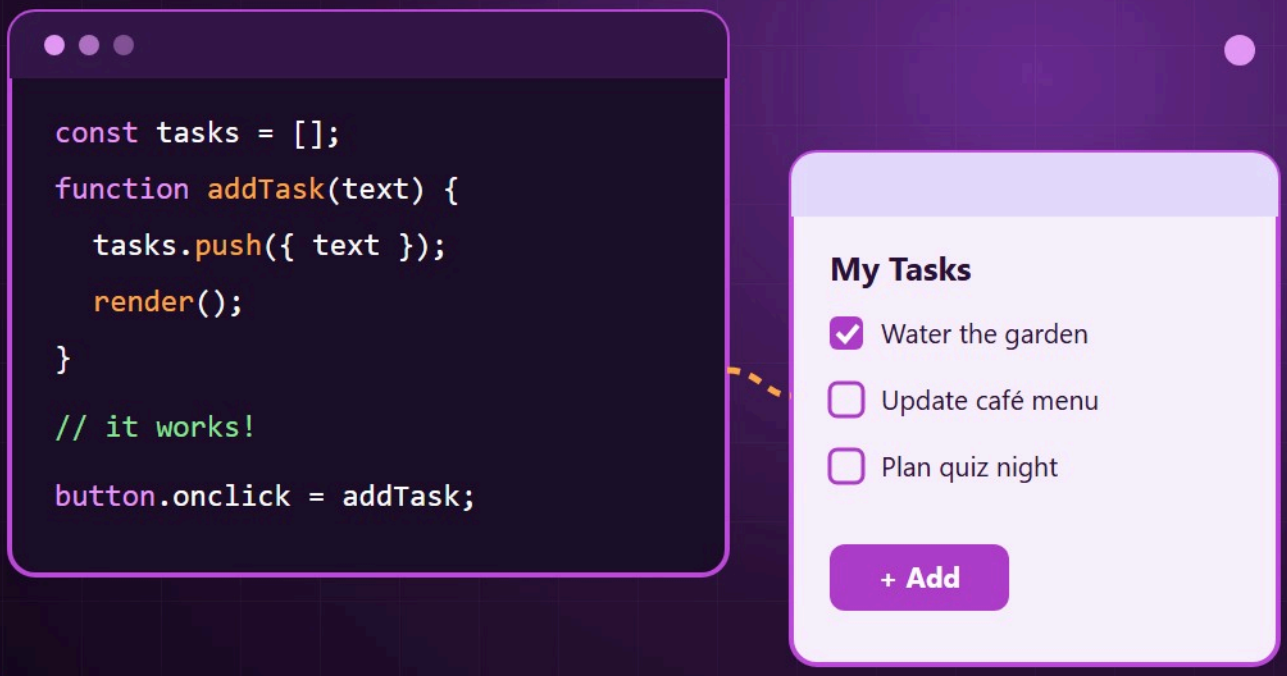


LEARN TO CODE BY BUILDING

JavaScript Fundamentals Through Projects

Learn to Code by Building Real Projects, Step by Step

For complete beginners · No frameworks · Just HTML, CSS, and modern JavaScript in your browser

BUILD · BREAK · FIX · SHIP

THENEXBOOKS

JavaScript Fundamentals Through Projects

Learn to Code by Building Real Projects, Step by Step

First Edition

THENEXBOOKS

JavaScript Fundamentals Through Projects
Learn to Code by Building Real Projects, Step by Step

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. The code in this book may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Browsers, tools, and web services change over time; consult the official documentation for the versions you use.

This book is an independent publication. JavaScript is a trademark of Oracle and/or its affiliates. Node.js is a trademark of the OpenJS Foundation. Visual Studio Code, GitHub, Chrome, Firefox, Safari, Edge, and other product names are trademarks of their respective owners; their use here is for identification only. Bramblewood and all people and businesses in the examples are fictional.

Preface

The best way to learn to program is to build things. Reading about variables and loops is useful, but the ideas only truly click when you use them to make something that works — a button that does what you expect, a list that remembers what you typed, a game that keeps score. That moment when your own code runs in the browser is what keeps people learning.

