



Mastering Console.log in JavaScript

Console.log is a powerful tool for JavaScript developers. It helps debug code, track variables, and understand program flow. This guide explores various console.log techniques for beginners.



by **Elizabeth Raglan**

Basic Logging Techniques

1 Simple Messages

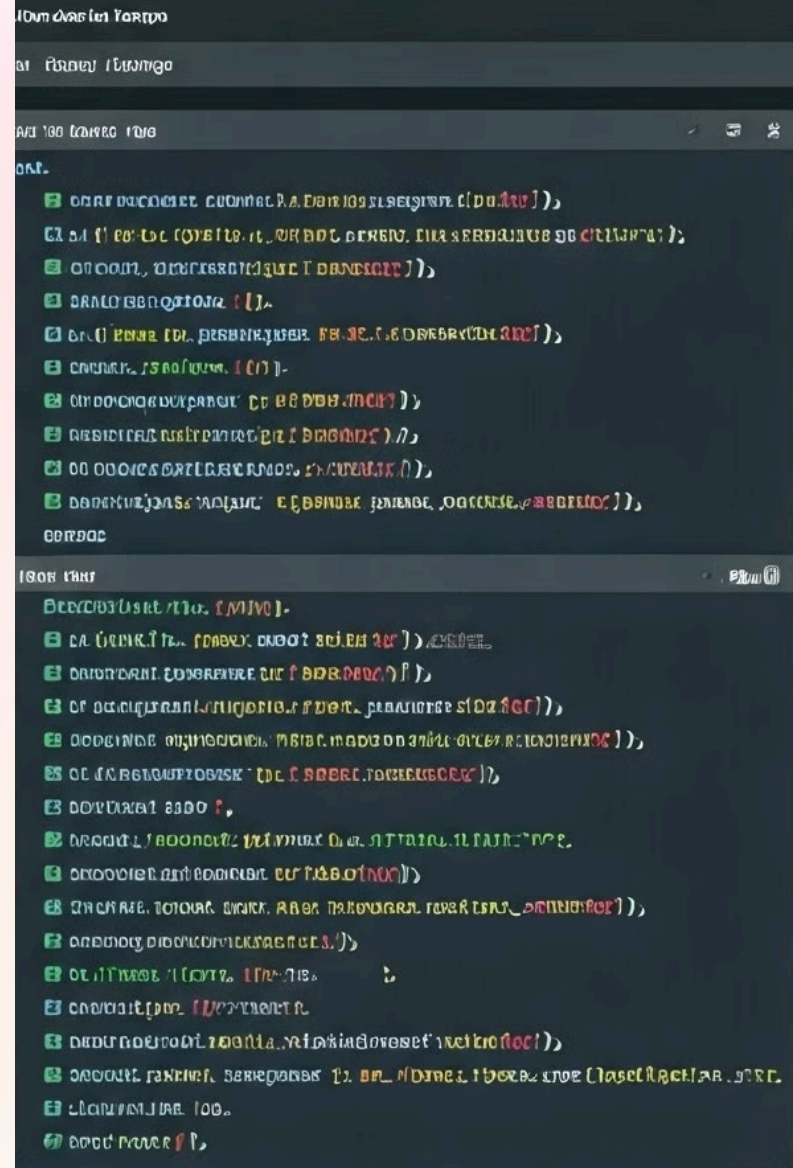
Use `console.log("Hello, world!")` to output basic strings. It's the simplest form of logging.

2 Variable Output

Log variables directly: `let name = "Alice"; console.log(name);`. This displays the variable's value.

3 String Interpolation

Use template literals for complex strings: `console.log(`My name is ${name}`)`. This embeds variables seamlessly.



```
const name = 'Alice';
const age = 30;
const isStudent = true;

// Simple Messages
console.log('Hello, world!');

// Variable Output
console.log(name);

// String Interpolation
console.log(`My name is ${name} and I am ${age} years old.`);

// Template Literals
const bio = `
  Name: ${name}
  Age: ${age}
  Student: ${isStudent}
`;
console.log(bio);
```

Advanced Object Logging

Standard Object Logging

Use `console.log(object)` to display an object's structure. It shows all properties and values.

Tabular Display

`console.table([objects])` presents data in a neat, organized table format. It's ideal for arrays of objects.

Expanded View

`console.dir(object, {depth: null})` shows nested object structures. It's useful for complex data.



Log Levels for Different Scenarios

1

Error Logging

Use `console.error()` for critical issues. It stands out in red in most consoles.

2

Warning Messages

`console.warn()` highlights potential problems. It typically appears in yellow for visibility.

3

Informational Logs

`console.info()` is for general information. It's useful for status updates and notifications.



Timing and Performance Tracking

1

Start Timer

Use `console.time("label")` to begin tracking time for a specific operation.

2

Execute Code

Run your JavaScript code between the timer start and end points.

