Avoid these 5 JavaScript Mistakes

Chetan Mahajan
@_chetanmahajan

1. Inefficient Use of Arrays and Objects

❌ *Avoid this*

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

for (let i = 0; i < myArray.length; i++) {
  console.log(myArray[i]);
}
```

✅ *Do this*

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

myArray.forEach(item => {
  console.log(item);
});
```


Use **built-in array methods** like forEach(), map(), filter(), and reduce() to perform common array operations more efficiently.

2. Improper Use of Closures

⊗ ***Avoid this***

```
function createCounter() {
  let count = 0;
  return function() {
    return count++;
  };
}

const counter1 = createCounter();
console.log(counter1()); // Output: 0
console.log(counter1()); // Output: 1

const counter2 = createCounter();
console.log(counter2()); // Output: 0
console.log(counter2()); // Output: 1
```

Chetan Mahajan
@_chetanmahajan

✅ Do this

```
function createCounter() {
  let count = 0;
  return {
    increment: () => ++count,
    decrement: () => --count,
    getCount: () => count
  };
}

const counter = createCounter();
console.log(counter.getCount()); // Output: 0
console.log(counter.increment()); // Output: 1
console.log(counter.decrement()); // Output: 0
```


Properly use closures to create private variables and encapsulate state within functions.

Chetan Mahajan
@_chetanmahajan

3. Inefficient Use of Arrays and Objects

⊗ *Avoid this*

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

for (let i = 0; i < myArray.length; i++) {
  console.log(myArray[i]);
}
```

✓ *Do this*

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

myArray.forEach(item => {
  console.log(item);
});
```

Use **built-in array methods** like forEach(), map(), filter(), and reduce() to perform common array operations more efficiently.

4. Misusing forEach for asynchronous operations

⊗ ***Avoid this***

```
async function processItems(items) {
    items.forEach(async (item) => {
        await processItem(item);
    });
    console.log('All items processed');
  // This logs before processing is complete
}
```

✅ *Do this*

```
async function processItems(items) {
    await Promise.all(items.map(async (item) => {
        await processItem(item);
    }));
    console.log('All items processed');
    // This logs after all items are processed
}
```

Use **Promise.all with map** for parallel asynchronous operations, or a for...of loop for sequential processing.

Chetan Mahajan
@_chetanmahajan

5. Not handling asynchronous operations properly

⊗ ***Avoid this***

```
function getData() {
    let data;
    fetch('https://api.example.com/data')
        .then(response => response.json())
        .then(result => {
            data = result;
        });
    return data; // Will always be undefined
}
```

Chetan Mahajan
@_chetanmahajan

✅ Do this

www.developerupdates.com

```
async function getData() {
    try {
        const response = await fetch('https://api.example.com/data');
        const data = await response.json();
        return data;
    } catch (error) {
        console.error('Error fetching data:', error);
    }
}
```

Use **async/await or properly chain promises** to handle asynchronous operations.

Chetan Mahajan
@_chetanmahajan

Are you looking for Front-end Developer Job?

If yes, Check the link in bio to get Interview Kit and start preparing Today.