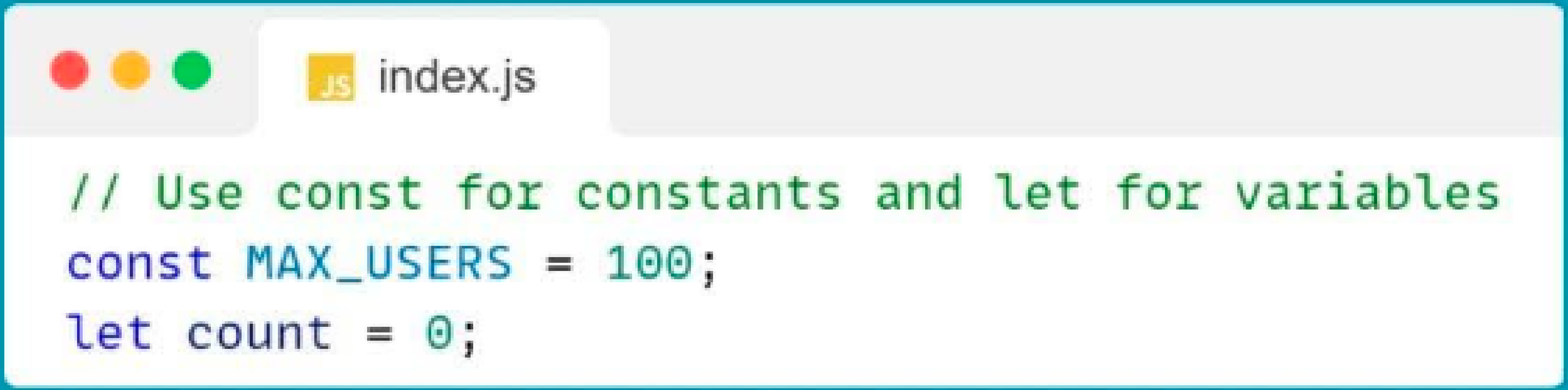
ENHANCE YOUR JAVASCRIPT CODE SPEED

1. Use let and const Instead of var

- Why?
  - var has function scope, leading to potential memory leaks.
  - let and const provide block scope, which helps reduce bugs.

Example:



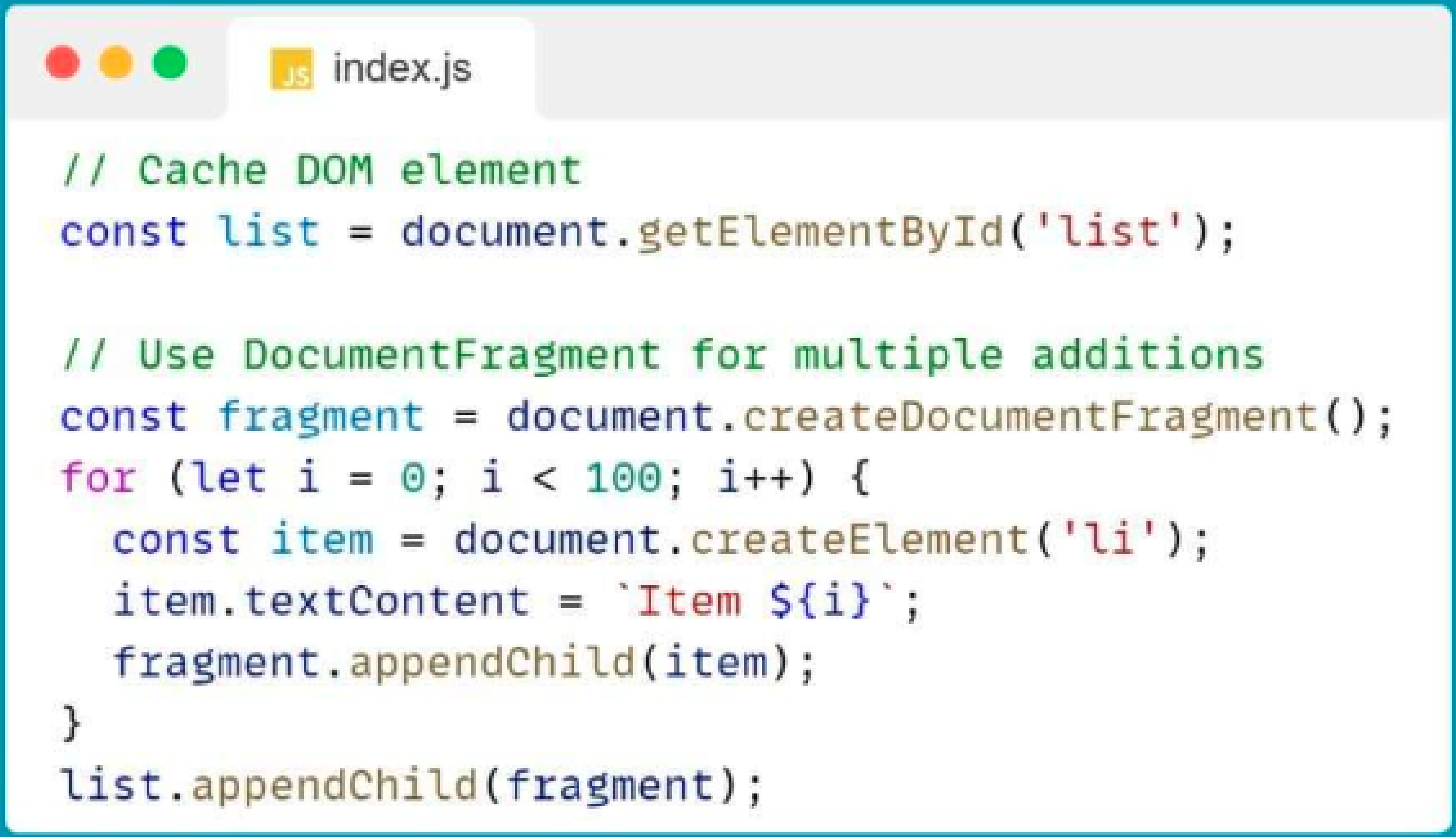
```js
// Use const for constants and let for variables
const MAX_USERS = 100;
let count = 0;
```

