];: Creates an array named fruits containing three elements: "apple", "banana", and "cherry".
- fruits[0]: Accesses the first element of the array (arrays use zero-based indexing).
- console.log(fruits[0]);: Logs "apple" to the console, as it is the element at index 0.

- **Array Methods:**


index.js

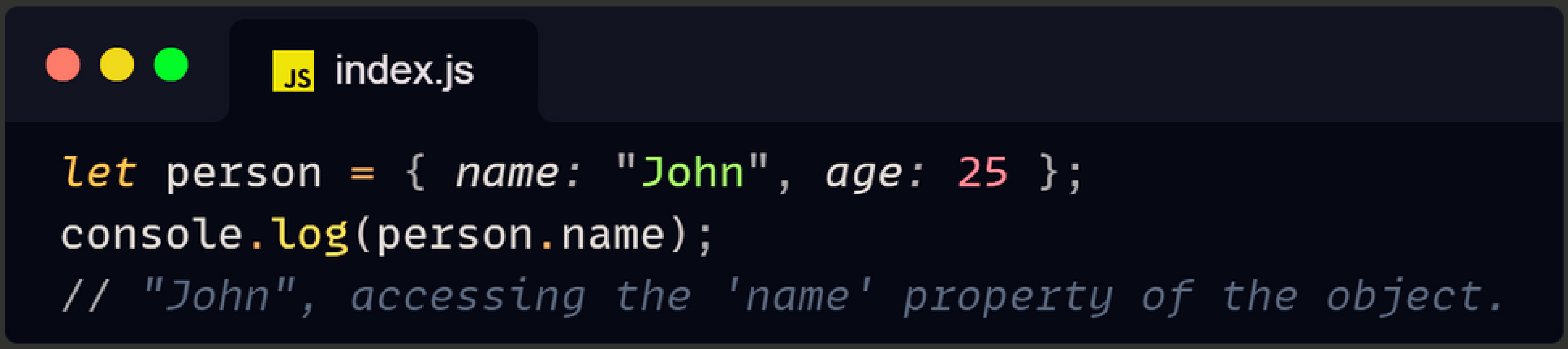
```javascript
let fruits = ["apple", "banana", "cherry"];

// Adds "orange" to the end of the array.
fruits.push("orange");  // ["apple", "banana", "cherry", "orange"]

// Removes the last element ("orange") from the array.
fruits.pop();  // ["apple", "banana", "cherry"]

// Removes the first element ("apple") from the array.
fruits.shift();  // ["banana", "cherry"]

// Adds "kiwi" to the beginning of the array.
fruits.unshift("kiwi");  // ["kiwi", "banana", "cherry"]

// Creates a new array where each fruit is transformed to uppercase.
let upperCaseFruits = fruits.map(fruit => fruit.toUpperCase());
// ["KIWI", "BANANA", "CHERRY"]
```

Explanation:

- .push() and .pop(): Work with the end of the array.
- .shift() and .unshift(): Work with the beginning of the array.
- .map(): Applies a transformation (e.g., converting to uppercase) to each element and returns a new array. The original array remains unchanged.

6. Objects

- **Creating Objects and Accessing Properties**


index.js

```
let person = { name: "John", age: 25 };
console.log(person.name);
// "John", accessing the 'name' property of the object.
```

Explanation:

- let person = { name: "John", age: 25 };: Defines an object person with two properties:
  - name: Stores the string "John".
  - age: Stores the number 25.
- person.name: Accesses the value of the name property in the object.
- console.log(person.name);: Logs the value of person.name ("John") to the console.

Objects in JavaScript are collections of key-value pairs where properties (keys) are used to store and access values.

