



learn
enough

JAVASCRIPT

TO BE DANGEROUS



MICHAEL HARTL

FREE SAMPLE CHAPTER



Praise for Learn Enough Tutorials

“I have nothing but fantastic things to say about @LearnEnough courses. I am just about finished with the #javascript course. I must say, the videos are mandatory because @mhartl will play the novice and share in the joy of having something you wrote actually work!”

—Claudia Vizena

“I must say, this Learn Enough series is a masterpiece of education. Thank you for this incredible work!”

—Michael King

“I want to thank you for the amazing job you have done with the tutorials. They are likely the best tutorials I have ever read.”

—Pedro Iatzky

This page intentionally left blank

LEARN ENOUGH
JAVASCRIPT
TO BE DANGEROUS

Learn Enough Series from Michael Hartl



Visit informit.com/learn-enough for a complete list of available publications.

The **Learn Enough** series teaches you the developer tools, Web technologies, and programming skills needed to launch your own applications, get a job as a programmer, and maybe even start a company of your own. Along the way, you'll learn technical sophistication, which is the ability to solve technical problems yourself. And Learn Enough always focuses on the most important parts of each subject, so you don't have to learn everything to get started—you just have to learn enough to be dangerous. The Learn Enough series includes books and video courses so you get to choose the learning style that works best for you.

LEARN ENOUGH JAVASCRIPT TO BE DANGEROUS

Write Programs, Publish Packages, and
Develop Interactive Websites with JavaScript

Michael Hartl

◆◆ Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town

Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City

São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Cover image: Philipp Tur/Shutterstock
Figures 1.5-1.9, 4.10, 10.5, 11.2-11.4: GitHub, Inc.
Figure 2.7: Replit, Inc.
Figures 2.14, 3.1, 5.9, 6.4: Courtesy of Mike Vanier
Figures 4.4, 4.5, 4.11, 7.6, 8.5: Regex101
Figures 5.4, 10.4: Google
Figure 7.4: Courtesy of David Heinemeier Hansson
Figure 8.2: OpenJS Foundation
Figure 10.2: Amazon Web Services, Inc.
Figures 10.3, 10.6: Wikimedia Foundation, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2022933200

Copyright © 2022 Softcover Inc.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearson.com/permissions.

ISBN-13: 978-0-13-784374-9

ISBN-10: 0-13-784374-7

ScoutAutomatedPrintCode

Contents

Preface	xiii
About the Author	xvii

Chapter 1 Hello, World! 1

1.1	Introduction to JavaScript	5
1.2	JS in a Web Browser	7
	1.2.1	Deployment 10
	1.2.2	Exercise 13
1.3	JS in a REPL	14
	1.3.1	Browser Console 14
	1.3.2	Node Prompt 19
	1.3.3	Exercise 20
1.4	JS in a File	21
	1.4.1	Exercise 22
1.5	JS in a Shell Script	22
	1.5.1	Exercise 24

Chapter 2 Strings 25

- 2.1 String Basics 25
 - 2.1.1 Exercise 27
- 2.2 Concatenation and Interpolation 27
 - 2.2.1 The Backtick Syntax 31
 - 2.2.2 Exercises 32
- 2.3 Printing 33
 - 2.3.1 Exercise 34
- 2.4 Properties, Booleans, and Control Flow 35
 - 2.4.1 Combining and Inverting Booleans 40
 - 2.4.2 Bang Bang 43
 - 2.4.3 Exercises 44
- 2.5 Methods 44
 - 2.5.1 Exercises 49
- 2.6 String Iteration 50
 - 2.6.1 Exercises 53

Chapter 3 Arrays 55

- 3.1 Splitting 55
 - 3.1.1 Exercises 56
- 3.2 Array Access 56
 - 3.2.1 Exercises 58
- 3.3 Array Slicing 58
 - 3.3.1 Exercises 59
- 3.4 More Array Methods 59
 - 3.4.1 Sorting and Reversing 60
 - 3.4.2 Pushing and Popping 61
 - 3.4.3 Undoing a Split 61
 - 3.4.4 Exercises 62
- 3.5 Array Iteration 62
 - 3.5.1 Exercises 64

Chapter 4 Other Native Objects 65

- 4.1 Math and Number 65
 - 4.1.1 More Advanced Operations 66

- 4.1.2 Math to String 67
- 4.1.3 Exercises 69
- 4.2 Dates 69
 - 4.2.1 Exercises 73
- 4.3 Regular Expressions 73
 - 4.3.1 Regex Methods 75
 - 4.3.2 String Methods 77
 - 4.3.3 Exercises 80
- 4.4 Plain Objects 81
 - 4.4.1 Exercise 83
- 4.5 Application: Unique Words 83
 - 4.5.1 Map 87
 - 4.5.2 Exercises 89

Chapter 5 Functions 91

- 5.1 Function Definitions 91
 - 5.1.1 Sorting Numerical Arrays 92
 - 5.1.2 Fat Arrow 94
 - 5.1.3 Exercise 95
- 5.2 Functions in a File 95
 - 5.2.1 Exercises 103
- 5.3 Method Chaining 104
 - 5.3.1 Caveat Emoji 108
 - 5.3.2 Exercises 109
- 5.4 Iteration for Each 110
 - 5.4.1 Exercises 113

Chapter 6 Functional Programming 115

- 6.1 Map 116
 - 6.1.1 Exercise 122
- 6.2 Filter 122
 - 6.2.1 Exercise 125
- 6.3 Reduce 126
 - 6.3.1 Reduce, Example 1 126
 - 6.3.2 Reduce, Example 2 129

6.3.3 Functional Programming and TDD 132

6.3.4 Exercises 133

Chapter 7 Objects and Prototypes 135

7.1 Defining Objects 135

7.1.1 Exercise 139

7.2 Prototypes 139

7.2.1 Exercise 145

7.3 Modifying Native Objects 147

7.3.1 Exercises 152

Chapter 8 Testing and Test-Driven Development 153

8.1 Testing Setup 154

8.1.1 Exercise 159

8.2 Initial Test Coverage 159

8.2.1 Pending Tests 162

8.2.2 Exercises 163

8.3 Red 164

8.3.1 Exercises 171

8.4 Green 172

8.4.1 Exercise 177

8.5 Refactor 177

8.5.1 Publishing the NPM Module 184

8.5.2 Exercises 185

Chapter 9 Events and DOM Manipulation 187

9.1 A Working Palindrome Page 187

9.1.1 Exercise 191

9.2 Event Listeners 192

9.2.1 Exercise 200

9.3 Dynamic HTML 202

9.3.1 Exercise 205

9.4 Form Handling 205

9.4.1 Exercises 210

Chapter 10 Shell Scripts with Node.js 215

- 10.1 Reading from Files 216
 - 10.1.1 Exercise 218
- 10.2 Reading from URLs 218
 - 10.2.1 Exercise 223
- 10.3 DOM Manipulation at the Command Line 224
 - 10.3.1 Exercises 233

Chapter 11 Full Sample App: Image Gallery 235

- 11.1 Prepping the Gallery 235
 - 11.1.1 Prepping the JavaScript 239
 - 11.1.2 Exercise 241
- 11.2 Changing the Gallery Image 242
 - 11.2.1 Exercises 246
- 11.3 Setting an Image as Current 250
 - 11.3.1 Exercise 252
- 11.4 Changing the Image Info 252
 - 11.4.1 Deploying 256
 - 11.4.2 Exercise 257
- 11.5 Conclusion 259
 - 11.5.1 Learning More JavaScript 260
 - 11.5.2 Learning a New Language 261

Index 263

This page intentionally left blank

Preface

Learn Enough JavaScript to Be Dangerous is designed to get you started writing practical and modern JavaScript programs as quickly as possible, using the latest JavaScript technologies and with a focus on the real tools used every day by software developers. JavaScript is a big language with correspondingly enormous tutorials. The good news, though, is that you don't have to learn everything to get started . . . you just have to learn enough to be *dangerous*.