This book teaches JavaScript from the very beginning, entirely through projects. You do not need any programming experience. In every chapter we introduce a small set of ideas, explain them patiently with short examples, and then use them to build a complete, working app, one step at a time. By the end you will have built more than twenty-five projects — from a bill splitter and a guessing game to a to-do app, a timer, an events board that loads real data, a quiz game, a budget tracker, and a recipe box — and you will understand the fundamentals well enough to build your own.

We use plain, modern JavaScript that runs in any current browser, with no frameworks and no complicated tools. When you finish, you will be ready for whatever comes next, whether that is a framework such as React, back-end development with Node.js, or TypeScript.

Who This Book Is For

- **Complete beginners** who want a friendly, structured path into programming.
- **Students** taking a first web development or programming course.
- **Designers, marketers, and career changers** who want to add interactivity to web pages and understand how the web works.
- **Self-taught learners** who have watched tutorials but want to build complete projects on their own.

What You Will Learn

- Programming fundamentals: values, variables, operators, strings, numbers, decisions, loops, and functions.
- Working with data: arrays, objects, and the array methods `map`, `filter`, `find`, and `reduce`.
- How to read error messages and debug with confidence.
- Just enough HTML and CSS, and how to change web pages with the DOM, events, and forms.
- Saving data with localStorage and JSON, working with dates and timers, and organizing code with modules and classes.
- Asynchronous JavaScript, fetching data, and building search, filters, and friendly error handling.
- Larger projects, a first look at Node.js, and how to share your work on the web.

How This Book Is Organized

Part I — First Steps (Chapters 1–5) gets you set up and teaches the core building blocks. **Part II — Building Blocks** (Chapters 6–10) covers functions, arrays, objects, array methods, and debugging. **Part III — The Browser and the DOM** (Chapters 11–15) connects your code to real web pages and ends with a complete to-do app. **Part IV — Data, Time, and Organization** (Chapters 16–19) adds saving, dates, timers, modules, and classes. **Part V — Talking to the Internet** (Chapters 20–23) introduces asynchronous code and fetching data. **Part VI — Bigger Projects and Next Steps** (Chapters 24–28) brings everything together in larger apps, introduces Node.js, and shows you how to share your projects. The appendices provide a cheat sheet, starter templates, checklists, and a glossary.

Every chapter ends with **Try It Yourself** challenges that extend the project and a **Key Takeaways** box for quick review.

Welcome to Bramblewood

The projects in this book are built for **Bramblewood**, a fictional small town. You have just joined the coding club at the **Bramblewood Community Center**, where the coordinator, Rosa Delgado, keeps finding small problems that a bit of code could solve. The librarian at the Bramblewood Public Library, the owner of the Bean & Bramble Café, and the volunteers at the community garden all have requests too. Building for real (if fictional) people gives every project a purpose — and makes it much easier to decide what your app should do.

How to Use This Book

Type the code yourself rather than copying it. It feels slower, but typing forces you to notice every detail, and fixing your own typos is excellent practice at debugging. Build each project step by step, run it often, and check the result before moving on. When something breaks — and it will, for everyone — read the error message, compare your code with the book, and remember that finding and fixing mistakes is a normal, everyday part of programming.

Conventions Used in This Book

> **TIP**
>
> A recommended practice or a shortcut that makes your work easier.

> **NOTE**
>
> Background information or a useful aside.

> **WARNING**
>
> A common mistake or something that can cause bugs or security problems.

Code appears in `monospace`. When a code block shows a file, its path appears above it, for example `projects/ch07-shopping-list/script.js`. Output from the browser console appears in a box labeled **Console output**.

