JavaScript Engineering Fundamentals

PDF 3 —Writing Your First JavaScript Program: Setting Up, Running Code, and Understanding the Basics

July 5, 2026

Contents

1 Introduction

You now understand what JavaScript is and how it works inside a browser. The next essential step is learning how to actually write and run JavaScript code. Before building interactive websites, it is critical to grasp several foundational concepts: where JavaScript code is placed, the mechanism by which browsers execute it, how to observe program output effectively, and how to identify and fix common errors.

Professional developers spend a significant portion of their time utilizing debugging tools. By mastering these skills early on, you set yourself up for smoother progress and greater confidence as you tackle more advanced programming topics.

2 Why Your First Program Matters

Your initial program is not about creating something complex or impressive. Instead, its primary purpose is to familiarize you with the fundamental process of coding in JavaScript. This includes creating JavaScript files, linking them to HTML documents, running the code, observing the results, and learning how to fix mistakes that inevitably arise.

Think of this stage as learning to use your tools before embarking on larger projects. Mastery of these basics provides a sturdy foundation on which to build more sophisticated applications.

3 Setting Up a JavaScript Project

A typical beginner’s JavaScript project generally contains three separate files: an HTML file for structure, a CSS file for styling, and a JavaScript file for behavior. Separating these concerns enhances organization and maintainability.

As projects grow more complex, adhering to a good file structure becomes increasingly important to keep your work manageable and scalable over time.

4 Ways to Add JavaScript to a Webpage

4.1 Internal JavaScript

Internal JavaScript involves writing code directly inside the HTML document within `<script>` tags. This method is useful for small examples and learning purposes but is generally not recommended for larger projects.

4.2 External JavaScript

External JavaScript places code in separate `.js` files, which are then linked to the HTML. This approach keeps your HTML clean and allows reuse of code across multiple pages, making it the preferred practice for professional development.

4.3 Inline JavaScript

Inline JavaScript is written directly inside HTML attributes (like `onclick`). While functional, professional developers typically avoid this method because it mixes structure and behavior, making maintenance and debugging harder.

5 External vs Internal JavaScript

For production-level projects, external JavaScript is cleaner and more maintainable. It enables multiple pages to share the same scripts, facilitating reuse and scalability.

Internal scripts mainly serve well for demonstrations or very simple pages but lack the organizational benefits of external files.

6 Understanding the `<script>` Tag

The `<script>` element instructs the browser to execute JavaScript code. Its placement within the HTML affects when and how the script runs. If JavaScript executes before the browser finishes building the page's structure, your code can fail to locate elements it needs to interact with.

Understanding where to place `<script>` tags is crucial to avoid many common beginner errors and ensure your scripts execute as intended.

7 When JavaScript Executes

Browsers load and process web pages in stages. JavaScript might run while the page is still loading, after the HTML structure is available, or only after the entire page finishes loading.

Selecting the correct timing for script execution guarantees that your code behaves predictably and interacts with the page elements correctly.

8 Using the Browser Console

The browser console is an invaluable tool for developers. It allows you to view output, inspect variables, test snippets of code, identify errors, and understand program behavior in real time.

Experienced developers rely on the console daily, making it a fundamental skill for any programmer to master early.

9 Your First JavaScript Statement

Your first JavaScript program does not need to be complex. The main goal is to verify that your JavaScript file is correctly linked, the browser successfully executes the code, and output appears in the console.

This initial confirmation ensures your development environment is properly set up and ready for further coding.

10 Understanding Output

Programs produce output to communicate results or statuses. During development, the console serves as the safest and most straightforward place to display output.

Later, you will learn techniques to present results directly on webpages. For now, rely on the console as your primary communication channel with your program.

11 Writing Comments

Comments are annotations in your code intended for human readers rather than the computer. They serve several purposes:

- Documenting ideas and intentions behind code
- Explaining complex logic that may not be immediately clear
- Temporarily disabling code during debugging
- Enhancing maintainability by clarifying your thought process

Well-crafted comments focus on the “why” rather than just the “what,” adding valuable context to your codebase.

12 Strict Mode

Strict Mode is a feature in JavaScript that enforces stricter parsing and error handling. It helps catch common coding mistakes and "unsafe" actions, encouraging better coding practices and making your code more robust.