Unlike most JavaScript tutorials, *Learn Enough JavaScript to Be Dangerous* treats JavaScript as a *general-purpose* programming language right from the start, so our examples won't be confined to the browser. In addition to interactive HTML websites, you'll learn how to write command-line programs and self-contained JavaScript packages as well. We'll even have a chance to explore important software development practices like version control, functional programming, and test-driven development. The result is a practical narrative introduction to JavaScript—a perfect complement both to in-browser coding tutorials and to the voluminous but hard-to-navigate JavaScript reference materials on the Web.

In addition to teaching you specific skills, *Learn Enough JavaScript to Be Dangerous* also helps you develop *technical sophistication*—the seemingly magical ability to solve practically any technical problem. Technical sophistication includes concrete skills like version control and HTML, as well as fuzzier skills like Googling the error message and knowing when to just reboot the darn thing. Throughout this book, we'll have abundant opportunities to develop technical sophistication in the context of real-world examples.

Chapter by Chapter

Chapter 1 begins at the beginning with a series of simple “hello, world” programs using several different techniques, including an “alert” in the browser and a command-line shell script using *Node.js*, a fast and widely used execution environment for JavaScript programs. We’ll even deploy a (very simple) dynamic JavaScript application to the live Web.

The next three chapters cover some of the most important JavaScript data structures. Chapter 2 covers strings, Chapter 3 covers arrays, and Chapter 4 covers other native objects like numbers, dates, and regular expressions. Taken together, these chapters constitute a gentle introduction to *object-oriented programming* with JavaScript.

In Chapter 5, you’ll learn the basics of *functions*, an essential subject for virtually every programming language. Chapter 6 then applies this knowledge to an elegant and powerful style of coding known as *functional programming*.

Chapter 7 shows how to make custom JavaScript objects using the example of palindromes (which read the same forward and backward). We’ll start off with the simplest palindrome definition possible, and then we’ll extend it significantly in Chapter 8 using a powerful programming technique known as *test-driven development*. In the process, you’ll learn how to create and publish a self-contained JavaScript software package called an *NPM module*.

Chapter 9 builds on the palindrome module to make a live website for detecting palindromes. In the process, we’ll learn about *events*, *DOM manipulation*, *alerts*, *prompts*, and an example of an HTML *form*.

Chapter 10 covers the much-neglected topic of *shell scripts* using JavaScript. You’ll learn how to read text both from local files and from live URLs. You’ll also learn how to extract information from a regular text file as if it were an HTML web page.

Chapter 11 completes the tutorial by showing you how to create a real, industrial-grade website using HTML, CSS, and JavaScript. The result is an interactive image gallery that dynamically changes images, CSS classes, and page text in response to user clicks. We’ll conclude by deploying the full sample website to the live Web.

Additional Features

In addition to the main tutorial material, *Learn Enough JavaScript to Be Dangerous* includes a large number of exercises to help you test your understanding and to extend the material in the main text. The exercises include frequent hints and often include the expected answers, with community solutions available by separate subscription at www.learnenough.com.

Final Thoughts

Learn Enough JavaScript to Be Dangerous gives you a practical introduction to the fundamentals of JavaScript, both in its original niche of the web browser and as a general-purpose programming language. After learning the techniques covered in this tutorial, and especially after developing your technical sophistication, you'll know everything you need to write shell scripts, publish Node packages, and design and deploy interactive websites with JavaScript. You'll also be ready for a huge variety of other resources, including books, blog posts, and online documentation. A particularly good next step is learning how to make dynamic database-backed web applications, as covered in *Learn Enough Ruby to Be Dangerous* and the *Ruby on Rails™ Tutorial*.

Learn Enough Scholarships

Learn Enough is committed to making a technical education available to as wide a variety of people as possible. As part of this commitment, in 2016 we created the Learn Enough Scholarship program (<https://www.learnenough.com/scholarship>). Scholarship recipients get free or deeply discounted access to the Learn Enough All Access subscription, which includes all of the Learn Enough online book content, embedded videos, exercises, and community exercise answers.

As noted in a 2019 RailsConf Lightning Talk (<https://youtu.be/AI5wmmzzBqc?t=1076>), the Learn Enough Scholarship application process is incredibly simple: just fill out a confidential text area telling us a little about your situation. The scholarship criteria are generous and flexible—we understand that there are an enormous number of reasons for wanting a scholarship, from being a student, to being between jobs, to living in a country with an unfavorable exchange rate against the U.S. dollar. Chances are that, if you feel like you've got a good reason, we'll think so, too.

So far, Learn Enough has awarded more than 2,500 scholarships to aspiring developers around the country and around the world. To apply, visit the Learn Enough Scholarship page at www.learnenough.com/scholarship. Maybe the next scholarship recipient could be you!

Register your copy of *Learn Enough JavaScript to Be Dangerous* on the InformIT site for convenient access to updates and/or corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN (9780137843749) and click Submit. Look on the Registered Products tab for an Access Bonus Content link next to this product, and follow that link to access any available bonus materials. If you would like to be notified of exclusive offers on new editions and updates, please check the box to receive email from us.

This page intentionally left blank

About the Author

Michael Hartl (www.michaelhartl.com) is the creator of the *Ruby on Rails™ Tutorial* (www.railstutorial.org), one of the leading introductions to web development, and is cofounder and principal author at Learn Enough (www.learnenough.com). Previously, he was a physics instructor at the California Institute of Technology (Caltech), where he received a Lifetime Achievement Award for Excellence in Teaching. He is a graduate of Harvard College, has a Ph.D. in Physics from Caltech, and is an alumnus of the Y Combinator entrepreneur program.

This page intentionally left blank

CHAPTER 11

Full Sample App: Image Gallery

As a final application of our newfound JavaScript powers, in this last chapter we'll build on the sample application developed in *Learn Enough CSS & Layout to Be Dangerous* (<https://www.learnenough.com/css-and-layout>). (We'll be *cloning* the initial sample repository, so you'll be able to complete this chapter even if you didn't follow the CSS tutorial.) In particular, we'll follow a time-honored tradition in JavaScript tutorials and create an *image gallery*, which will allow us to display and swap custom images—in our case, a fancy three-column layout (<https://www.learnenough.com/css-and-layout-tutorial/flex-intro#sec-pages-3col>).

After prepping the gallery (Section 11.1), we'll learn how to change the gallery image (Section 11.2), set an image as “current” (Section 11.3), and change the image title and description (Section 11.4). Because our starting point is the professional-grade website developed in *Learn Enough CSS & Layout to Be Dangerous*, the result is unusually polished for a JavaScript tutorial sample gallery (Figure 11.1).

11.1 Prepping the Gallery

To get started with our image gallery, you'll need to get a copy of the full starting application (https://github.com/learnenough/le_js_full) for the site. The first step is to make a personal copy of the app, which you can do using the *fork* capability at GitHub (Figure 11.2).

The next step depends on whether or not you currently have a GitHub Pages site at `<username>.github.io`. If you don't have such a repository, you can rename

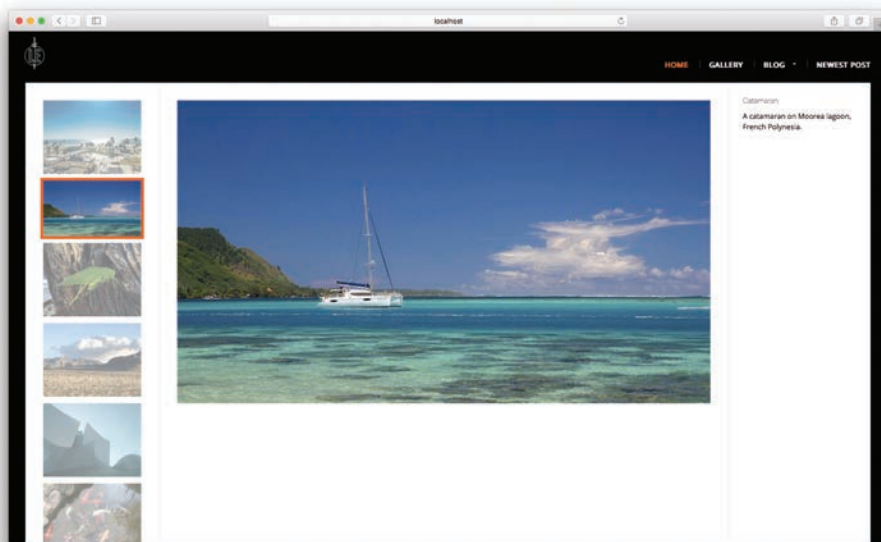


Figure 11.1: This is the gallery we're looking for.

your app accordingly (Figure 11.3), and it will automatically be available at the URL `<username>.github.io`.