- **Adding/Updating Object Properties**

index.js

```
person.gender = "male";
// Adds a new property 'gender' with value 'male' to the object.
person.age = 26;
// Updates the existing 'age' property to 26.
```

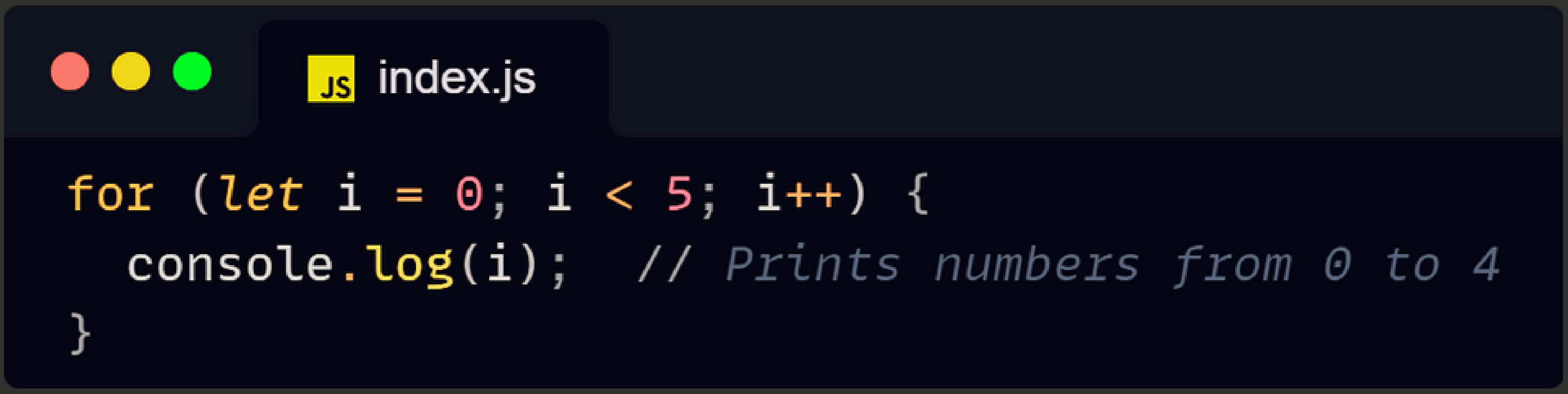
Explanation:

- person.gender = "male";: Adds a new property gender to the person object with the value "male".
- person.age = 26;: Updates the existing age property in the person object to 26.

In JavaScript, you can add new properties or update existing ones in an object by directly assigning values to them.