2. Minimize DOM Manipulations

- Why?
    - DOM access is slow, so minimize changes and reflows..

- Tips:
    - Use DocumentFragment for batch operations.
    - Cache DOM queries.

Example:



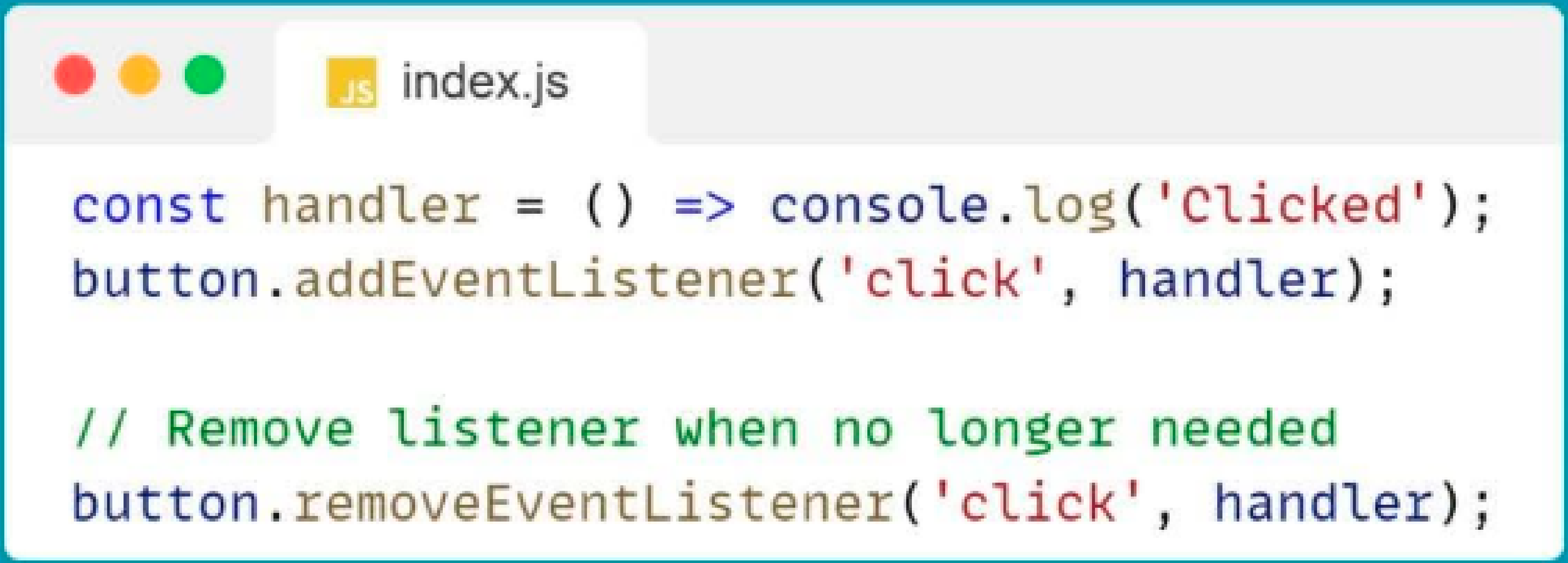
```js
// Cache DOM element
const list = document.getElementById('list');

// Use DocumentFragment for multiple additions
const fragment = document.createDocumentFragment();
for (let i = 0; i < 100; i++) {
  const item = document.createElement('li');
  item.textContent = `Item ${i}`;
  fragment.appendChild(item);
}
list.appendChild(fragment);
```