Contents

Preface

PART I — FIRST STEPS

1 Hello, JavaScript: Setting Up and Your First Program

2 Values, Variables, and Operators

3 Working with Strings and Numbers

4 Making Decisions with if, else, and switch

5 Loops and Repetition

PART II — BUILDING BLOCKS

6 Functions: Reusable Pieces of Code

7 Arrays: Working with Lists

8 Objects: Grouping Related Data

9 Array Methods: map, filter, find, and reduce

10 Debugging and Handling Errors

PART III — THE BROWSER AND THE DOM

11 Just Enough HTML and CSS for JavaScript

12 The DOM: Finding and Changing Elements

13 Events: Responding to Clicks, Keys, and Input

14 Forms and Validation

15 Project: Building a To-Do App

PART IV — DATA, TIME, AND ORGANIZATION

16 Saving Data with localStorage and JSON

17 Dates and Times

18 Timers and Simple Animation

19 Organizing Code: Modules and Classes

PART V — TALKING TO THE INTERNET

20 Asynchronous JavaScript: Callbacks, Promises, and async/await

21 Fetching Data from APIs

22 Search, Filter, and Sort

23 Handling Errors, Loading, and Empty States

PART VI — BIGGER PROJECTS AND NEXT STEPS

24 Project: Quiz Game with Scores and High Scores

25 Project: Budget Tracker with a Simple Chart

26 Project: Recipe Box — a Multi-View App

27 JavaScript Beyond the Browser: A First Look at Node.js

28 Next Steps: Sharing Your Projects and Keep Learning

APPENDICES

A JavaScript Cheat Sheet

B Project Starter Templates and DOM Reference

C Checklists

D Glossary

PART I

First Steps

Set up your tools, write your first program, and learn the core building blocks — values, variables, strings, numbers, decisions, and loops — by building small apps from day one.

1 · Hello, JavaScript — Welcome Board

2 · Values and Variables — Café Bill Splitter

3 · Strings and Numbers — Name Badge and Unit Converter

4 · Decisions — Event Ticket Price Calculator

5 · Loops — Number Guessing Game

CHAPTER 1 · PART I — FIRST STEPS

Hello, JavaScript: Setting Up and Your First Program

In this chapter we set up a small, free toolkit, write our very first lines of JavaScript, and finish with a real web page: a Welcome Board for the Bramblewood Community Center that greets visitors and shows today's date.

Welcome to the Coding Club

Picture a bright room at the back of the Bramblewood Community Center. There are a few tables, a slightly wobbly whiteboard, and a sign on the door that says "Coding Club — everyone welcome." You have just signed up as a volunteer. Rosa Delgado, the center's coordinator, has a list of small problems the town would love to solve with simple web apps: a board that greets visitors, a calculator for splitting the bill at the Bean & Bramble Café, a price calculator for event tickets, and many more. By the end of this book you will have built every one of them.

You do not need any experience to start. If you have never written a line of code, you are exactly the reader this book was written for. We will go slowly, explain every new word the first time it appears, and build each project in small steps so you can see your progress on the screen after every few lines. When something goes wrong, we will read the error together and fix it.

Here is the plan for this first chapter:

1. Understand what JavaScript is and what it does in a web page.
2. Install a code editor and learn to open the browser's built-in developer tools.
3. Type our first JavaScript commands directly into the browser console.
4. Create a project folder with three files: an HTML page, a stylesheet, and a script.
5. Build the Welcome Board step by step, ending with the complete working files.

What Is JavaScript?

Every web page you visit is built from three languages that work as a team. It helps to think of the community center's front noticeboard:

- **HTML** is the *content* of the board: the headings, the paragraphs, the list of this week's events. HTML describes what is on the page and what each piece is (a heading, a list, a button).
- **CSS** is the *look* of the board: the colors, the fonts, the spacing, the border around the edge. CSS describes how things appear.

- **JavaScript** is the *behavior*: the part that can change the board by itself. It can write today's date, react when someone clicks a button, calculate a total, or remember what you typed yesterday.

HTML and CSS describe a page that stays the same until someone edits the file. JavaScript is a real programming language: you write **instructions**, and the browser follows them one after another. That is what makes a page feel alive. When a shopping site updates the total as you change a quantity, JavaScript is doing that work.

Every modern web browser — Chrome, Edge, Firefox, Safari and others — has a JavaScript engine built in. That is great news for us: there is nothing special to install to *run* JavaScript. If you have a browser, you already have everything needed to run the programs in this book. We only need a comfortable place to *write* them.