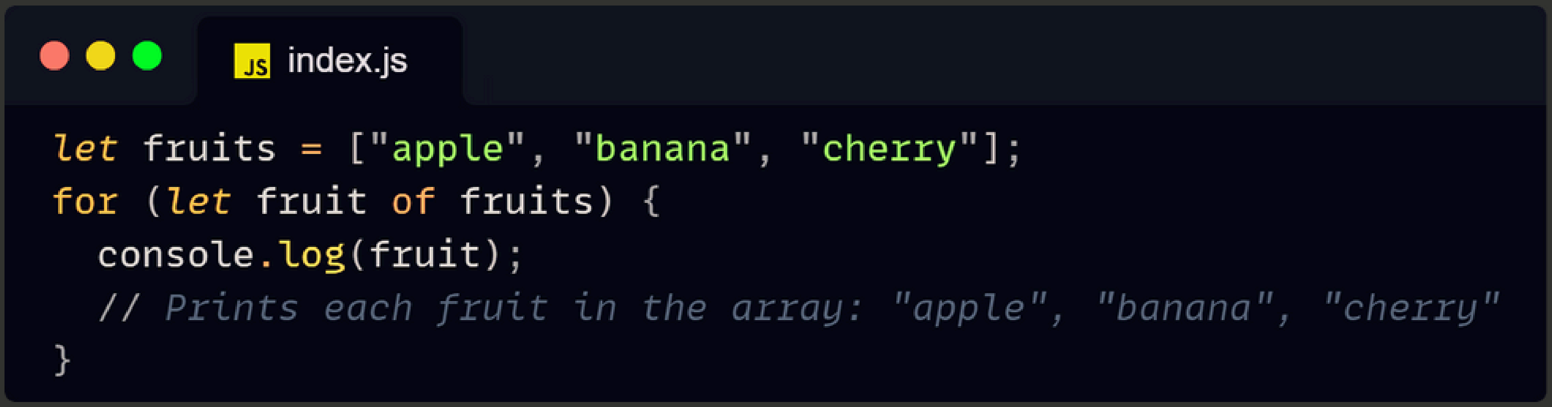
7. Loops

- **For Loop**



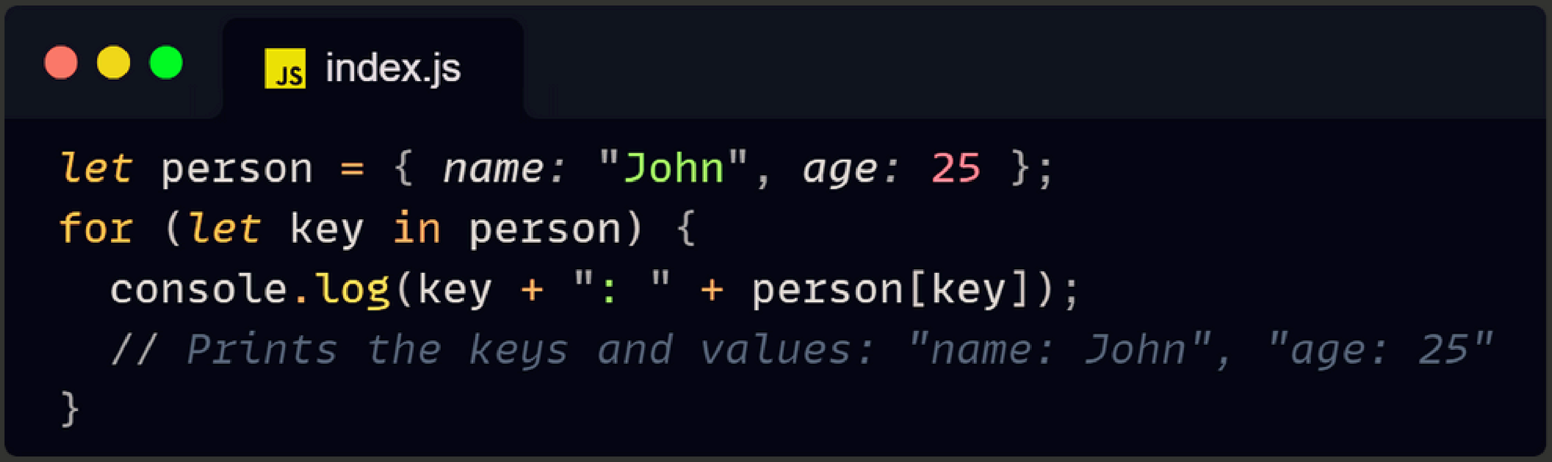
```js
for (let i = 0; i < 5; i++) {
  console.log(i);  // Prints numbers from 0 to 4
}
```

- **For...of Loop (for Arrays)**



```js
let fruits = ["apple", "banana", "cherry"];
for (let fruit of fruits) {
  console.log(fruit);
  // Prints each fruit in the array: "apple", "banana", "cherry"
}
```

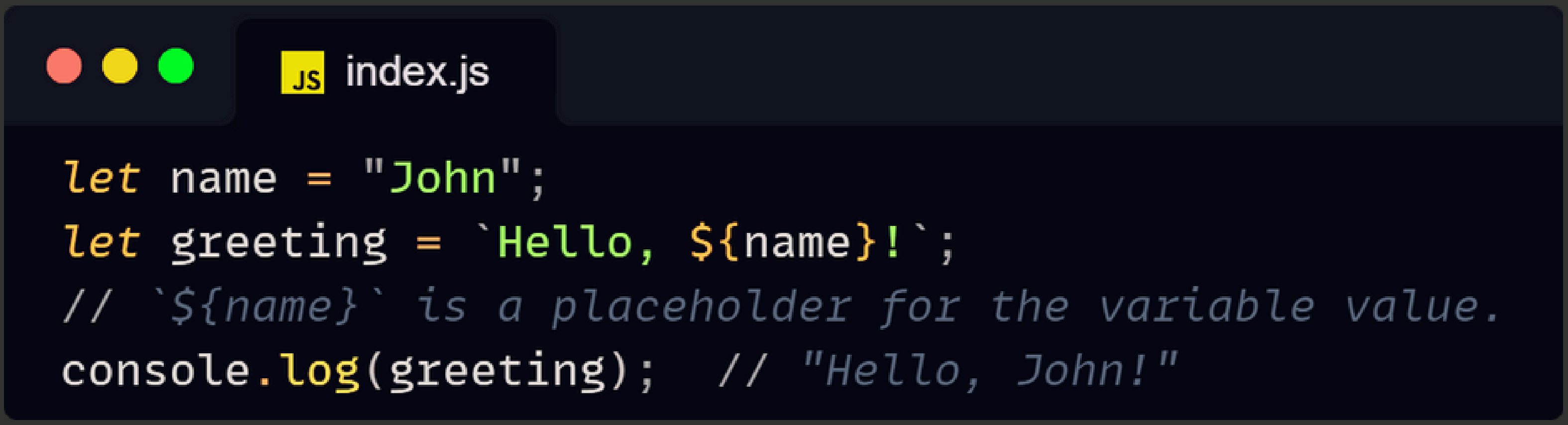
- **For...in Loop (for Objects)**



```js
let person = { name: "John", age: 25 };
for (let key in person) {
  console.log(key + ": " + person[key]);
  // Prints the keys and values: "name: John", "age: 25"
}
```

