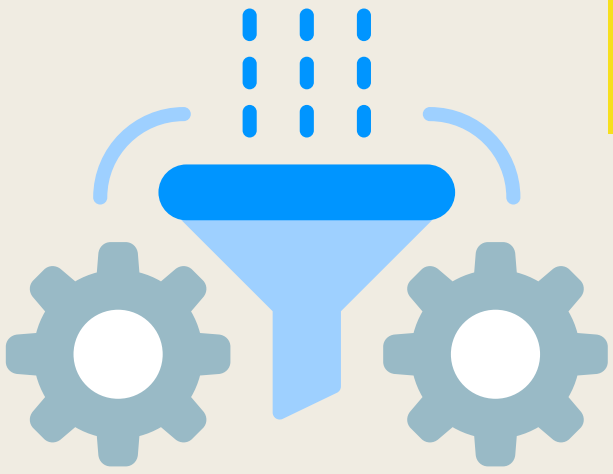
JAVASCRIPT TIPS BY MRFAIZANSHARIFF

BEST practices for writing efficient JavaScript code



SWIPE TO SEE MORE

JAVASCRIPT TIPS BY MRFAIZANSHARIFF

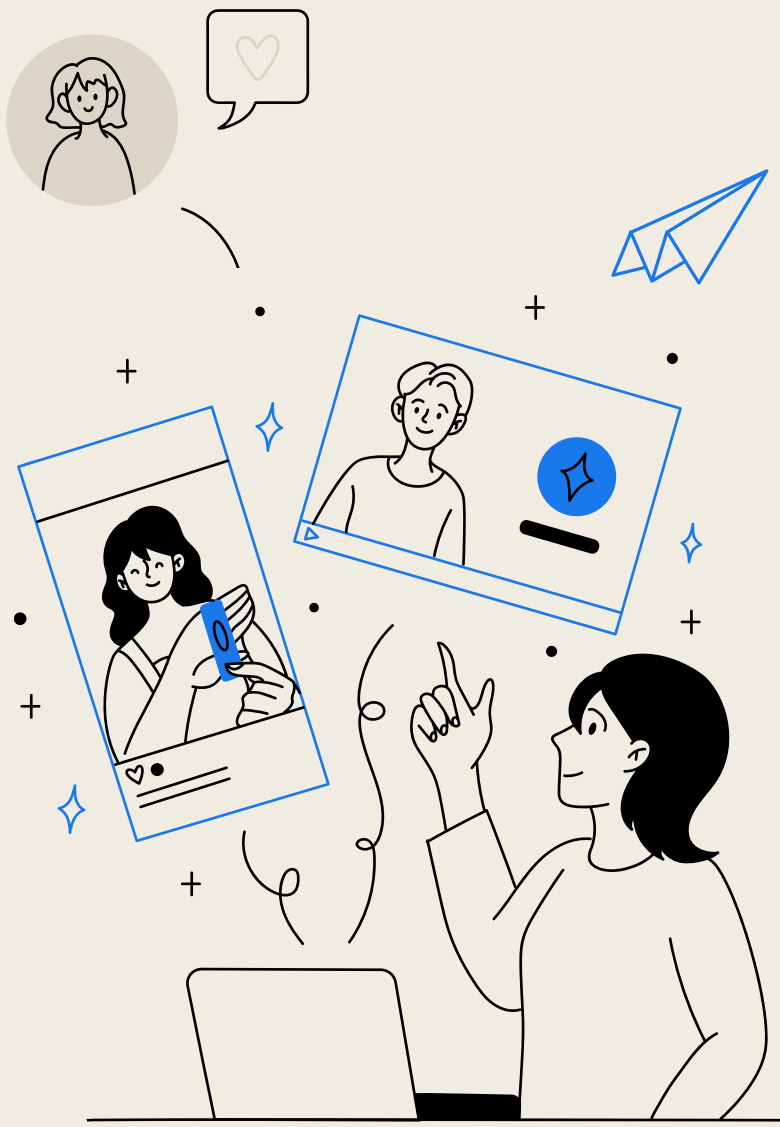
Variable Naming Convention

Use descriptive and meaningful names for your variables, functions, and classes.

Avoid using single-letter or vague names, as they can be confusing and make your code hard to read and understand.

It is also important to avoid using any of the following characters in variable names:

- Spaces
- Special characters (such as $, @, #, etc.)
- Reserved words (such as var, function, if, etc.)



{ JavaScript }

JAVASCRIPT TIPS BY MRFAIZANSHARIFF

Use *let* and *const* instead of *var*

In modern JavaScript, it is recommended to use let and const instead of the outdated var keyword.

let allows block scoping, which limits the scope of variables to the nearest enclosing block, while const ensures that a variable is not reassigned.

{ JavaScript }

JAVASCRIPT TIPS BY MRFAIZANSHARIFF

Build modular and specialized functions

No one function should have to do it all, for both efficiency and readability's sake.

design your functions to perform a single task and name them accordingly. This makes your code easier to reuse, test, and debug.

{ JavaScript }

JAVASCRIPT TIPS BY MRFAIZANSHARIFF

Comment your code often

Comments are essential to explain the logic and intention of your code, especially if it is complex or not obvious

Use comments to document your functions, variables, and algorithms, and to provide examples of usage or expected output.

{ JavaScript }

JAVASCRIPT TIPS BY MRFAIZANSHARIFF

Use template literals to combine strings

Template literals are a new feature of JavaScript that allow you to embed expressions and variables inside strings using backticks. They also support multi-line strings and interpolation.

EXAMPLE

```
const name = "John Doe";
const age = 30;
console.log(`Hello, ${name}!
You are ${age} years old.`);
```

{ JavaScript }

JAVASCRIPT TIPS BY MRFAIZANSHARIFF

Use promises or async/await to handle asynchronous operations

JavaScript is asynchronous, which means that some operations (such as network requests) do not block the execution of the code. Instead, they take a callback function that is executed when the operation is completed.

However, this can lead to nested callbacks that make the code hard to read and maintain. To avoid this, use promises or async/await to handle asynchronous operations in a more linear and readable way.

```
async function getData(url) {
try {
const response = await fetch(url);
const data = await response.json();
return data;
}
 catch (error) {
 console.error(error); }
}
```

{ JavaScript }

Thanks for reading.
Please leave your feedback in the comments.