> **NOTE**
> Despite the similar name, JavaScript and Java are two different languages with different rules and different uses. The name is a historical accident. Everything in this book is JavaScript, and you do not need to know anything about Java.

Setting Up Your Toolkit

Our toolkit has only three parts, and all of them are free:

Tool	What it does for us
A web browser	Opens our pages and runs our JavaScript
A code editor	Where we write and save our files
Developer tools	Built into the browser; shows messages and errors

A browser

Use any current, up-to-date browser; the examples in this book work in all the major ones. If you are unsure, pick the browser you already use every day and make sure it is updated.

A code editor: Visual Studio Code

You could write JavaScript in a plain text app like Notepad, but a code editor makes life much easier. It colors different parts of your code so they are easier to read, it indents lines for you, it warns about some mistakes as you type, and it lets you see all the files of a project in one window. We will use **Visual Studio Code** (usually called VS Code), a free editor that runs on Windows, macOS and Linux.

1. Search the web for "Visual Studio Code download" and open the official download page from Microsoft.
2. Download the installer for your operating system and run it, accepting the default options.
3. Open VS Code. You will see a welcome tab; you can close it.

That is the whole installation. VS Code has many optional extensions, but we do not need any of them yet.

> **TIP**
>
> Turn on **Auto Save** in VS Code (open the **File** menu and click **Auto Save** so it has a check mark). Beginners very often edit a file, switch to the browser, and wonder why nothing changed — because the file was never saved. With Auto Save on, your changes are written to disk automatically. A white dot on a file's tab means it has unsaved changes.

Developer tools and the console

Every major browser hides a set of **developer tools** (often shortened to DevTools) behind a keyboard shortcut. These tools let you look inside any web page. The part we will use most is the **console**: a panel where JavaScript can print messages, where errors appear in red, and where you can type JavaScript and run it immediately.

Open any web page (even a blank tab) and try one of these shortcuts:

Browser	Windows / Linux	macOS
Chrome, Edge	Ctrl+Shift+J	Cmd+Option+J
Firefox	Ctrl+Shift+K	Cmd+Option+K
Most browsers	F12, then Console tab	Cmd+Option+I

Safari needs one extra step the first time: open Safari's settings, go to the **Advanced** section, and turn on the option to show features for web developers. After that, a **Develop** menu appears, and it contains the JavaScript console. You can also right-click anywhere on a page and choose **Inspect**, then click the **Console** tab.

The console usually opens at the bottom or the side of the browser window. You will see a blinking cursor next to a small `>` symbol. That is where you type. You can clear old messages with the clear button (often a circle with a line through it).

Your First Lines of JavaScript

Let us talk to the browser. With the console open, click next to the `>` symbol, type the following, and press Enter:

```
2 + 3
```

The console answers `5`. Congratulations — you just ran a JavaScript program. It was a very short program, but the browser read your instruction, worked out the answer, and showed you the result. Try a few more, pressing Enter after each line:

```
10 * 4
"Bramble" + "wood"
```

The first line multiplies (the `*` symbol means "times") and shows `40`. The second line joins two pieces of text and shows `'Bramblewood'`. Text in quotes is called a **string**, and the `+` sign glues strings together. We will learn much more about numbers and strings in the next two chapters.

console.log: printing a message

Typing a calculation straight into the console is handy for quick experiments, but real programs live in files, and they need a way to print messages on purpose. For that we use `console.log`. Type this and press Enter:

```
console.log("Hello, Bramblewood!");
```

Console output

```
Hello, Bramblewood!
```

Let us take this line apart slowly, because almost every program in this book uses the same shape:

- `console` is a built-in object the browser gives us — think of it as the console panel itself.
- The dot (`.`) means "the thing that belongs to". So `console.log` is "the log tool that belongs to the console".
- `log` is a **function**: a named action that does a job when we ask it to. Its job is to write a message into the console.
- The parentheses `( )` are how we *call* (run) a function. Whatever we put inside is the information we hand to it.
- `"Hello, Bramblewood!"` is the string we hand over. The quotes mark where the text starts and ends; they are not printed.
- The semicolon `;` marks the end of the instruction, like a period at the end of a sentence.