3. Avoid Memory Leaks

- Why?
  - Unused variables or event listeners can cause memory issues.
- Tips:
  - Use WeakMap or WeakSet for objects you want garbage collected.
  - Remove event listeners when they are no longer needed.

Example:



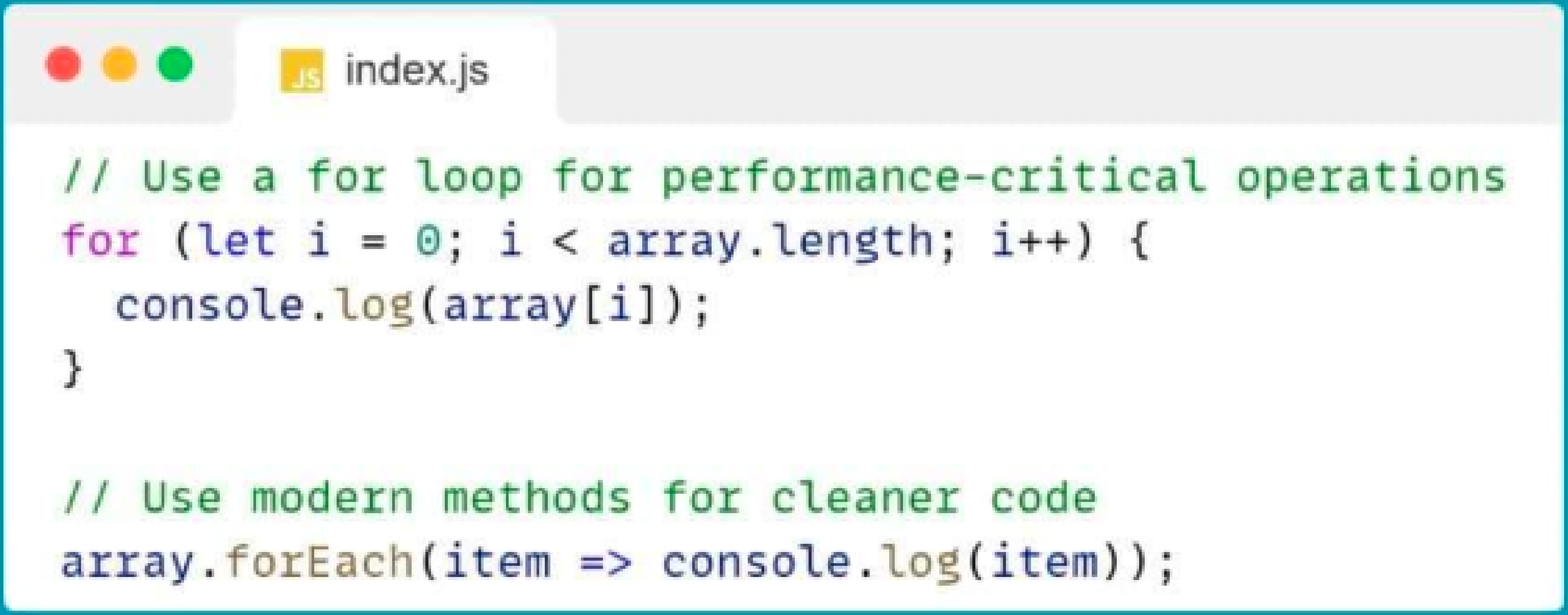
```javascript
const handler = () => console.log('Clicked');
button.addEventListener('click', handler);

// Remove listener when no longer needed
button.removeEventListener('click', handler);
```