8. ES6 Features

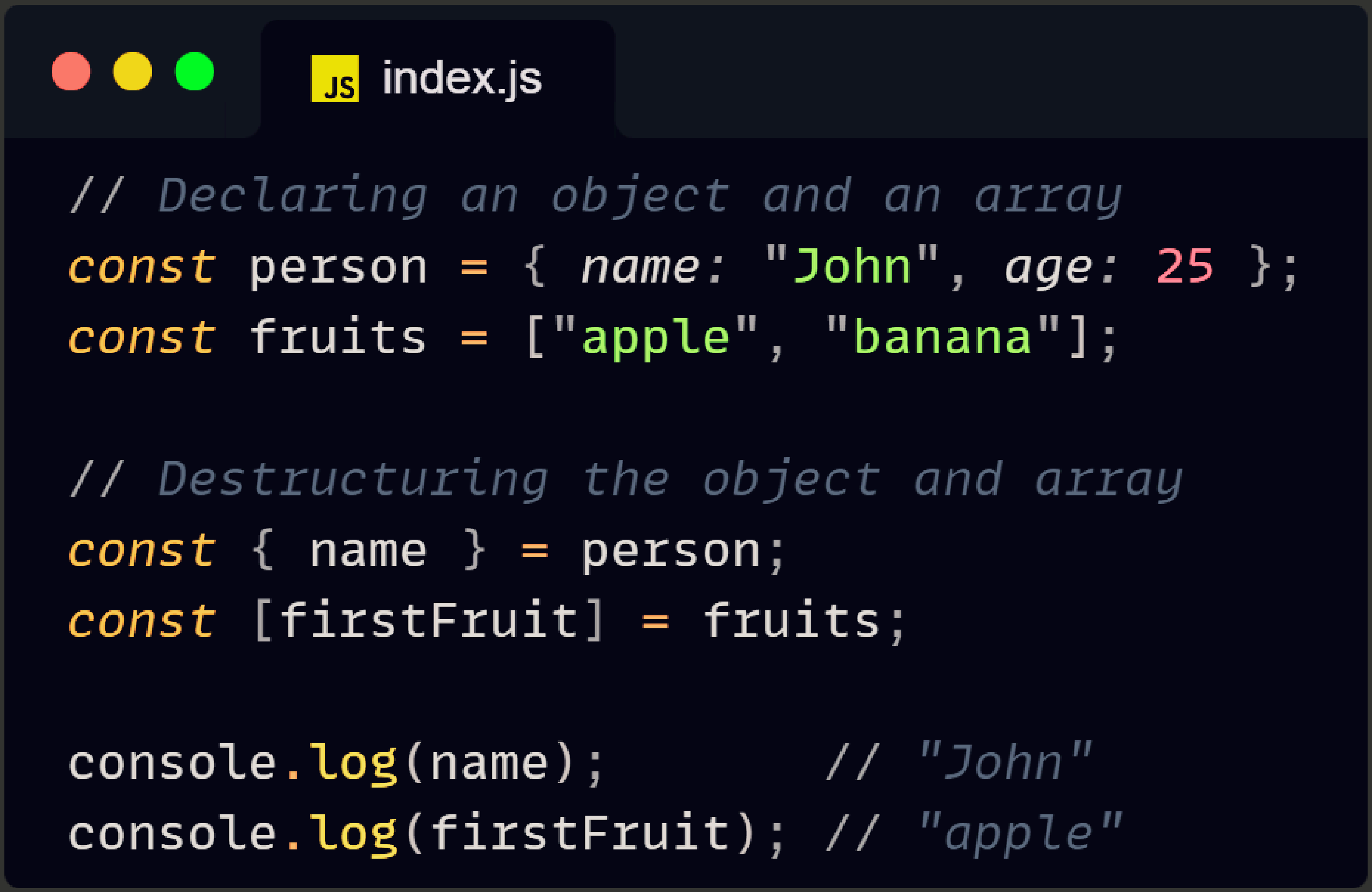
- **Template Literals**



index.js

```javascript
let name = "John";
let greeting = `Hello, ${name}!`;
// `${name}` is a placeholder for the variable value.
console.log(greeting);  // "Hello, John!"
```

- **Destructuring**

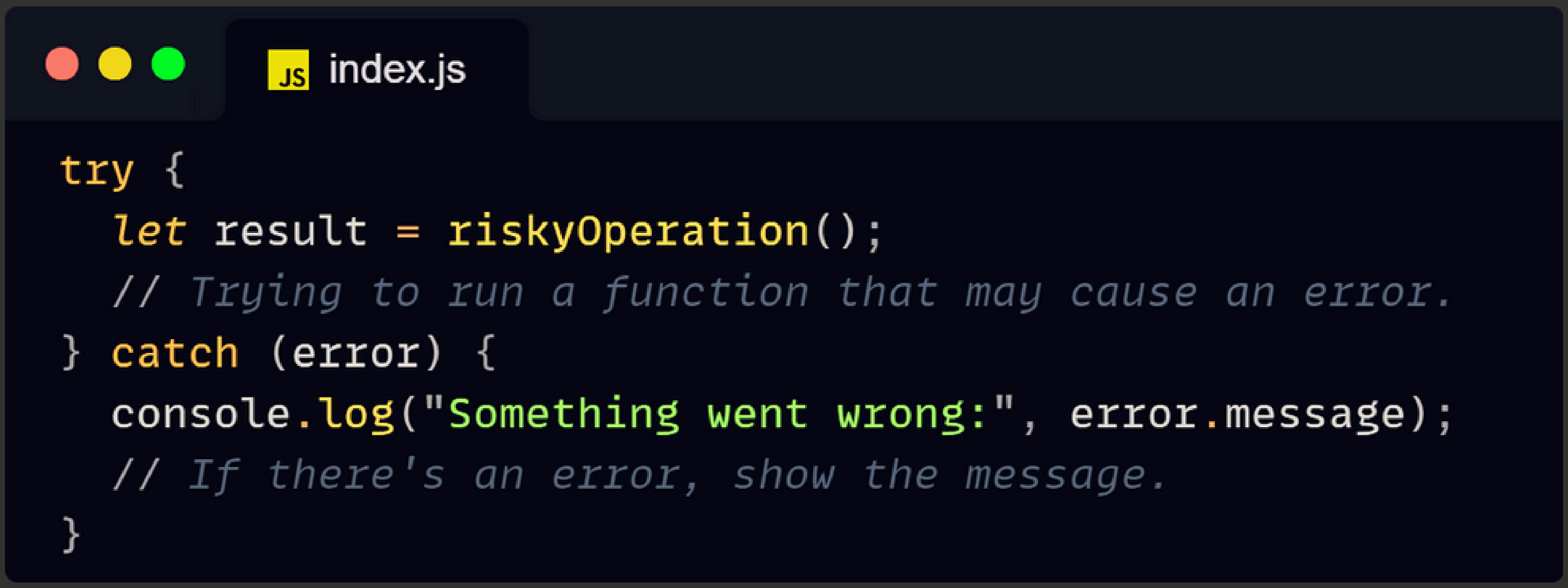


index.js

```javascript
// Declaring an object and an array
const person = { name: "John", age: 25 };
const fruits = ["apple", "banana"];

// Destructuring the object and array
const { name } = person;
const [firstFruit] = fruits;

console.log(name);       // "John"
console.log(firstFruit); // "apple"
```

