Swipe Left

Top JavaScript Tricks For Cleaner Code

Ariba M.
@frontendcharm

Ariba M.
@frontendcharm

Keep Swiping...

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.

JS script.js

```
const user = {
    name: 'John',
    age: 24
};

// Without destructuring
const name = user.name;
const age = user.age;

// With destructuring
const { name, age } = user;
```

Ariba M.
@frontendcharm

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.

JS script.js

```
function greet(name = 'Max') {
  console.log('Hello ' + name);
}

greet(); // Hello Max
greet('John'); // Hello John
```

Setting default values makes functions more flexible to use.

Ariba M.
@frontendcharm

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

```
const name = 'John';
const age = 24;

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

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

Template literals remove lots of pesky concatenation. Neat!

Ariba M.
@frontendcharm

04. Use Let & Const

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

JS script.js

```
// var is function scoped
if(true) {
  var snack = 'chips';
}
console.log(snack); // chips
```

JS script.js

```
// let and const are block scoped
if(true) {
  let fruit = 'apples';
  const color = 'red';
}
// fruit not defined here
```

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'];

// Without destructuring
const fruit1 = fruits[0];
const fruit2 = fruits[1];

// With destructuring
const [fruit1, fruit2] = fruits;
```

Ariba M.
@frontendcharm

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.

Ariba M.
@frontendcharm

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';
```

Ariba M.
@frontendcharm

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

```
// Get object keys
const keys = Object.keys(user);

// Convert object to JSON
const json = JSON.stringify(user);
```

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

Ariba M.
@frontendcharm

That's it.

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'];
```