Top JavaScript Tricks For Cleaner Code







01. Use Object Destructuring

Object destructuring allows you to extract data from objects into distinct variables. This can make your code cleaner by avoiding repetitively using dot notation.

```
1 const user = {
2    name: 'John',
3    age: 24
4 };
5
6 // Without destructuring
7 const name = user.name;
8 const age = user.age;
9
10 // With destructuring
11 const { name, age } = user;
```



02. Use Default Parameters

Default parameters allow you to set default values for function parameters if none are provided. This avoids repeating yourself by redefining values every time.

```
1 function greet(name = 'Max') {
2  console.log('Hello ' + name);
3 }
4
5 greet(); // Hello Max
6 greet('John'); // Hello John
```

Setting default values makes functions more flexible to use.



03. Use Template Literals

Template literals make string concatenation cleaner using backticks and \${expression} instead of plus signs. Even a kid could see this is an easier way to build strings!

```
JS script.js

1 const name = 'John';
2 const age = 24;

3

4 // Without template literals
5 const greeting = 'Hello ' + name + ', you are ' + age;

6

7 // With template literals
8 const greeting = `Hello ${name}, you are ${age}`;
```

Template literals remove lots of pesky concatenation. Neat!



04. Use Let & Const

Using let and const avoids unintended behavior from var. They make sure variables are blockscoped and constants can't be reassigned.

```
JS script.js
 1 // var is function scoped
   if(true) {
     var snack = 'chips';
5 console.log(snack); // chips 6 // fruit not defined here
```

```
JS script.js
1 // let and const are block scoped
  if(true) {
    let fruit = 'apples';
    const color = 'red';
```

05. Use Array Destructuring

Array destructuring works similarly to object destructuring, allowing you to extract array

elements into variables.

```
Js script.js
    const fruits = ['apples', 'oranges', 'bananas'];
  3 // Without destructuring
    const fruit1 = fruits[0];
    const fruit2 = fruits[1];
  7 // With destructuring
    const [fruit1, fruit2] = fruits;
```



06. Use Array Methods

JavaScript has handy array methods to help us write cleaner code. Things like map(), filter(), find(), reduce() etc. can avoid lots of loops and make code more expressive.

```
JS script.js

// Filter out all even numbers
const evenNumbers = numbers.filter(num ⇒
num % 2 == 0);

// Extract names from objects
const names = users.map(user ⇒ user.name);
```

JavaScript's array methods are super easy to grasp.



07. Use Ternary Operator

The ternary operator allows you to write one-line if/else statements in a simpler way.

```
JS script.js

// Without ternary
let message;
if(isLoggedIn) {
   message = 'Welcome back!';
} else {
   message = 'Please log in';
}
```

```
JS script.js

// With ternary
const message = isLoggedIn ? 'Welcome back!':
    'Please log in';
```



08. Use Object Methods

JavaScript gives objects built-in methods like Object.keys(), Object.values(), JSON.stringify() etc. Using these avoids reimplementing repetitive tasks.

```
JS script.js

1 // Get object keys
2 const keys = Object.keys(user);
3
4 // Convert object to JSON
5 const json = JSON.stringify(user);
```

The built-in object methods help keep code tidy and reusable.



09. Rest/Spread Properties

The rest/spread syntax allows us to handle function parameters and array elements in flexible ways.

```
JS script.js

// Rest parameters
function sum(...numbers) {
   return numbers.reduce((a, b) ⇒ a + b);
}

// Spread syntax
const newArray = [...array, 'new item'];

// Spread syntax
```

Did you Find it Useful?



@frontendcharm Ariba

Follow For More!