9. Error Handling

- **Try...Catch**



index.js

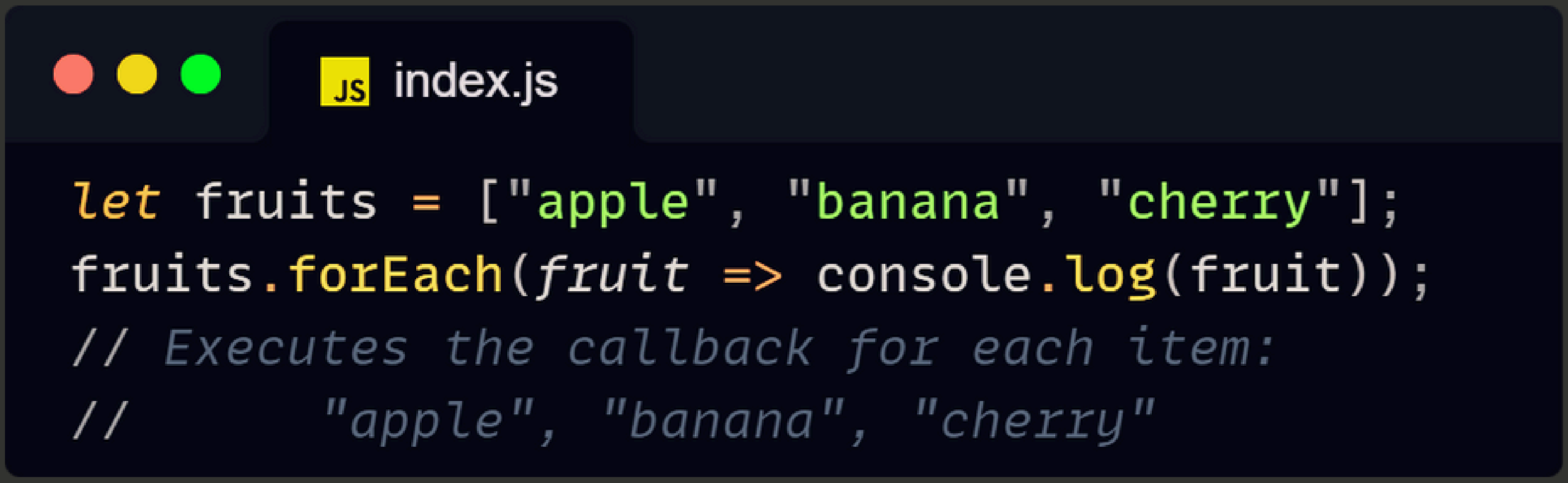
```
try {
  let result = riskyOperation();
  // Trying to run a function that may cause an error.
} catch (error) {
  console.log("Something went wrong:", error.message);
  // If there's an error, show the message.
}
```

Explanation:

- **try block**: This part tries to run the code inside it. Here, it tries to run a function called riskyOperation(). This function may cause an error.
- **catch block**: If an error happens inside the try block, the code jumps to the catch block. The catch block takes the error and logs a message to the console, explaining that something went wrong.
- **Why use this?**: The try...catch helps you prevent the program from crashing by safely handling errors and showing meaningful messages.

