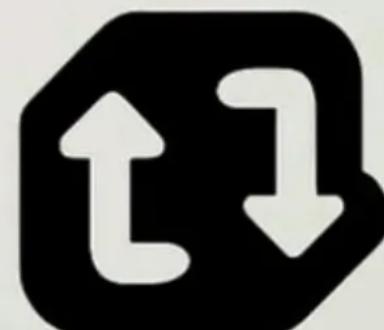


Say Goodbye to console.log in JavaScript



Follow
@asifaliquraishy



Repost



Why Move Beyond `console.log`? 🤔

Using `console.log` for everything can lead to messy debugging and unclear outputs, especially in complex applications. By leveraging more specialized console methods, you can:

- ⭐ Highlight critical information.
- 📚 Organize related logs.
- 👀 Visualize data in a clearer format.

Here are some of the best alternatives 

1

console.info()

- **When to use:** For informational messages that aren't warnings or errors.
- **Result:** Displays the message with an “info” style.



console.info

```
1 console.info("Data loaded successfully! 🚀");
```

Data loaded successfully! 🚀

>

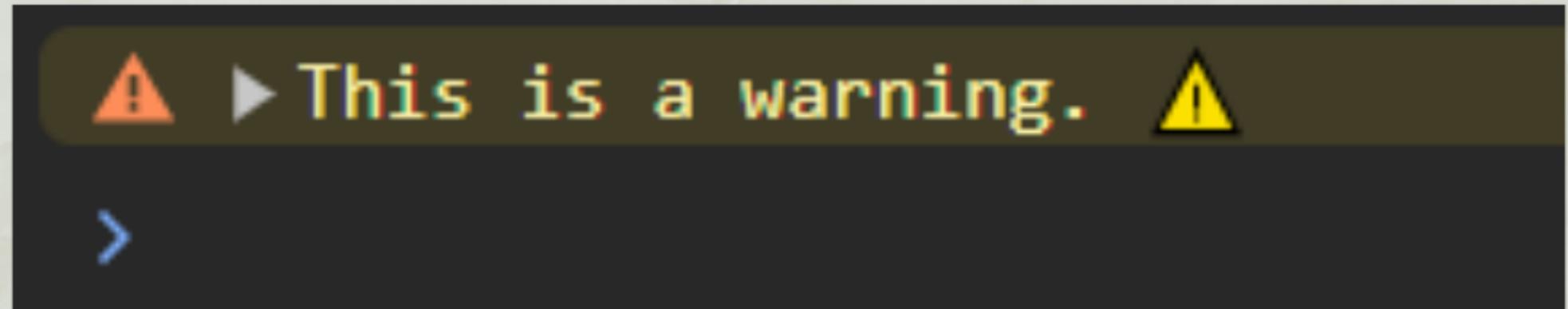
2

console.warn() !

- **When to use:** To highlight potential issues that don't require immediate action.
- **Result:** Displays a warning styled with a yellow background or icon in most browsers.



```
● ● ●          console.warn  
1 console.warn("This is a warning !");
```



3

console.error() X

- **When to use:** For logging critical errors that need immediate attention.
- **Result:** Displays an error styled with a red background or icon.



console.error

```
1 console.error("Failed to fetch data from the  
API. X");
```



► Failed to fetch data from the API. X



4

console.table()



- **When to use:** For displaying tabular data (arrays or objects) in an easy-to-read table format.
- **Result:** Renders a table in the console with rows and columns.



console.table

```
1 const users = [  
2   { id: 1, name: "Alice", role: "Admin" },  
3   { id: 2, name: "Bob", role: "User" },  
4 ];  
5 console.table(users);
```

script.js:9

(index)	id	name	role
0	1	'Alice'	'Admin'
1	2	'Bob'	'User'

▶ Array(2)

>

5

console.dir()



- **When to use:** To inspect JavaScript objects or DOM elements.
- **Result:** Displays an expandable tree structure.



console.dir

```
1 const element = document.querySelector("#app");
2 console.dir(element);
```

```
> #app
  └─ children
  └─ innerHTML
  └─ classList
    └─ ...
```

6

console.group() / console.groupEnd()



- **When to use:** To group related logs together for better organization.
- **Result:** Groups the logs in an expandable/collapsible format.



console.group/.groupEnd

```
1 console.group("User Details 🧑💻");
2 console.log("Name: Alice");
3 console.log("Age: 25");
4 console.log("Role: Admin");
5 console.groupEnd();
```

