

# Mistakes To Avoid As A Javascript Developer



JS

# 01.

## Not Using Strict Mode

**Strict mode** is a feature in JavaScript that enforces stricter rules for certain behaviours. It can help you avoid common mistakes and ensure that your code is more secure and performant.

For example, using **strict mode** can prevent you from accidentally creating global variables.



```
1 "use strict";  
2 x = 10;  
3 // This will cause an error because x is not declared
```

# Not Using Let And Const

Using **let** and **const** instead of **var** can help you avoid variable hoisting and prevent accidental re-declarations.

For example :

 JS script.js

```
1 let x = 1;
2 let x = 2;
3 // This will cause an error because x is already declared
4
5 const y = {};
6 y.name = "Tom"; // This is allowed
7 y = {}; // This will cause an error because y is a constant
```

03.

## Not Using Arrow Functions

Arrow functions are a shorthand notation for writing function expressions in JavaScript.

They are more concise and easier to read than traditional function expressions and can help improve the performance of your code.



JS script.js

```
1 // Traditional function expression
2 let add = function(a, b) {
3   return a + b;
4 };
5
6 // Arrow function
7 let add = (a, b) => a + b;
```



04.

## Not Using Spread Operator

The **spread operator** allows you to expand an array or object into its individual elements.

It can help you write more concise and performant code by avoiding unnecessary loops and operations.



JS script.js

```
1 let numbers = [1, 2, 3];  
2 let newNumbers = [0, ...numbers, 4];  
3 console.log(newNumbers); // [0, 1, 2, 3, 4]
```

05.

## Not Using Destructuring Assignment

Destructuring assignment allows you to extract values from arrays and objects in a more concise and readable way. It can help you write more efficient and maintainable code.



JS script.js

```
1 let person = {  
2   name: "Tom",  
3   age: 18  
4 };  
5 let { name, age } = person;  
6 console.log(name); // "Tom"  
7 console.log(age); // 18
```

06.

## Not Using Promises

**Promises** are a more modern way of handling asynchronous operations in JavaScript. They can help you write more readable and maintainable code by avoiding callback hell and chaining asynchronous operations.

JS script.js

```
1 // Callback
2 fs.readFile("file.txt", function(err, data) {
3   if (err) {
4     console.log(err);
5   } else {
6     console.log(data);
7   }
8 });
```

Instead of  
callbacks  
use promises

JS script.js

```
1 // Promise
2 fs.promises.readFile("file.txt").then(data => {
3   console.log(data);
4 }).catch(err => {
5   console.log(err);
6 });
```



07.

## Not Using === Instead Of ==

Strict equality(===) compares values without type coercion, whereas Loose equality(==) compares values after type coercion.

Using strict equality can help you avoid unexpected results and improve the performance of your code.



JS script.js

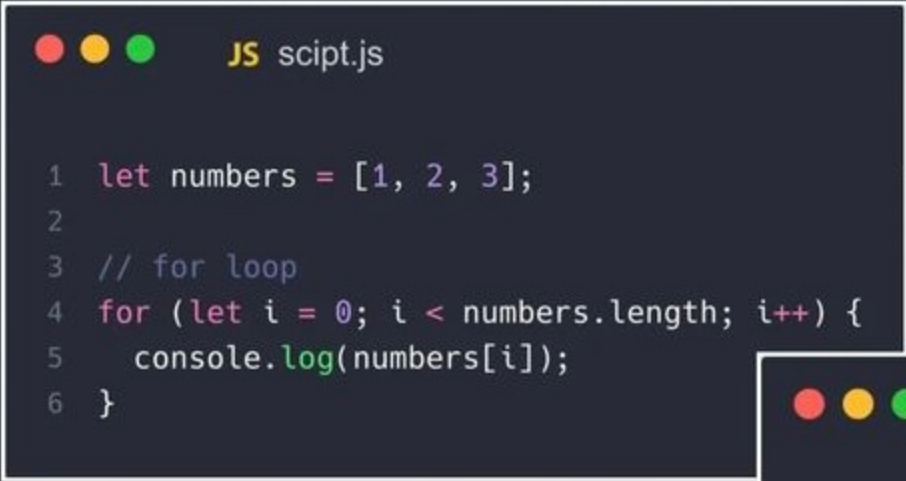
```
1 console.log(1 == "1"); // true
2 console.log(1 === "1"); // false
```



08.

## Not Using ForEach

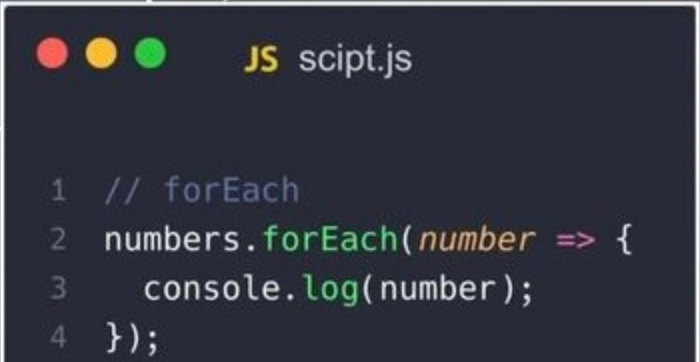
**forEach** is a higher-order function that can be used to iterate over an array and perform a specific operation on each element. It can help you write more concise and readable code by avoiding for loops and reducing the amount of boilerplate code.



```
JS script.js

1 let numbers = [1, 2, 3];
2
3 // for loop
4 for (let i = 0; i < numbers.length; i++) {
5   console.log(numbers[i]);
6 }
```

A code editor window titled "JS script.js" with a dark background. It contains a for loop that iterates over an array of numbers [1, 2, 3] and logs each element to the console. The code is written in a light-colored font with syntax highlighting. A white arrow points from the for loop code to the forEach code in the adjacent window.



```
JS script.js

1 // forEach
2 numbers.forEach(number => {
3   console.log(number);
4 });
```

A code editor window titled "JS script.js" with a dark background. It contains a forEach loop that iterates over the same array of numbers [1, 2, 3] and logs each element to the console. The code is written in a light-colored font with syntax highlighting. This window is connected to the previous one by a white arrow.