Once you've renamed the repo, you can clone the gallery app to your local system using the clone URL from GitHub (Figure 11.4):

```
$ git clone <clone URL> <username>.github.io
```

If you already have a repository at `<username>.github.io` from following *Learn Enough CSS & Layout to Be Dangerous*, you should clone the gallery app (without renaming it) to the default directory by omitting the second argument to **git clone**:

```
$ git clone <clone URL> # Command if you already have <username>.github.io
```

This will create a local repository called `le_js_full`, which you can use as a reference for copying over the required files. In particular, you'll need the gallery `index.html` and the large and small images:

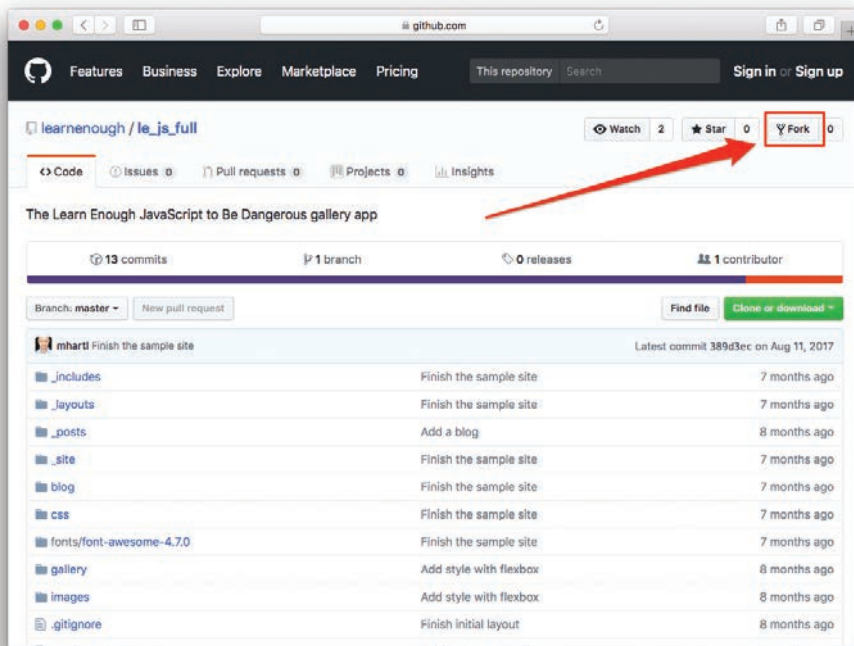


Figure 11.2: Forking the starting application at GitHub.

```
# Run these commands only if you already have <username>.github.io
# from following Learn Enough CSS & Layout to Be Dangerous.
$ cd le_js_full/
$ cp gallery/index.html /path/to/repo/<username>.github.io/gallery/
$ cp -r images/* /path/to/repo/<username>.github.io/images/
```

(If you already have a repo at <username>.github.io that *isn't* the result of following *Learn Enough CSS & Layout to Be Dangerous*, I'll assume you have the requisite technical sophistication to figure something out on your own.)

In either case, once the app is put together you can run it using the Jekyll static site builder. The Jekyll setup instructions (<https://www.learnenough.com/css-and-layout-tutorial/struct-layout#sec-jekyll>) in *Learn Enough CSS & Layout to Be Dangerous* explain how to install Jekyll on your system in case it isn't installed already. The short version is that you first need to install *Bundler*:

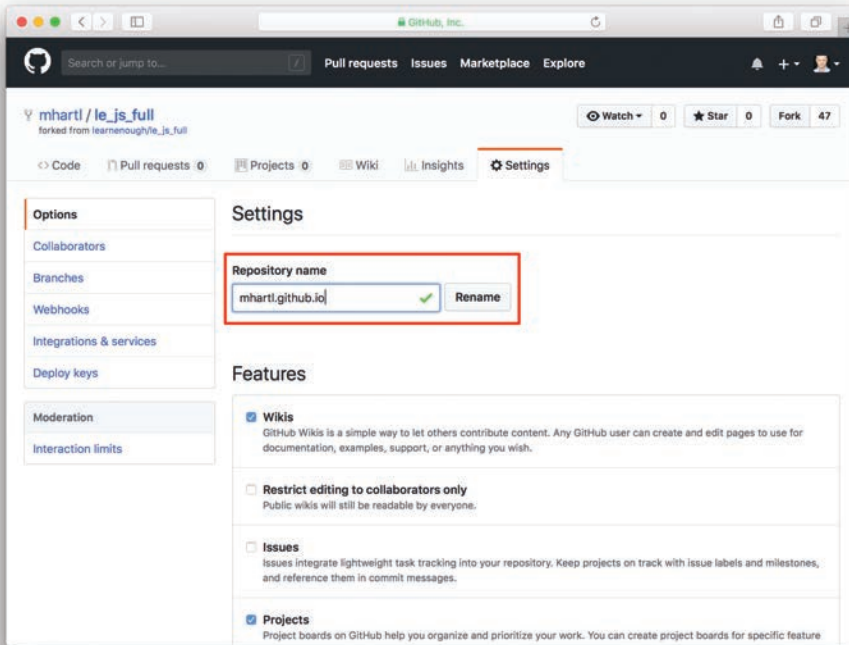


Figure 11.3: Renaming to the default GitHub Pages name.

```
$ gem install bundler -v 2.2.17
```

Then use the **bundle** command to install the **jekyll** gem listed in the **Gemfile** included with the repository:

```
$ bundle _2.2.17_ install
```

Once Jekyll is installed, you can serve the sample website by using Bundler to execute the correct version of the **jekyll** program:

```
$ bundle exec jekyll serve
```

At this point, the app will be running on localhost:4000, and should look something like Figure 11.5.

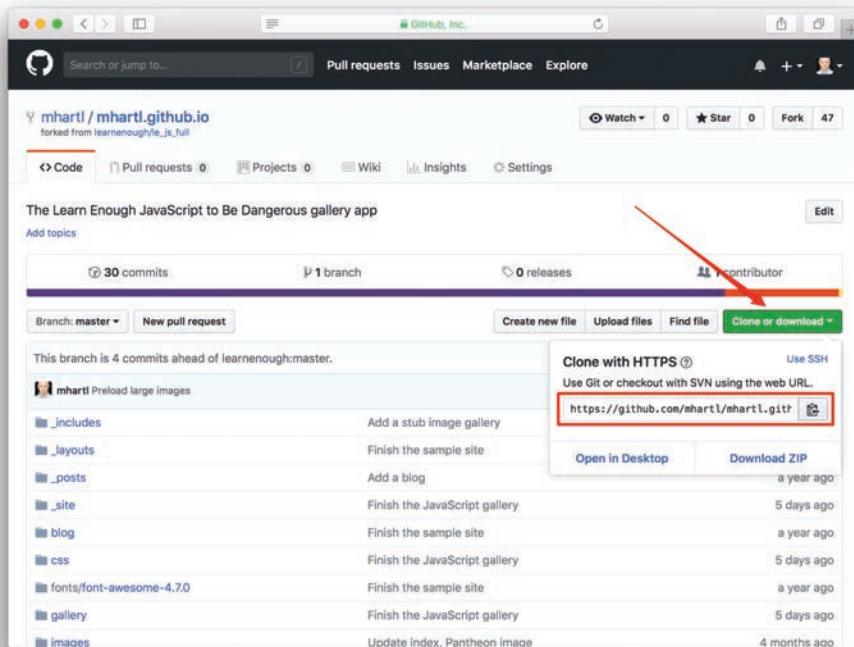


Figure 11.4: Getting the clone URL at GitHub.