Activating Strict Mode is as simple as adding `"use strict";` at the beginning of your scripts. This disciplined approach helps prevent subtle bugs early in development.

13 Common JavaScript Errors

Beginners often encounter typical errors such as syntax mistakes, undefined variables, or type mismatches. Understanding these errors is critical to becoming proficient.

Errors can be broadly categorized:

- Syntax errors: Mistakes in the structure of code
- Reference errors: Using variables that haven't been declared
- Type errors: Performing operations on incompatible data types

Recognizing these errors quickly reduces frustration and accelerates learning.

14 Reading Error Messages

Error messages provide vital clues to diagnose problems. They typically indicate the type of error, the file name, and the line number where it occurred.

By carefully examining these messages and tracing back to your code, you can pinpoint issues more efficiently, turning errors into valuable learning opportunities.

15 Using Developer Tools

Modern browsers offer integrated Developer Tools that provide a suite of debugging features beyond the console. These tools allow you to inspect the DOM, monitor network requests, profile performance, and step through code execution line-by-line.

Gaining familiarity with Developer Tools early will accelerate your debugging skills and improve code quality.

16 Organizing JavaScript Files

As your projects grow, organizing JavaScript files becomes essential. Separating code into modules or logically grouped files increases readability and maintainability.

A well-structured project might include dedicated folders for utilities, components, and third-party libraries, making collaboration and scaling easier.

17 Real-World Examples

Consider a simple webpage that loads a JavaScript file to display a greeting message in the console. By observing the console output and tweaking the code, beginners grasp the flow of connection, execution, and output.

Such straightforward examples build confidence and demonstrate practical applications of concepts learned.

18 Common Beginner Mistakes

Some frequent pitfalls include:

- Placing `<script>` tags improperly causing code to run too early
- Forgetting to link external JavaScript files
- Omitting semicolons (though optional, can cause confusion)
- Misusing variable declarations
- Ignoring error messages or not using debugging tools

Being aware of these errors helps you avoid them and develop best practices from the start.

19 Practical Action Plan

To solidify your foundation:

1. Set up a basic project with separate HTML, CSS, and JavaScript files.
2. Write a simple JavaScript statement that logs a message to the console.
3. Experiment with adding comments and enabling Strict Mode.
4. Use the browser console and Developer Tools to observe output and debug.
5. Deliberately introduce errors to practice reading error messages.

This hands-on approach reinforces learning and builds essential skills.

20 Mini Project

Create a webpage that:

- Includes an external JavaScript file.
- Logs a personalized greeting message to the console.

- Contains comments explaining each step.
- Uses Strict Mode.
- Demonstrates error handling by fixing an intentional mistake.

This mini project integrates all concepts covered and prepares you for more complex tasks.

21 Key Takeaways

- Your first JavaScript program is about mastering the development environment.
- Understanding the flow of code execution and output is fundamental.
- Browser Developer Tools and the console are indispensable for debugging.
- Strict Mode enhances code reliability and prevents common errors.
- Clear comments significantly improve code maintainability.

These fundamentals form the backbone of proficient JavaScript development.

22 Summary Page

This chapter introduced you to writing and running your first JavaScript program. You learned about project setup, script placement, debugging basics, and best practices such as using Strict Mode and writing comments.

Mastering these concepts early paves the way for exploring programming fundamentals with confidence.

23 Beginner JavaScript Checklist

Item	Status / Notes
Set up project files (HTML, CSS, JS)	
Link external JavaScript file correctly	
Write and run a simple `console.log` statement	
Use browser console to view output	
Write comments explaining code	
Enable Strict Mode in scripts	
Identify and fix syntax errors	
Use Developer Tools for debugging	
Understand script tag placement	
Recognize common beginner mistakes	

24 Related Resources

- MDN Web Docs: JavaScript Guide
- JavaScript Info: The Modern JavaScript Tutorial
- Browser Developer Tools: Chrome DevTools
- Interactive coding playground: CodePen
- Video Tutorials and Courses: freeCodeCamp

25 Recommended Next Learning Path

After mastering your first JavaScript program, consider progressing to:

- Learning JavaScript data types and variables
- Understanding control flow and loops
- Exploring functions and scope
- Diving into DOM manipulation techniques
- Introducing event handling in web pages

Each step builds upon your foundational skills and brings you closer to professional JavaScript development.

Website: haas.dev