▼ User Details 🧑💻

Name: Alice

Age: 25

Role: Admin

7

console.time() / console.timeEnd()



- **When to use:** To measure the execution time of code blocks.
- **Result:** Displays the time taken in milliseconds.



console.time/.timeEnd

```
1 console.time("API Fetch ⏳");
2 setTimeout(() => {
3   console.timeEnd("API Fetch ⏳");
4 }, 2000);
```

API Fetch ⏳: 2002.908935546875 ms



8

debugger



- **When to use:** To pause code execution and inspect variables interactively.
- **Result:** Opens the browser's debugger tool and pauses code execution.



debugger

```
1 const data = fetchData();
2 debugger; // Opens the browser's debugger tool.
3 processData(data);
```

Execution pauses, allowing you to step through the code in your browser's developer tools.

9

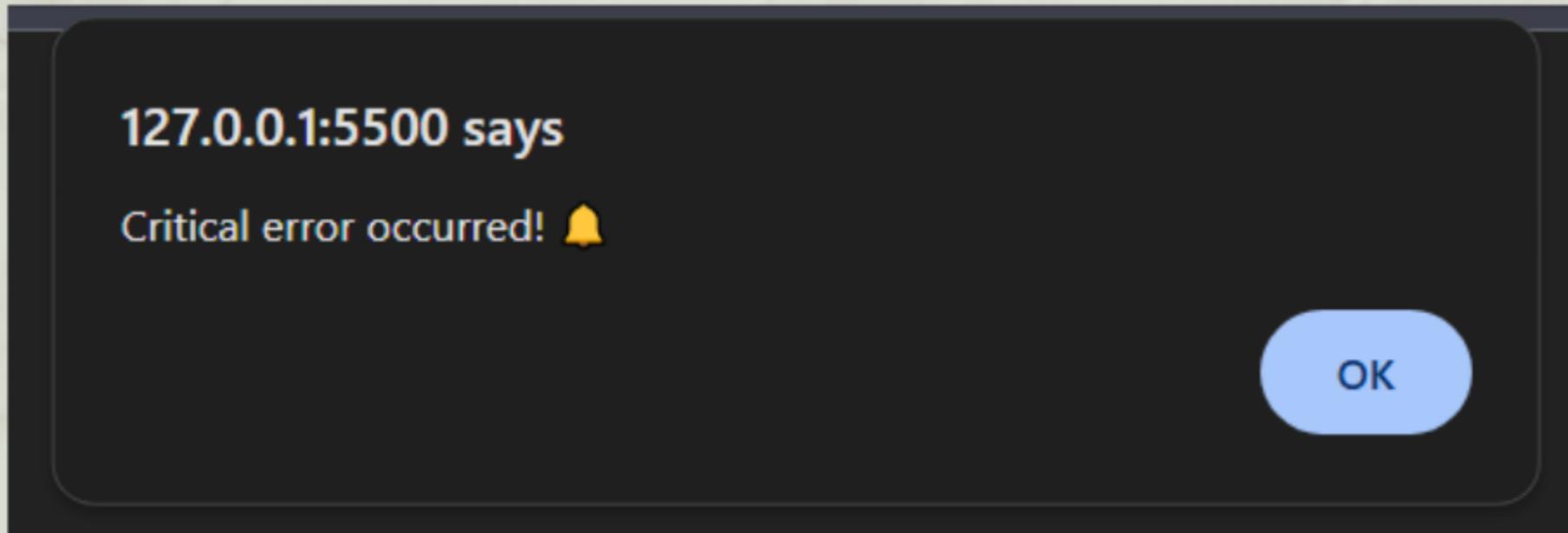
alert() for Critical Messages



- **When to use:** To notify users of important updates during development.
- **Result:** Displays a pop-up alert message.



```
● ● ●           alert  
1 alert("Critical error occurred! 🎗");
```



Wrapping It All Up



While **console.log** is a reliable go-to tool, these alternatives can make your debugging process more effective and insightful. By choosing the right method for the right task, you'll:



- Save time during debugging.
- Keep your logs organized.
- Improve collaboration with your team.

Next time you're debugging or logging, try out one of these methods!

Found this **helpful**?



Share to help others



Save to refer to later



Follow Hadil Ben Abdallah for more
amazing content about programming