10. Common Array Methods

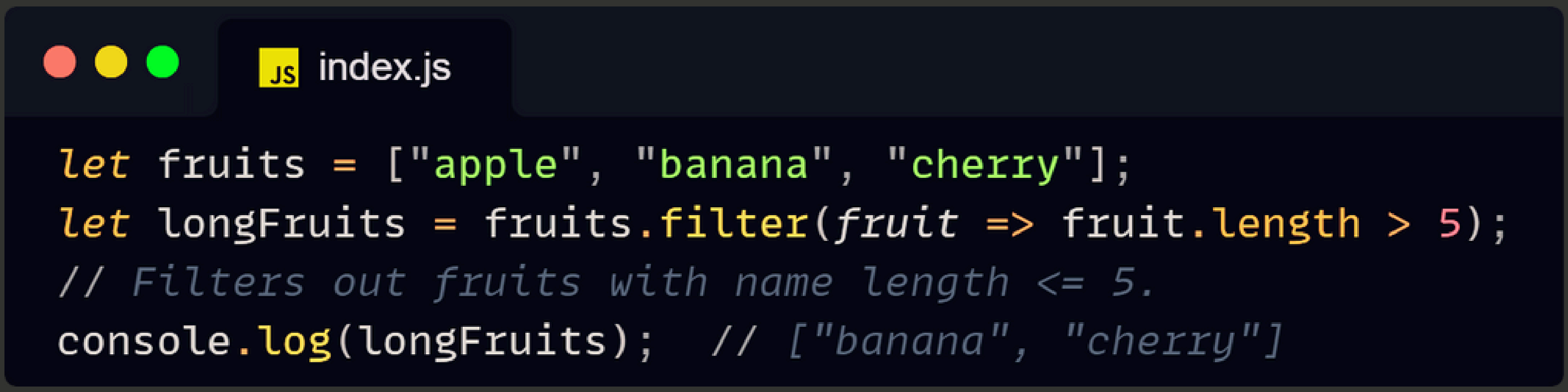
- .forEach()



index.js

```javascript
let fruits = ["apple", "banana", "cherry"];
fruits.forEach(fruit => console.log(fruit));
// Executes the callback for each item:
//       "apple", "banana", "cherry"
```

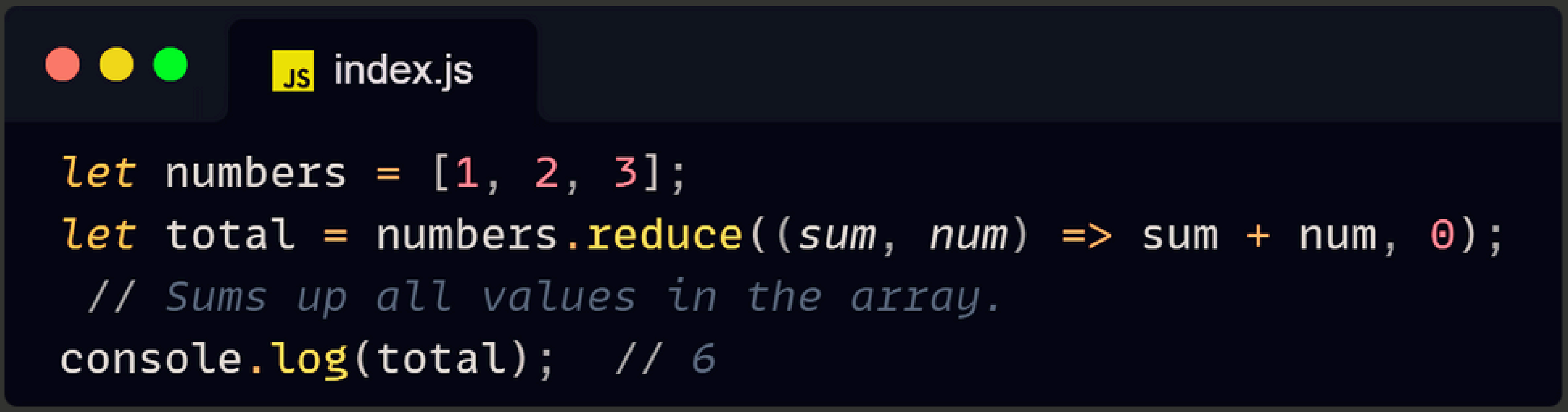
- .filter()



index.js

```javascript
let fruits = ["apple", "banana", "cherry"];
let longFruits = fruits.filter(fruit => fruit.length > 5);
// Filters out fruits with name length <= 5.
console.log(longFruits);  // ["banana", "cherry"]
```

- .reduce()



index.js

```javascript
let numbers = [1, 2, 3];
let total = numbers.reduce((sum, num) => sum + num, 0);
 // Sums up all values in the array.
console.log(total);  // 6
```

Hopefully You Found It Usefull!

“Be sure to save this post so you can come back to it later ”

like Comment Share