One complete instruction like this is called a **statement**. A program is simply a list of statements that the browser runs from top to bottom.

> **WARNING**
>
> Some browsers show a warning the first time you paste code into the console and ask you to type a phrase such as "allow pasting" before they accept it. This protection exists because scammers sometimes tell people to paste code into their console to "unlock" something — and that code can take over the account they are logged into. Only paste code you understand, such as the examples in this book. Never paste code a stranger sends you.

Mistakes are normal (and helpful)

Let us make a mistake on purpose so the first error you see is one you expected. Type this, with `console` misspelled:

```
consol.log("Hello");
```

The console shows a red message that contains the words `consol is not defined`, labeled as a `ReferenceError`. In plain English: "You asked for something called `consol`, and nothing by that name exists." The browser is not angry, and nothing is broken. It is telling you exactly what it did not understand. Fix the spelling and run the line again.

Now try forgetting the closing quote:

```
console.log("Hello);
```

This time you get a `SyntaxError`. **Syntax** means the grammar of the language, so a syntax error means the browser could not even read the line, because the string starts but never ends. The exact wording depends on the browser, but the message usually points at the spot where it got confused.

Every programmer sees messages like these many times a day. The skill is not avoiding errors; it is reading them calmly. Whenever something goes wrong, the first question is always: *what does the console say?*

> **NOTE**
>
> JavaScript is **case-sensitive**: `console.log` works, but `Console.log` and `console.Log` do not, because capital and small letters count as different characters. If a line looks right but fails, check the capitals.

The Project: A Welcome Board

Rosa has an old laptop connected to a screen in the center's lobby. Right now it shows a photo of last year's picnic. She would like it to show a friendly welcome, today's date (so she no longer has to update a paper sign every morning), and a short list of what is happening this week. That is our first project.

When it is finished, the page will look roughly like this:

```
+------------------------------------------------+
| BRAMBLEWOOD COMMUNITY CENTER                   |
| Welcome to Bramblewood!                        |
| Today is Monday, September 28, 2026.           |
|                                                |
| Happening this week                            |
|  - Coding Club for teens - Thursday, 6:00 PM ... |
|  - Community lunch - Saturday, 12:00 PM ...    |
```

```
|  - Garden work morning - Sunday, 9:00 AM ...     |
|                                                  |
| Questions? Ask Rosa Delgado at the front desk.   |
| Bramblewood Community Center, 2026               |
+--------------------------------------------------+
```

Of course, the date line will show whatever day it is when *you* open the page. That is the whole point: the JavaScript works out the date by itself every time the page loads.

Step 1: Create the project folder

Good habits start on day one. Create a folder somewhere easy to find, such as your Documents folder, and name it `projects`. Every project in this book gets its own folder inside it. Inside `projects`, create a folder named `ch01-welcome-board`.

Now open that folder in VS Code: choose **File**, then **Open Folder**, and select `ch01-welcome-board`. (If VS Code asks whether you trust the files in this folder, you can say yes — you created them.) The Explorer panel on the left shows the folder's contents, which is empty for now. Hover over the folder name and click the **New File** icon three times to create these files:

```
ch01-welcome-board/
  index.html
  style.css
  script.js
```

This three-file pattern — HTML for content, CSS for looks, JavaScript for behavior — is the one we will use for almost every project in the book.

> **TIP**
>
> Use lowercase names without spaces for files and folders, with hyphens between words (`ch01-welcome-board` , not `Chapter 1 Welcome Board`). Web addresses treat spaces and capitals awkwardly, and some servers see `Script.js` and `script.js` as two different files. Simple names avoid a whole family of confusing problems.

Step 2: Write the HTML page

We need just enough HTML to hold our content. (Chapter 11 covers HTML and CSS in more depth; here we only need the basics.) Open `index.html` and type the following. Take your time, and do not worry about memorizing it — you will see this structure so often that it will soon feel familiar.

