Beginner’s Guide to JavaScript Debugging & Error Handling: Write Bug-Free Code

Subtitle: Learn how to identify, fix, and prevent errors to make your JavaScript code reliable.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Bugs are inevitable in coding, but knowing how to **debug and handle errors** makes you a better developer. This guide teaches beginners practical techniques to identify issues, fix them efficiently, and write more reliable JavaScript code.

Step 1: Common Types of Errors

- **Syntax Errors:** Mistyped code or missing brackets
- **Reference Errors:** Accessing undefined variables
- **Type Errors:** Using incorrect data types
- **Logical Errors:** Code runs but produces wrong output

Step 2: Using Console for Debugging

- `console.log()` – Print variable values
- `console.error()` – Highlight errors
- `console.table()` – Display arrays/objects clearly

```
const user = {name: "Hafsa", age: 20};
console.log(user.name);  // Hafsa
console.table(user);     // Nicely formatted table
```

Step 3: Browser DevTools

- Use Chrome/Firefox DevTools
- Features:
    - Inspect DOM

- Breakpoints to pause code
- Watch variables and call stack

- **Tip:** Step through code line by line to find issues

Step 4: Try-Catch for Error Handling

- Catch runtime errors without breaking the program

```
try {
  let result = riskyFunction();
} catch (error) {
  console.error("Error occurred:", error.message);
}
```

- Optionally use `finally` for cleanup tasks

Step 5: Throwing Custom Errors

- Create meaningful errors for validation

```
function divide(a, b) {
  if(b === 0) throw new Error("Cannot divide by zero");
  return a / b;
}

try {
  divide(5,0);
} catch (e) {
  console.error(e.message);
}
```

Step 6: Debugging Asynchronous Code

- Use `async/await` with `try-catch`

```
async function fetchData() {
  try {
    const res = await fetch('https://api.example.com/data');
    const data = await res.json();
```

```
    console.log(data);
  } catch (err) {
    console.error("Fetch error:", err);
  }
}
```

Step 7: Common Debugging Techniques

- **Break down problems:** Test small pieces of code first
- **Rubber duck debugging:** Explain code line by line
- **Check assumptions:** Verify inputs, outputs, and conditions
- **Comment out sections:** Identify where errors occur

Step 8: Logging Best Practices

- Log only necessary information in production
- Avoid logging sensitive user data
- Use descriptive messages for easier debugging

Step 9: Mini Projects to Practice

- Calculator app with validation for divide-by-zero
- Form validation with custom error messages
- API fetch app with proper `try-catch` error handling

Step 10: Key Takeaways

- Use console logs and DevTools to identify errors
- Handle errors gracefully with `try-catch`
- Throw custom errors for better control and debugging
- Debugging is a skill — practice consistently on small projects

---

Visit **haas.dev** for step-by-step debugging tutorials, error handling exercises, and beginner-friendly JavaScript resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app