4. Optimize Loops

- Why?
  - Loops can be computationally expensive. Use the most efficient type.
- Tips:
  - For loop: Most efficient for arrays.
  - Array methods: Use forEach, map, or reduce for cleaner code.

Example:



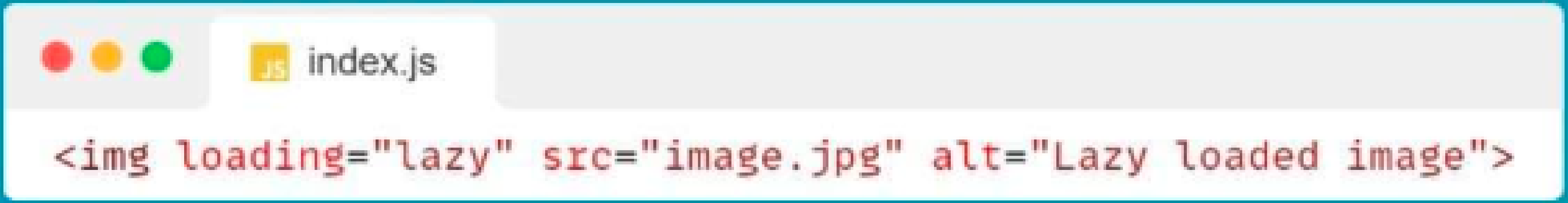
```javascript
// Use a for loop for performance-critical operations
for (let i = 0; i < array.length; i++) {
  console.log(array[i]);
}

// Use modern methods for cleaner code
array.forEach(item => console.log(item));
```

6. Lazy Load Images and Components

- Why?
  - Improves initial load time.

Example:



```html
<img loading="lazy" src="image.jpg" alt="Lazy loaded image">
```

7. Minify and Bundle Your Code

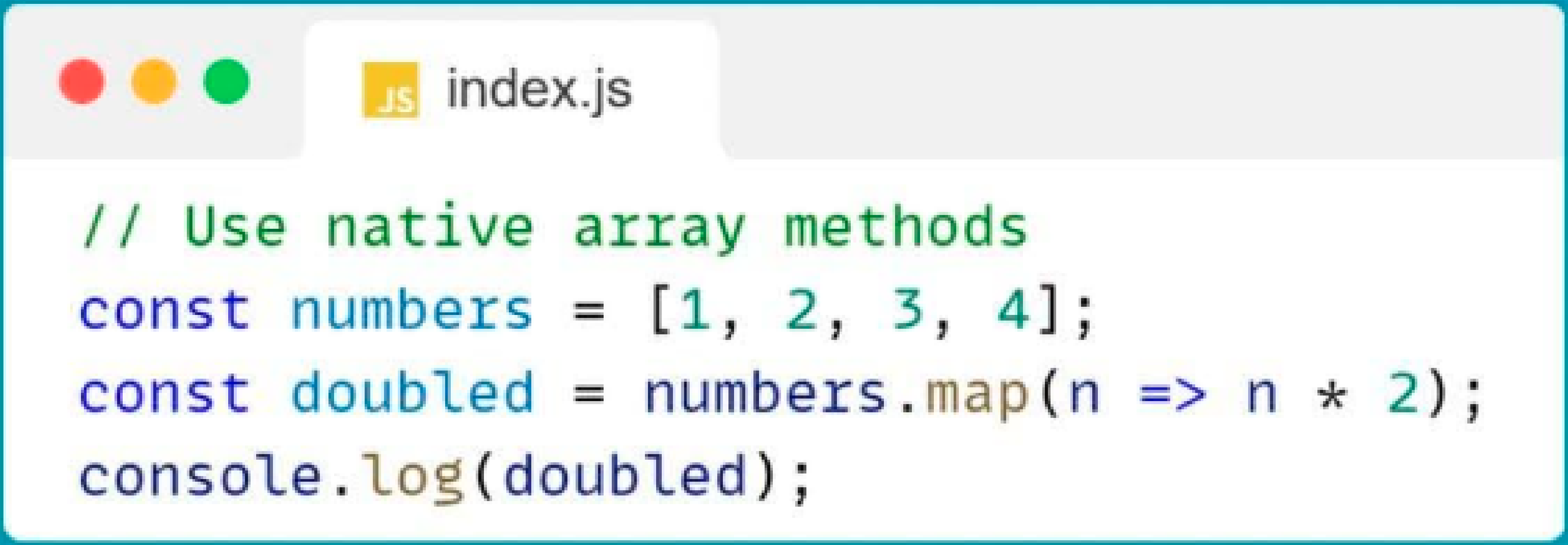
- Why?
  - Reduces file size and improves load time.