11.1.1 Prepping the JavaScript

As a final bit of prep, we'll add a stub for the main gallery function, `activateGallery`, which we'll be filling in throughout the rest of this chapter. Because we'll be doing everything in plain JavaScript, there will be no need to include any Node modules, run `browserify`, etc. In fact, all we'll need to do is write a single function.

Our first step is to make a directory and JavaScript file (remember, this is in the app directory, not `js_tutorial`):

```
$ mkdir js
$ touch js/gallery.js
```

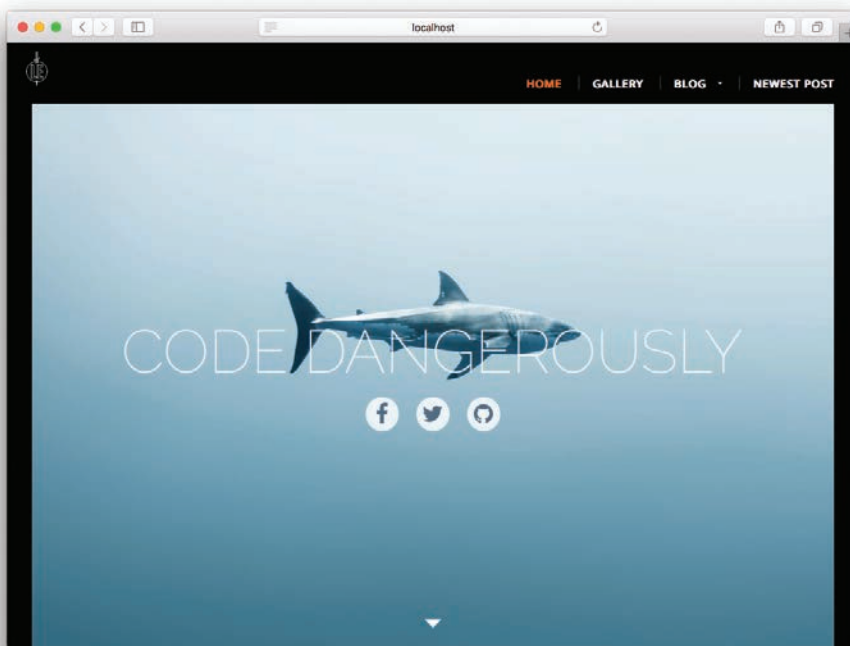



Figure 11.5: Our initial sample app.

Just to get started, we'll add an initial alert to **gallery.js** (Listing 11.1).

Listing 11.1: A stub gallery file.

js/gallery.js

```
function activateGallery() {  
  alert("Hello from the gallery file!");  
}
```

In the head of the file, we'll include the gallery JavaScript using the **src** attribute (Section 5.2), and add an event listener (Section 9.2) to run the gallery activation function automatically after the DOM is loaded (Listing 9.9). The result appears in Listing 11.2.

Listing 11.2: Including the gallery JavaScript.*_includes/head.html*

```
<head>
  .
  .
  .
  <link rel="stylesheet" href="/css/main.css">

  <script src="/js/gallery.js"></script>
  <script>
    document.addEventListener("DOMContentLoaded", function() {
      activateGallery();
    });
  </script>
</head>
```

Now visiting the local gallery page confirms that the JavaScript was loaded correctly (Figure 11.6).

11.1.2 Exercise

1. Deploy your stub gallery to GitHub Pages and confirm that it works in production.

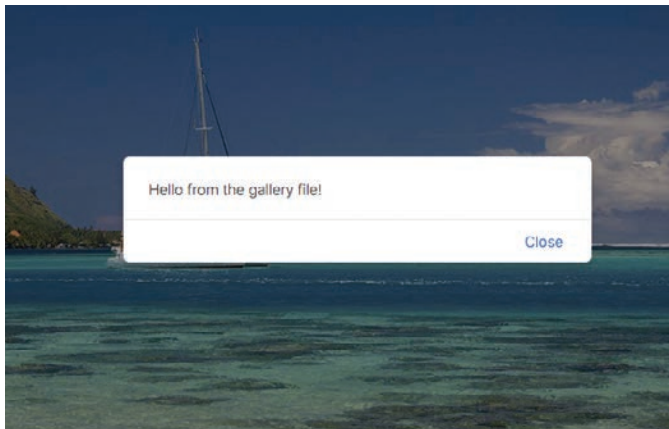


Figure 11.6: Hello from the gallery!

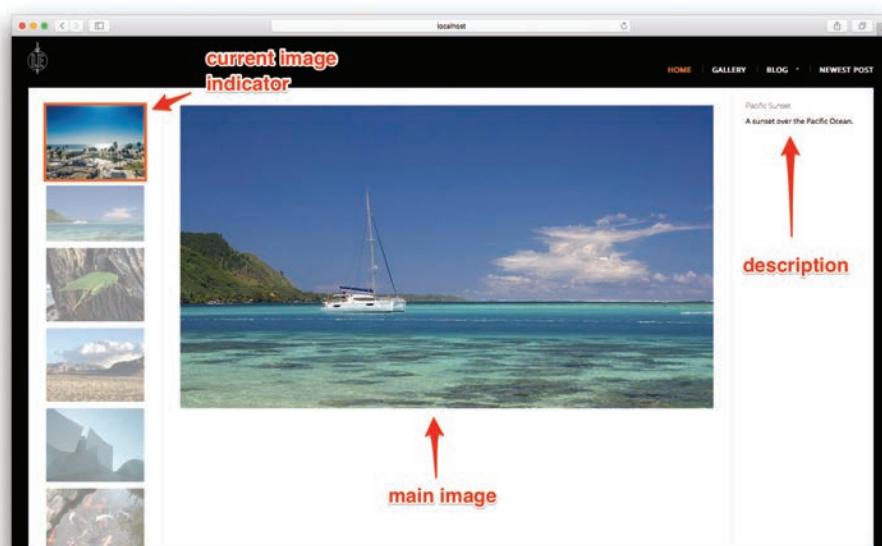


Figure 11.7: The initial gallery.

11.2 Changing the Gallery Image

Let's take a look at the current state of the application. The gallery page has three columns: one with smaller “thumbnail” images, one with the main image, and one with the description. As seen in Figure 11.7, in the default state the “current image” indicator in the thumbnails doesn't match the main image, and the description doesn't match either.

We can see the origins of this mismatch by taking a look at the current HTML structure of the gallery, which appears as in Listing 11.3.

Listing 11.3: The gallery HTML.

gallery/index.html

```

1 ---
2 layout: default
3 title: Gallery for Learn Enough JavaScript to Be Dangerous
4 ---
5
6 <div class="gallery col-three">
```

```
7 <div class="col col-nav gallery-thumbs" id="gallery-thumbs">
8 <div class="current">
9 
13 </div>
14 .
15 .
16 .
17 <div>
18 
22 </div>
23 </div>
24 <div class="col col-content">
25 <div class="gallery-photo" id="gallery-photo">
26 
27 </div>
28 </div>
29 <div class="col col-aside gallery-info" id="gallery-info">
30 <h3 class="title">Pacific Sunset</h3>
31 <p class="description">A sunset over the Pacific Ocean.</p>
32 </div>
33 </div>
```

From Listing 11.3, we see that the current image is indicated with a CSS class **current** (Line 8), the main image is in an HTML **div** with CSS id **gallery-photo** (Line 25), and the title and description are in a div with CSS id **gallery-info** (Line 29). Our task is to dynamically update this HTML (Section 9.3) so that all three columns match.

