8 Common JavaScript Mistakes

Save for later

#1

Misusing Double Equals

“ == “ allows type coercion, leading to unexpected results & errors. Instead use “ === “ for strict equality checks.

```
// Problem: (type coercion happens)
console.log('5' == 5); // true
console.log(false == 0); // true

// Solution:
console.log('5' === 5); // false
console.log(false === 0); // false
```

#2

Forgetting ‘let’ and ‘const’

Using ‘var’ can cause scope and hoisting issues.
Use ‘**let**’ for mutables and ‘**const**’ for constants.

```
// Problem with `var`:
if (true) { var x = 10 }
console.log(x);
// 10 (leaks outside block scope)

// Solution:
if (true) { let y = 10 }
console.log(y);
// ReferenceError: y is not defined
```

#3

Ignoring ‘undefined’ vs ’null’

Confusing **undefined** (absence of value) with **null** (explicitly nothing). Use ‘null’ intentionally and let JavaScript handle ‘undefined’.

```
// Problem:
let a; // undefined (not initialized)
let b = null; // null (explicitly set to nothing)

// Solution:
function fetchData() {
    return null; // Intentionally no data
}
```

#4

Ignoring Asynchronous Behavior

Promises are asynchronous, not synchronous.
Use **async/await**.

```
async function fetchData() {
    let data = await fetch('https://api.example.com');
    console.log('Data fetched');
}
```

#5

Hardcoding Magic Numbers

Hardcoded numbers reduce readability.
Use descriptive constants.

```
const SECONDS_IN_A_DAY = 86400;
```

#6

Writing Functions Without Defaults

Functions fail without parameters.

Use default parameters.

```
function greet(name = 'Guest') {
    return `Hello, ${name}!`;
}
```

#7

Not Handling Errors Gracefully

Use 'try' and 'catch' blocks to handle errors.

```
try {
    riskyOperation();
} catch (error) {
    console.error('Error occurred:', error);
}
```

Which mistake have you struggled with?

Let me know in the comments! 💬

Follow me
for more
tips!

DevHanza