- **Tools:**
  - Webpack
  - Parcel
  - Rollup

8. Prefer Native Methods

- Why?
    - Native methods are faster than custom implementations.

Example:



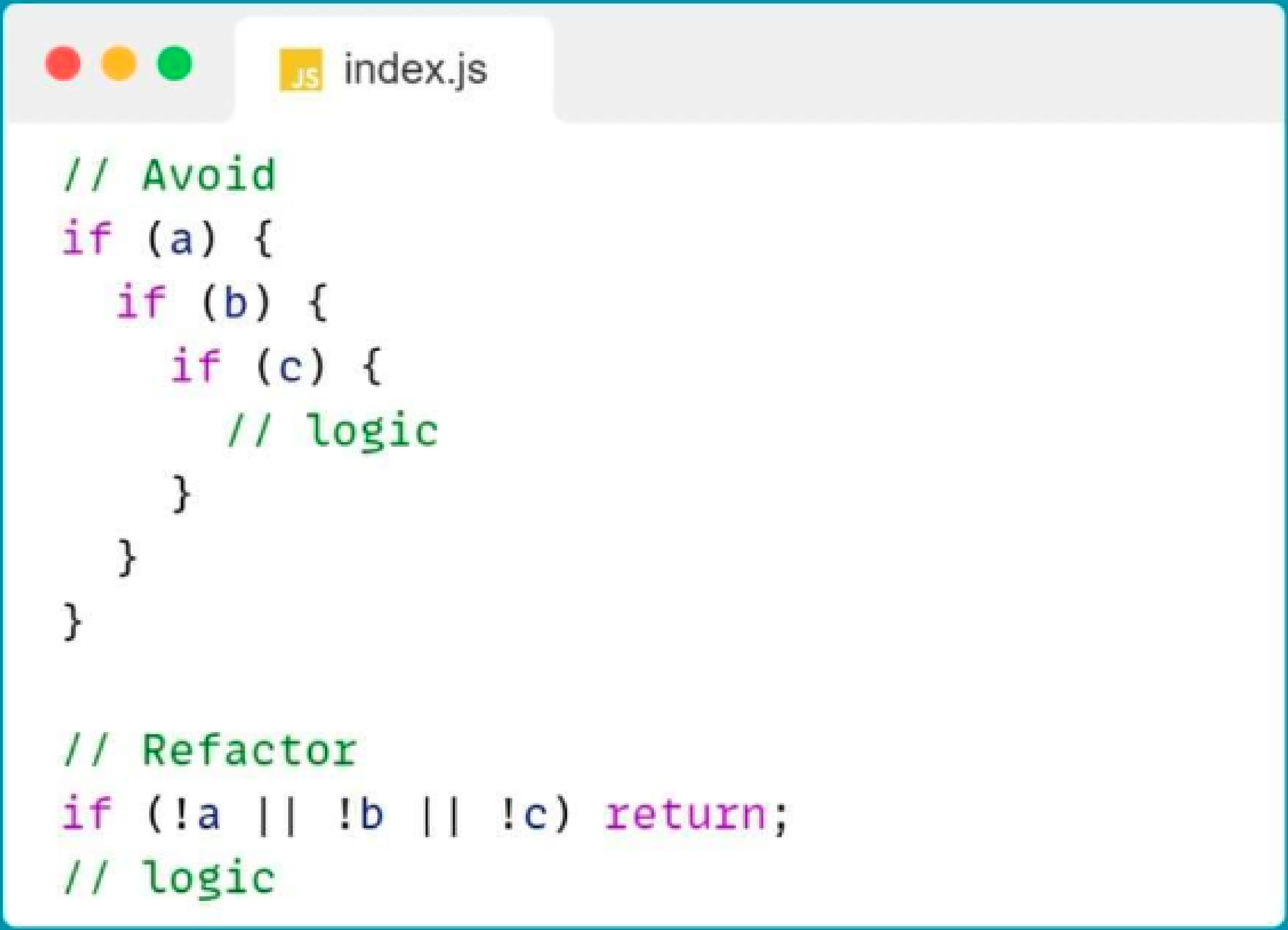
```javascript
// Use native array methods
const numbers = [1, 2, 3, 4];
const doubled = numbers.map(n => n * 2);
console.log(doubled);
```