3

End Timer

Call `console.timeEnd("label")` to stop the timer and log the elapsed time.

Debugging Techniques

Assertions

Use `console.assert(condition, message)` to test conditions. It logs only when the assertion fails.

Stack Traces

`console.trace()` shows the call stack. It's invaluable for understanding function call sequences.

Conditional Logging

Implement `if (debug) console.log()` for selective logging. It helps manage output in different environments.



Styling Console Output



Color

Use %c with CSS to add color: `console.log("%cColored text", "color: blue");`



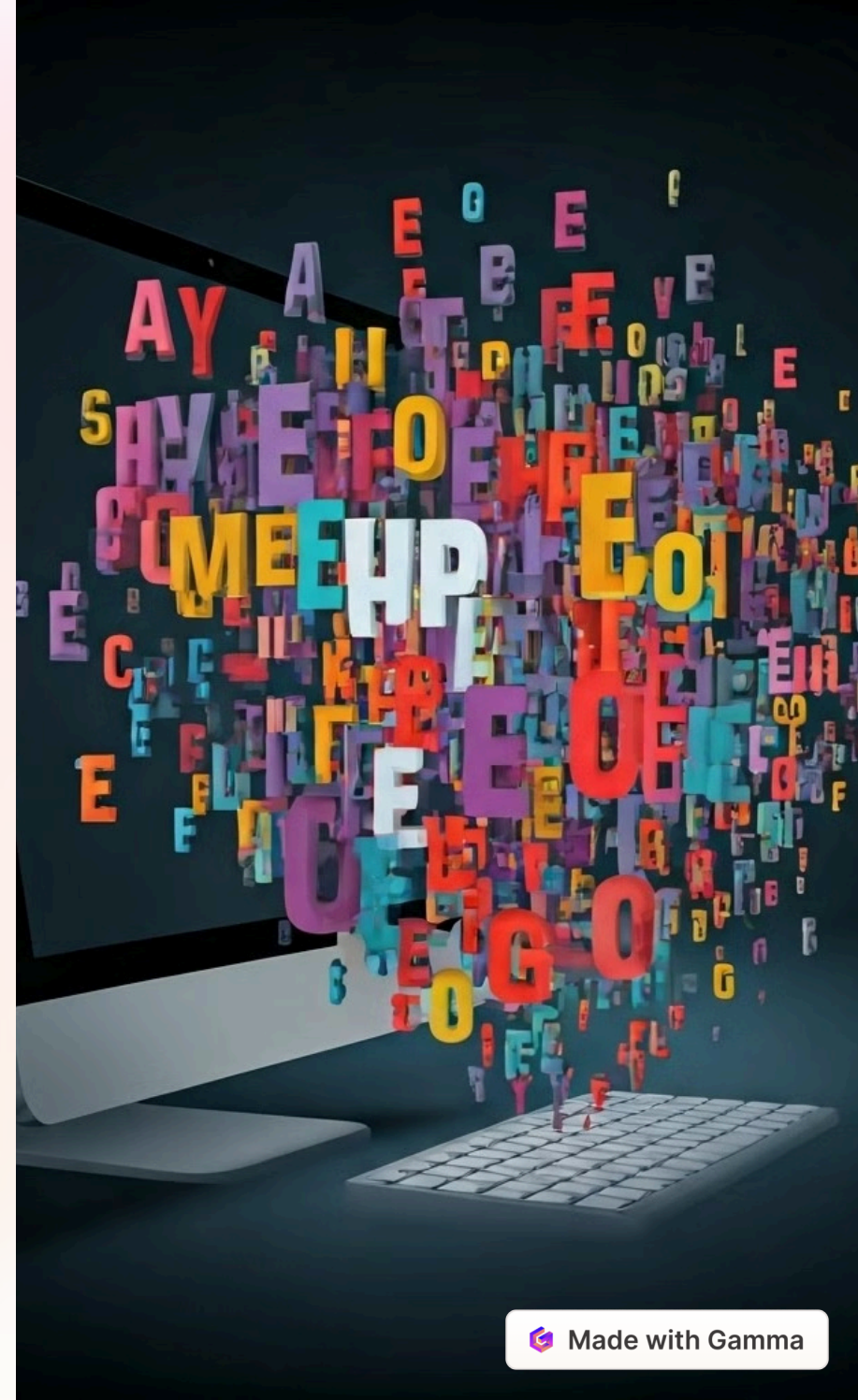
Font

Modify font styles: `console.log("%cBig text", "font-size: 20px");`



Images

Include images: `console.log("%c ", "background: url(image.png); background-size: cover;");`





Best Practices for Effective Logging

Use descriptive labels

Helps identify log sources quickly

Group related logs

Improves readability in complex scenarios

Clear console strategically

Keeps output clean and relevant

Use custom log functions

Ensures consistency across your application