Our first task is the biggest one in terms of the user interface, namely, swapping out the main image when the user clicks on a thumbnail. Our strategy is to put an event listener (Section 9.2) on each image, and then change the source (**src**) of the main display image on click.

To do this, we'll first create a variable with a list of all the images.¹ Inspecting the HTML source in Listing 11.3, we see that the thumbnail images are all **img** tags inside

1. As noted briefly in Section 10.3, technically the result of **querySelectorAll** is a “NodeList” object, not an array, but we can treat it as an array for the purposes of iteration. Specifically, we can traverse its elements using the **forEach** method.

a **div** with CSS id **gallery-thumbs**. As a result, we can select all the thumbnails using method chaining (Section 5.3) by combining **querySelector** (Section 9.2) to select the thumbnail div and **querySelectorAll** (Section 10.3) to select all the images:

```
let thumbnails = document.querySelector("#gallery-thumbs").
    querySelectorAll("img");
```

Note that JavaScript allows us to break method calls across lines in order to make the structure clearer and avoid breaking the 80-character limit (Box 2.3).

By iterating through the collection of **thumbnails**, we can put an event listener on each one using code like this:

```
thumbnails.forEach(function(thumbnail) {
  thumbnail.addEventListener("click", function() {
    // code to set clicked image as main image
  });
});
```

This arranges to listen for the same “click” event we saw in Listing 9.13.

As indicated in the JavaScript comment in the middle of the code sample, the body of the listener should set the clicked image as the main image. The way we’ll do this is to set the **src** attribute of the current display image to the “large” version of the image clicked. Referring to Listing 11.3, we see that the main image is inside a **div** with CSS id **gallery-photo**, so we can select it by chaining **querySelector**:

```
let mainImage = document.querySelector("#gallery-photo").
    querySelector("img");
```

In fact, **querySelector** is smart enough to let us combine this into a single command:

```
let mainImage = document.querySelector("#gallery-photo img");
```

It’s worth noting that there’s an equivalent alternate notation that uses an angle bracket **>** to emphasize the nesting relationship between the elements (in this case, an **img** element nested inside an element with CSS id **gallery-photo**):

```
let mainImage = document.querySelector("#gallery-photo > img");
```

We’ll use this alternate notation with **querySelectorAll** in Section 11.2.1.

Once we have the main image, we can use the `setAttribute` method (javascript dom set attribute src) to change its `src` attribute:

```
mainImage.setAttribute("src", newImageSrc);
```

If you've been following along closely, you're now aware that everything we need has been created except for `newImageSrc`, the source of the new image. Happily, the sample app has already arranged to encode the necessary path in the image tag itself. Suppose for the sake of argument that we clicked on the Pacific sunset image, whose HTML looks like this:

```
<div>
  
</div>
```

Encoding data in a tag like this is an essential aspect of *unobtrusive JavaScript*, which involves never putting JavaScript in the body of the HTML itself. When using these data attributes on HTML tags, the browser automatically creates a special `dataset` attribute, whose values correspond to the HTML source as follows:

```
data-large-version -> thumbnail.dataset.largeVersion
data-title         -> thumbnail.dataset.title
data-description   -> thumbnail.dataset.description
```

In general, the data tag `data-foo-bar-baz` on HTML element `object` corresponds to the variable `object.dataset.fooBarBaz`, where the final attribute is in CamelCase (Figure 2.3).

We now have everything we need to replace the main image with the clicked image. If you'd like to give it a go on your own, it makes for an excellent exercise. As usual, use the debugging console (Box 5.1) if you run into trouble. The answer appears in Listing 11.4.

Listing 11.4: Setting the main gallery image.

js/gallery.js

```
// Activates the image gallery.
// The main task is to attach an event listener to each image in the gallery
// and respond appropriately on click.
```

```

function activateGallery() {
  let thumbnails = document.querySelector("#gallery-thumbs").
    querySelectorAll("img");
  let mainImage = document.querySelector("#gallery-photo img");

  thumbnails.forEach(function(thumbnail) {
    thumbnail.addEventListener("click", function() {
      // Set clicked image as main image.
      let newImageSrc = thumbnail.dataset.largeVersion;
      mainImage.setAttribute("src", newImageSrc);
    });
  });
}

```

In addition to changing the **src** attribute, we should also change the **alt** attribute of the swapped-in image. Adding this detail is left as an exercise (Section 11.2.1).

Scrolling down and clicking on the Pacific sunset image produces the expected result (Figure 11.8). The agreement with the third-column description, however, is a coincidence, which can be seen by clicking on any other image (Figure 11.9). In addition, the orange “current image” indicator matches the main image in the gallery only if we happen to click on the corresponding thumbnail (Figure 11.10).

11.2.1 Exercises

1. The code in Listing 11.4 swaps in the **src** of the new large image, but unfortunately the **alt** attribute is still the default one from Listing 11.3 (Figure 11.11). Remedy this minor blemish in Listing 11.5 by replacing **FILL_IN** with the proper value. *Hint:* The value of the image **src** for **thumbnail** is given by **thumbnail.src**, so how do you suppose you get the value of **thumbnail**’s **alt** attribute?
2. As hinted in the main text, it’s possible to change the **thumbnails** definition in Listing 11.4 to eliminate method chaining. We begin by noting that the gallery thumbnails are **img** tags inside **div** tags inside an element with CSS id **gallery-thumbs**; conveniently, we can indicate “inside” using the right angle bracket **>**. By replacing **???** in Listing 11.6 with the appropriate tags, show that we can condense the definition of **thumbnails** down to a single line. *Note:* I generally recommend choosing one convention and sticking with it, but for now we’ll leave the arguments of **querySelectorAll** and **querySelector** inconsistent (one with angle brackets, one without) to emphasize that either notation works.

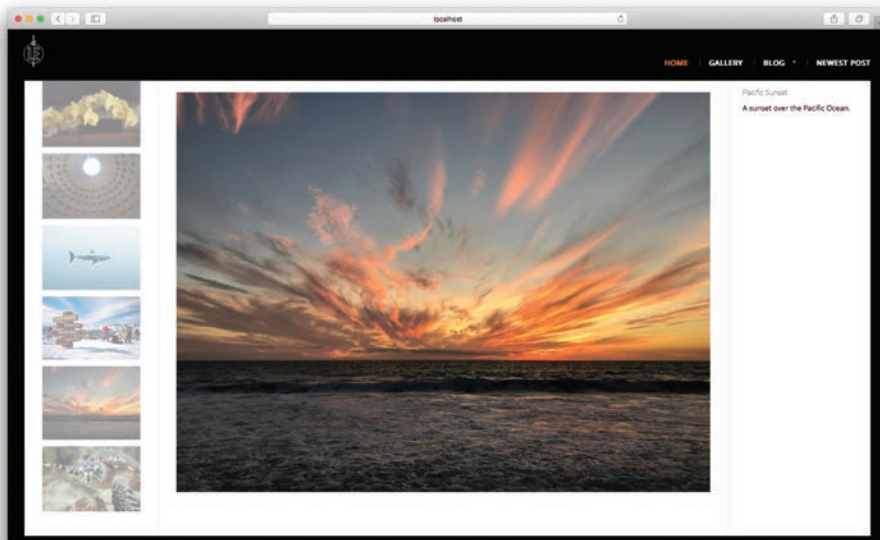


Figure 11.8: A Pacific sunset.