9. Avoid Deep Nesting

- Why?
  - Improves readability and performance.

Example:

Refactor code into smaller functions.



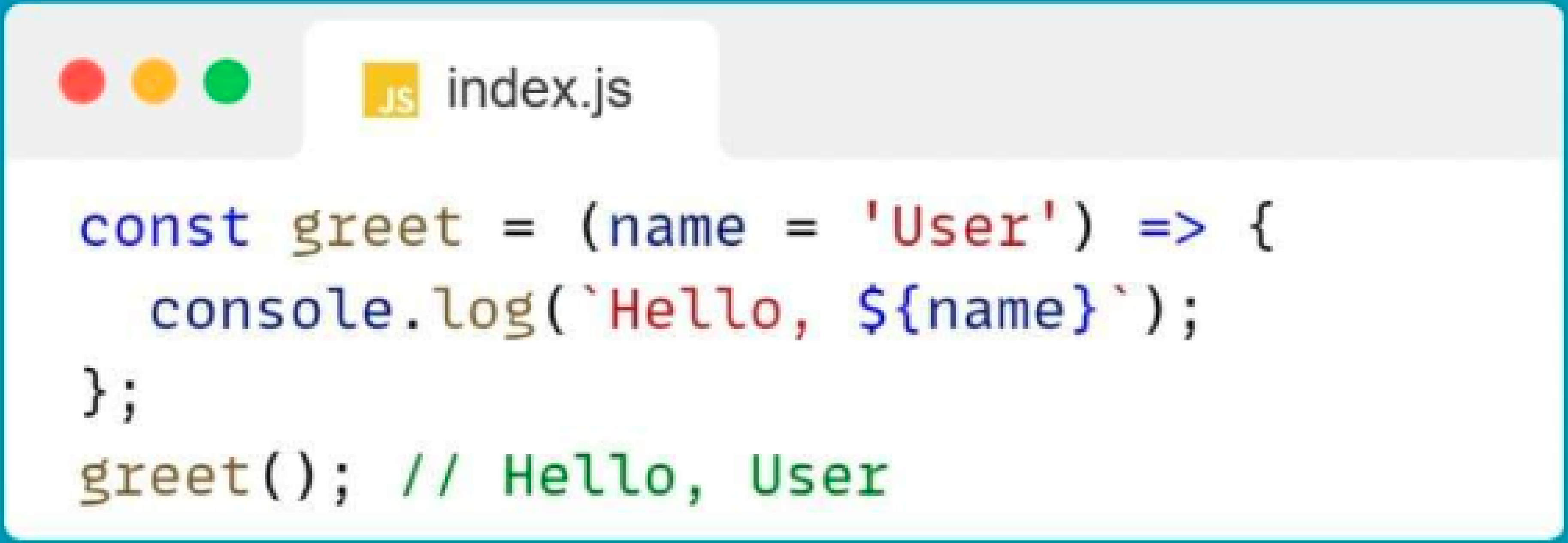
```javascript
// Avoid
if (a) {
  if (b) {
    if (c) {
      // logic
    }
  }
}

// Refactor
if (!a || !b || !c) return;
// logic
```

10. Use Default Parameters

- Why?
    - Simplifies handling of optional parameters.

Example:



```javascript
const greet = (name = 'User') => {
  console.log(`Hello, ${name}`);
};
greet(); // Hello, User
```