`projects/ch01-welcome-board/index.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
```

```
  <title>Welcome Board - Bramblewood Community Center</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <main class="board">
    <p class="eyebrow">Bramblewood Community Center</p>
    <h1 id="greeting">Welcome!</h1>
    <p id="today">Loading today's date...</p>

    <h2>Happening this week</h2>
    <ul>
      <li>Coding Club for teens - Thursday, 6:00 PM, Room 2</li>
      <li>Community lunch - Saturday, 12:00 PM, Main Hall</li>
      <li>Garden work morning - Sunday, 9:00 AM, Community Garden</li>
    </ul>

    <p class="contact">Questions? Ask Rosa Delgado at the front desk.</p>
    <p class="footer" id="footer">Bramblewood Community Center</p>
  </main>

  <script src="script.js"></script>
</body>
</html>
```

Here is what the important parts mean:

- `<!DOCTYPE html>` tells the browser this is a modern HTML page.
- HTML is made of **elements**, written with **tags** in angle brackets. Most elements have an opening tag like `<h1>` and a closing tag like `</h1>`, with content in between.
- The `<head>` holds information *about* the page: the character set, the title shown on the browser tab, and a `<link>` to our stylesheet.
- The `<body>` holds everything visible: headings (`<h1>`, `<h2>`), paragraphs (`<p>`) and a list (`<ul>` with `<li>` items).
- Some elements have an **id**, such as `id="greeting"` and `id="today"`. An id is a unique name tag. JavaScript will use these names to find exactly the elements it should change.
- Some have a **class**, such as `class="board"`. Classes are labels that CSS uses for styling, and many elements can share one.
- Finally, just before `</body>`, the `<script>` element loads our JavaScript file.

Notice the placeholder text: the heading says "Welcome!" and the date paragraph says "Loading today's date...". If our JavaScript ever fails, the visitor still sees sensible text. When the script works, it replaces those placeholders instantly.

Why the script goes at the end

The browser reads an HTML file from top to bottom, building the page as it goes. When it reaches a `<script>` element, it stops, loads the script and runs it immediately. If our script sat at the top of the page, it would run before the heading and the date paragraph even existed, and it would not be able to

find them. By placing the `<script>` element at the very end of the body, we make sure every element is already built when our code runs.

Step 3: Add a little style

A lobby screen should look friendly. Open `style.css` and add these rules. Each rule starts with a **selector** that says which elements to style (`body` means the body element, `.board` means elements with class `board`, and `#today` means the element with id `today`), followed by property and value pairs inside curly braces.

projects/ch01-welcome-board/style.css

```
body {
  margin: 0;
  font-family: Arial, Helvetica, sans-serif;
  background-color: #f3efe6;
  color: #2d2a26;
}

.board {
  max-width: 600px;
  margin: 40px auto;
  padding: 24px 32px;
  background-color: #ffffff;
  border-top: 8px solid #3f7d4e;
  border-radius: 8px;
}

.eyebrow {
  margin: 0;
  color: #3f7d4e;
  font-weight: bold;
  text-transform: uppercase;
  letter-spacing: 1px;
}

h1 {
  margin: 8px 0;
  font-size: 2.2rem;
}

#today {
  font-size: 1.2rem;
  color: #6b5d4f;
}

.contact {
  font-style: italic;
}

.footer {
  margin-top: 32px;
```

```
  font-size: 0.85rem;
  color: #8a7d70;
}
```

You do not need to understand every property yet. Colors are written as hexadecimal codes (`#3f7d4e` is a leafy green), `max-width` keeps the board from stretching across a wide screen, and `margin: 40px auto` centers it.

Step 4: Connect the script and say hello

Now for the JavaScript. Start small, with just one line in `script.js`, to prove that the page and the script are connected:

`projects/ch01-welcome-board/script.js`

```
console.log("The Welcome Board script has started.");
```

Save the file. Then open the page in your browser: find `index.html` in your file manager and double-click it (or drag it onto a browser window). The address bar will show a path starting with `file://`, which means the browser is opening a file straight from your computer rather than from the internet. That works perfectly for the projects in the early chapters.

You should see the page with its placeholder text. Now open the console with the shortcut from earlier. You should see:

`Console output`