Listing 11.5: Updating the image `alt` attribute.

js/gallery.js

```
// Activates the image gallery.
// The main task is to attach an event listener to each image in the gallery
// and respond appropriately on click.
function activateGallery() {
  let thumbnails = document.querySelector("#gallery-thumbs").
    querySelectorAll("img");
  let mainImage = document.querySelector("#gallery-photo img");

  thumbnails.forEach(function(thumbnail) {
    thumbnail.addEventListener("click", function() {
      // Set clicked image as main image.
      let newImageSrc = thumbnail.dataset.largeVersion;
      mainImage.setAttribute("src", newImageSrc);
      mainImage.setAttribute("alt", FILL_IN);
    });
  });
}
```

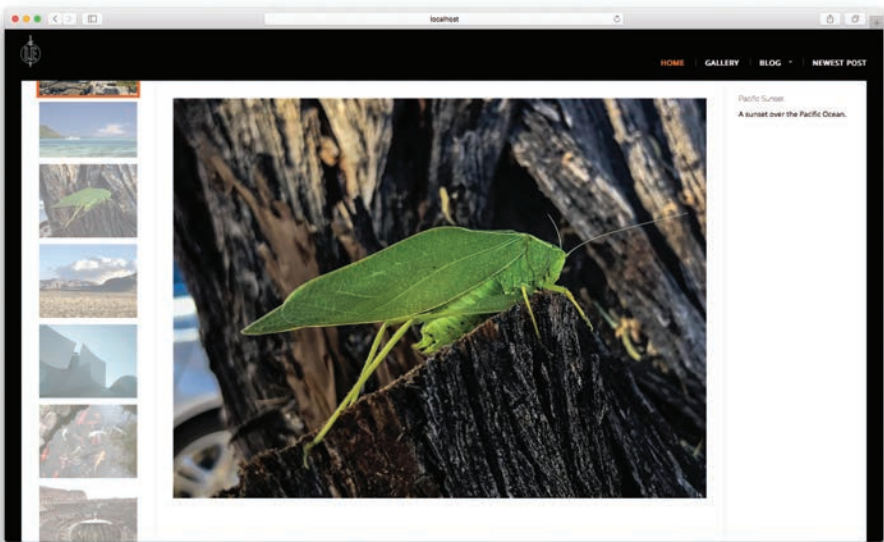


Figure 11.9: The image/description match in Figure 11.8 was a coincidence.

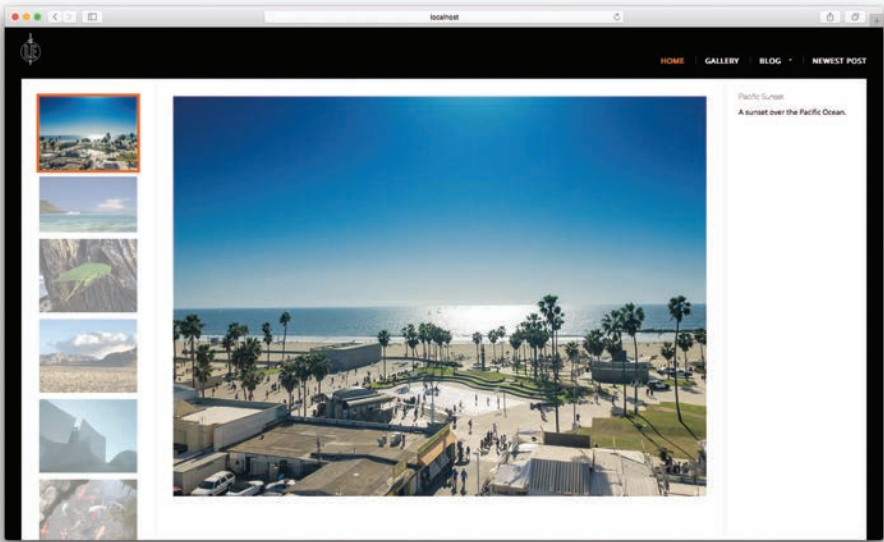


Figure 11.10: The “current image” match here is also a coincidence.

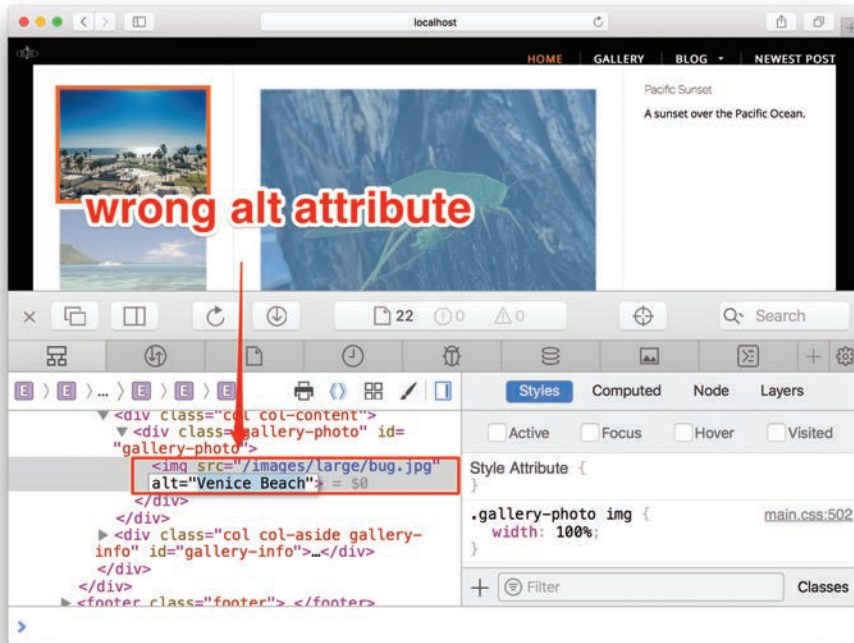


Figure 11.11: The `alt` attribute doesn't match the image `src`.

Listing 11.6: Condensing `thumbnails` into a single line.

`js/gallery.js`

```

// Activates the image gallery.
// The main task is to attach an event listener to each image in the gallery
// and respond appropriately on click.
function activateGallery() {
  let thumbnails = document.querySelectorAll("#gallery-thumbs > ??? > ???");
  let mainImage = document.querySelector("#gallery-photo img");
  .
  .
  .
}

```

11.3 Setting an Image as Current

Section 11.2 represents a major accomplishment: The main task of a photo gallery—namely, swapping the main display image based on a user’s click—is done. All we need to do now is change the “current image” indicator in the first column (this section) and update the image info in the third column (Section 11.4). Both tasks involve a mix of new and old techniques.

As seen in Listing 11.3, the current image is indicated in the HTML source using a CSS class called **current**:

```
<div class="current">
  
</div>
```

This arranges for an orange box shadow due to a line in **main.css**:

```
.
.
.
.gallery-thumbs .current img {
  box-shadow: 0 0 0 5px #ed6e2f;
  opacity: 1;
}
.
.
.
```

Our basic strategy is to add code to the listener in Listing 11.4 that arranges to remove the current image indicator from the thumbnail it’s on and move it to the thumbnail that’s been clicked. This is a little trickier than it looks because the class isn’t on the image—it’s on the **div** surrounding the image. Luckily, JavaScript lets us navigate up and down the DOM with ease, so that we can easily access the DOM element one level up in the tree (Figure 9.6)—the so-called *parent node*.

In short, our algorithm for changing the current image class is as follows:

1. Find the current thumbnail and remove the **current** class.
2. Add the **current** class to the *parent* of the clicked image.

Because there's only one element on the page with class **current**, we can select it using **querySelector**:

```
document.querySelector(".current");
```

But how can we remove the class? Ah: javascript dom remove class. This leads us to the **classList** method and its attendant **remove** method:

```
document.querySelector(".current").classList.remove("current");
```

There's a lot of method chaining here, but its meaning is clear enough.

Happily, once we know how to find the parent node of an element (javascript dom parent node), we can use the corresponding **classList.add** method (javascript dom add class) to add the desired class:

```
thumbnail.parentNode.classList.add("current");
```

Putting these together means we're already done! The result appears in Listing 11.7 (which includes the result of solving the exercise in Section 11.2.1).

Listing 11.7: Changing the current class.

js/gallery.js

```
// Activates the image gallery.
// The main task is to attach an event listener to each image in the gallery
// and respond appropriately on click.
function activateGallery() {
  let thumbnails = document.querySelectorAll("#gallery-thumbs > div > img");
  let mainImage = document.querySelector("#gallery-photo img");

  thumbnails.forEach(function(thumbnail) {
    thumbnail.addEventListener("click", function() {
      // Set clicked image as display image.
      let newImageSrc = thumbnail.dataset.largeVersion;
      mainImage.setAttribute("src", newImageSrc);

      // Change which image is current.
      document.querySelector(".current").classList.remove("current");
      thumbnail.parentNode.classList.add("current");
    });
  });
}
```

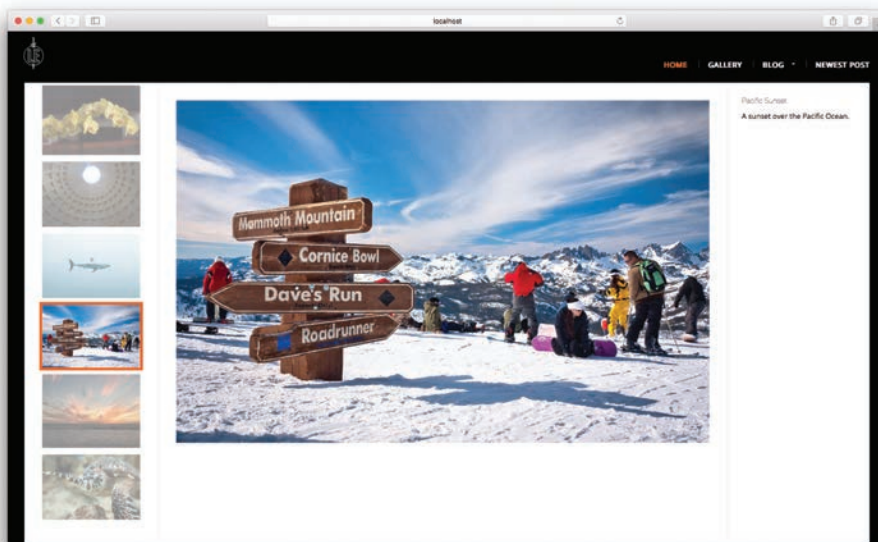


Figure 11.12: Mammoth Mountain.

As a result of the code in Listing 11.7, clicking on a thumbnail automatically updates the current image indicator, whether the image is Mammoth Mountain in the Sierras (Figure 11.12) or The Huntington in San Marino, California (Figure 11.13).

11.3.1 Exercise

1. There's a little duplication in Listing 11.7; in particular, it repeats the string literal **"current"**. Eliminate this duplication by factoring the string into a variable called **currentClass**.

11.4 Changing the Image Info

Our final task is to update the image information (title and description) in the third column of our gallery. Doing this doesn't actually require anything we haven't seen before—we just have to put things we already know together in a slightly new way, making this an excellent way to end the tutorial.

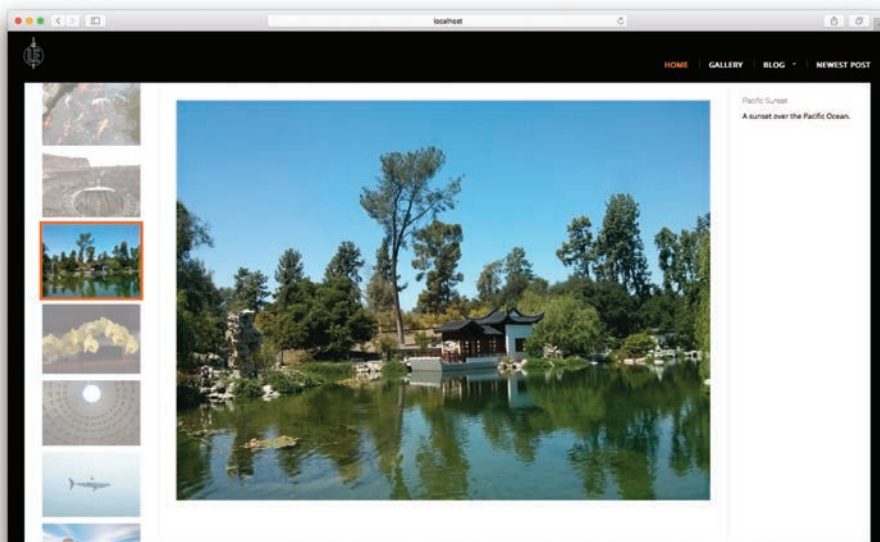


Figure 11.13: The Chinese Garden at The Huntington.

The sequence we'll follow is simple:

1. Find the DOM elements for the image title and description.
2. Replace the contents with the corresponding data from the clicked image.

To find the necessary DOM elements, we first observe that they are both inside the **div** with CSS id **gallery-info**:

```
<div class="col col-aside gallery-info" id="gallery-info">
  <h3 class="title">Pacific Sunset</h3>
  <p class="description">A sunset over the Pacific Ocean.</p>
</div>
```

Inside that **div**, both are the first (and only) elements with the **title** and **description** classes, respectively, which means we can select them as follows:

```
let galleryInfo = document.querySelector("#gallery-info");
let title       = galleryInfo.querySelector(".title");
let description = galleryInfo.querySelector(".description");
```

Note that I've added extra spaces to line up the equals signs, which is a nice (though not strictly necessary) code formatting practice (Box 2.3).

We can get the corresponding values for the clicked image using the **dataset** variable introduced in Section 11.2:

```
thumbnail.dataset.title
```

for the title and

```
thumbnail.dataset.description
```

for the description.

The final piece of the puzzle is the **innerHTML** property we first saw in Section 9.3, which lets us directly update the inner HTML of a DOM element:

```
title.innerHTML = thumbnail.dataset.title;
description.innerHTML = thumbnail.dataset.description;
```

Putting everything together gives the final version of the **activateGallery** function, shown in Listing 11.8.

Listing 11.8: Updating the image title and description on click.

js/gallery.js

```
// Activates the image gallery.
// The main task is to attach an event listener to each image in the gallery
// and respond appropriately on click.
function activateGallery() {
  let thumbnails = document.querySelectorAll("#gallery-thumbs > div > img");
  let mainImage = document.querySelector("#gallery-photo img");
  // Image info to be updated
  let galleryInfo = document.querySelector("#gallery-info");
  let title = galleryInfo.querySelector(".title");
  let description = galleryInfo.querySelector(".description");

  thumbnails.forEach(function(thumbnail) {
    thumbnail.addEventListener("click", function() {
      // Set clicked image as display image.
      let newImageSrc = thumbnail.dataset.largeVersion;
      mainImage.setAttribute("src", newImageSrc);
    });
  });
}
```

```

// Change which image is current.
document.querySelector(".current").classList.remove("current");
thumbnail.parentNode.classList.add("current");

// Update image info.
title.innerHTML = thumbnail.dataset.title;
description.innerHTML = thumbnail.dataset.description;
});
});
}

```

Our final change involves syncing up the three columns for new visitors, so that the first column (current image indicator), second column (main image), and third column (image information) all match. This just involves updating the gallery index HTML as in Listing 11.9.