```
The Welcome Board script has started.
```

If you see that message, your HTML, your script and your browser are all talking to each other. That connection is the foundation of everything that follows. If you do not see it, check the troubleshooting list later in this chapter before moving on.

From now on, the working loop is always the same: **edit** the file in VS Code, **save** (or let Auto Save do it), switch to the browser and **reload** the page (F5, or Cmd+R on macOS), then **look** at both the page and the console.

Step 5: Ask JavaScript for today's date

JavaScript has a built-in tool for dates called `Date`. Writing `new Date()` creates a date value holding the current moment, taken from your computer's clock. Try it in the console first:

```
new Date()
```

The console shows a long line with the day, the date, the time and a time zone. It is accurate, but not very friendly for a lobby screen. Browsers can format a date in a readable way with a function called `toLocaleDateString`. Add these lines to `script.js`, below the first message:

```
// 1. Get today's date and turn it into friendly words.
const today = new Date();
const dateText = today.toLocaleDateString("en-US", {
  weekday: "long",
  year: "numeric",
  month: "long",
  day: "numeric"
});

console.log("Today is " + dateText);
```

There are several new ideas here, so let us go line by line.

The first line starts with `//`. That makes it a **comment**: a note for humans that the browser ignores completely. Comments are a great way to explain *why* your code does something, and we will use them generously.

The line `const today = new Date();` creates a date and stores it under the name `today`. The word `const` creates a **variable** — a labeled box that holds a value so we can use it later by name. Chapter 2 is all about variables, so for now, read `const today = ...` as "let's call this value `today`".

Then we call `today.toLocaleDateString(...)`, which means "ask the `today` date to describe itself as text". We hand it two pieces of information, separated by a comma:

- `"en-US"` is a **locale**: a code for a language and region. It tells the browser to use American English words and ordering.
- The part in curly braces is a set of **options** saying which parts of the date to include and how long each should be.

The result is a string, and we store it in a second variable, `dateText`. Finally, `"Today is " + dateText` joins two strings, and `console.log` prints the result. Save, reload and check the console. If today were Monday, September 28, 2026, you would see:

Console output

```
The Welcome Board script has started.
Today is Monday, September 28, 2026
```

Here are the option values that are most useful for a welcome board. Try changing them to see what happens:

Option	Example values	Sample result
`weekday`	"long", "short"	Monday, Mon

`month`	"long", "short", "numeric"	September, Sep, 9
`day`	"numeric", "2-digit"	28, 28
`year`	"numeric", "2-digit"	2026, 26

If you leave out an option, that part is left out of the result. For example, `{ month: "short", day: "numeric" }` produces something like `Sep 28`.

Step 6: Put the greeting and date on the page

Messages in the console are great for us, but visitors to the lobby will never open DevTools. We need to change the page itself. Add these lines:

```
// 2. Put a greeting and the date on the page.
document.querySelector("#greeting").textContent = "Welcome to Bramblewood!";
document.querySelector("#today").textContent = "Today is " + dateText + ".";
```

This is a small preview of something we will study properly in Part III, so here is just enough to understand it:

- `document` is the browser's representation of the whole page. JavaScript can read and change the page through it.
- `querySelector("#greeting")` finds the first element that matches a selector. The `#` means "the element with this id", exactly like in CSS. So this finds our `<h1 id="greeting">`.
- `.textContent` is the text inside that element.
- The single `=` sign means "set this to". So the whole line reads: "find the element with id greeting and set its text to Welcome to Bramblewood!".

Save and reload. The placeholder "Welcome!" becomes "Welcome to Bramblewood!", and "Loading today's date..." becomes the real date. You have just made JavaScript change a web page.

> **NOTE**
>
> The id in the JavaScript must match the id in the HTML exactly, including capitals, and the JavaScript needs the `#` in front while the HTML does not. In HTML you write `id="today"`; in JavaScript you write `"#today"`.

Step 7: Keep the footer up to date

Paper signs go out of date, and so do footers that say "2019". Let us have JavaScript fill in the current year. A date value can tell us its year with `getFullYear()`:

```
// 3. Keep the footer year up to date automatically.
const year = today.getFullYear();
document.querySelector("#footer").textContent =
```