Listing 11.9: All three columns synced.

gallery/index.html

```

---
layout: default
title: Gallery for Learn Enough JavaScript to Be Dangerous
---

<div class="gallery col-three">
  <div class="col col-nav gallery-thumbs" id="gallery-thumbs">
    <div class="current">
      
    </div>
    .
    .
    .
  </div>
  <div class="col col-content">
    <div class="gallery-photo" id="gallery-photo">
      
    </div>
  </div>
  <div class="col col-aside gallery-info" id="gallery-info">
    <h3 class="title">Venice Beach</h3>
    <p class="description">An overhead shot of Venice Beach, California.</p>
  </div>
</div>

```

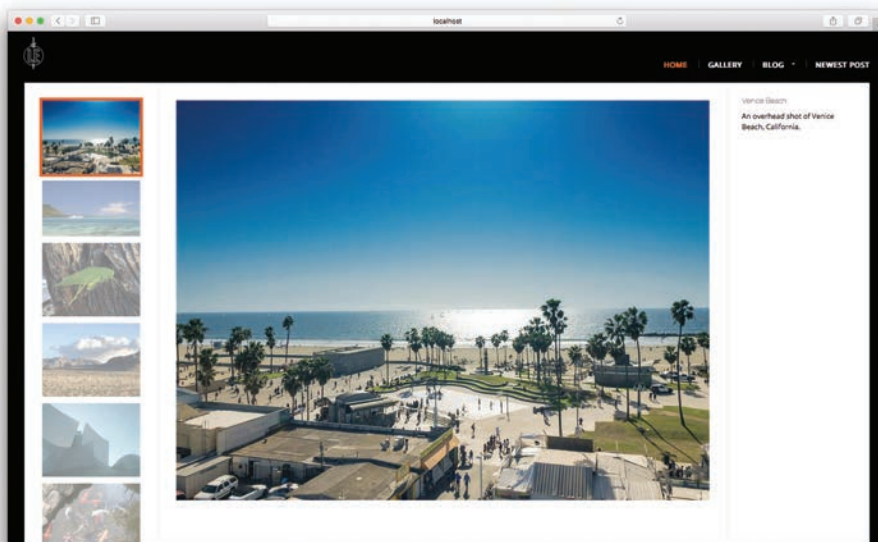


Figure 11.14: An overhead shot of Venice Beach, California.

Now all three of our columns agree, whether it's the Venice Beach pic that greets new visitors (Figure 11.14), a friendly sea turtle (Figure 11.15), Walt Disney Concert Hall in downtown Los Angeles (Figure 11.16), or the Flavian Amphitheater (Colosseum) in Rome (Figure 11.17).

11.4.1 Deploying

Because all the necessary files—including all the JavaScript—are completely local to our project (unlike some of the NPM modules in previous chapters), we can deploy our app to GitHub Pages with a simple **git push**:

```
$ git add -A
$ git commit -m "Finish the JavaScript gallery"
$ git push
```

Visiting the gallery at `<username>.github.io` and clicking on an image confirms it: We've deployed our dynamic JavaScript application to the live Web (Figure 11.18)!

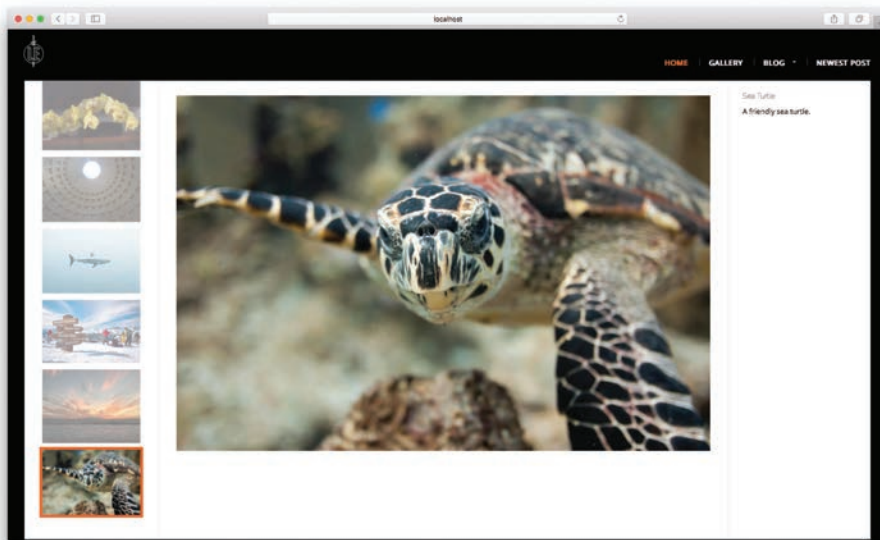


Figure 11.15: A friendly sea turtle.

(To learn how to host a GitHub Pages site using a custom domain instead of a github.io subdomain, see the free tutorial *Learn Enough Custom Domains to Be Dangerous* (<https://www.learnenough.com/custom-domains>).)

11.4.2 Exercise

1. When clicking on a new thumbnail image on the live site (Figure 11.18), you might notice a slight delay before the main image appears in the center. This is because, unlike the thumbnails, the large versions haven't been downloaded yet.

It's a common practice to prevent this small but annoying delay by *preloading* the images in the background to put them into the browser cache—a task we can accomplish with JavaScript. The trick is to create a new **Image** object (javascript image object) and assign it the **src** of the large image corresponding to each thumbnail. This forces the browser to download *all* the large images before the page is even loaded.

By filling in the code in Listing 11.10 and deploying the result, confirm that image preloading works, and that the resulting image swapping is snappy and

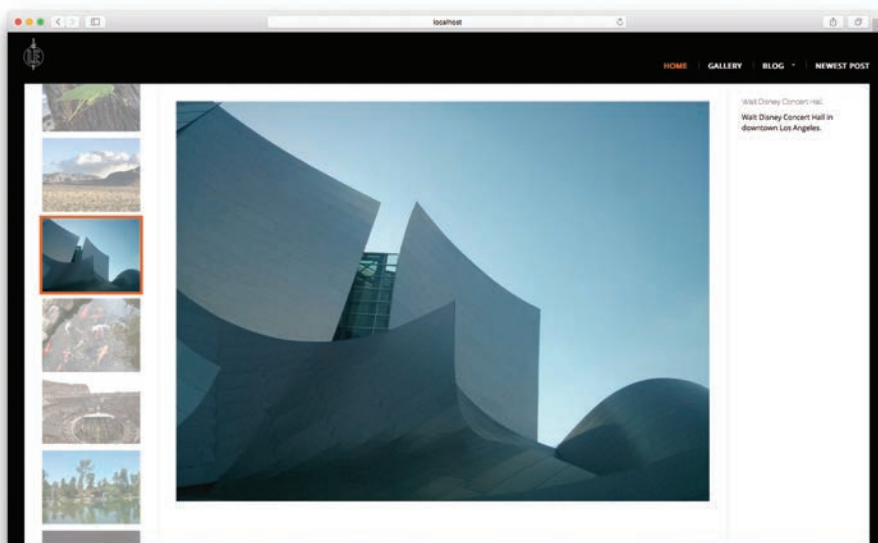


Figure 11.16: Walt Disney Concert Hall in downtown Los Angeles.

responsive. (Note that we've hoisted `newImageSrc` out of the listener, which is a big hint about what to use to replace `FILL_IN`.)

Listing 11.10: Preloading large versions.

`js/gallery.js`

```

// Activates the image gallery.
// The main task is to attach an event listener to each image in the gallery
// and respond appropriately on click.
function activateGallery() {
  let thumbnails = document.querySelectorAll("#gallery-thumbs > div > img");
  let mainImage = document.querySelector("#gallery-photo img");

  thumbnails.forEach(function(thumbnail) {
    // Preload large images.
    let newImageSrc = thumbnail.dataset.largeVersion;
    let largeVersion = new Image();
    largeVersion.src = FILL_IN;
    thumbnail.addEventListener("click", function() {
      // Set clicked image as display image.

```

```
mainImage.setAttribute("src", newImageSrc);

// Change which image is current.
document.querySelector(".current").classList.remove("current");
thumbnail.parentNode.classList.add("current");

// Update image info.
let galleryInfo = document.querySelector("#gallery-info");
let title       = galleryInfo.querySelector(".title");
let description = galleryInfo.querySelector(".description");

title.innerHTML      = thumbnail.dataset.title;
description.innerHTML = thumbnail.dataset.description;
});
});
}
```

11.5 Conclusion

Congratulations! You now know enough JavaScript to be *dangerous*.

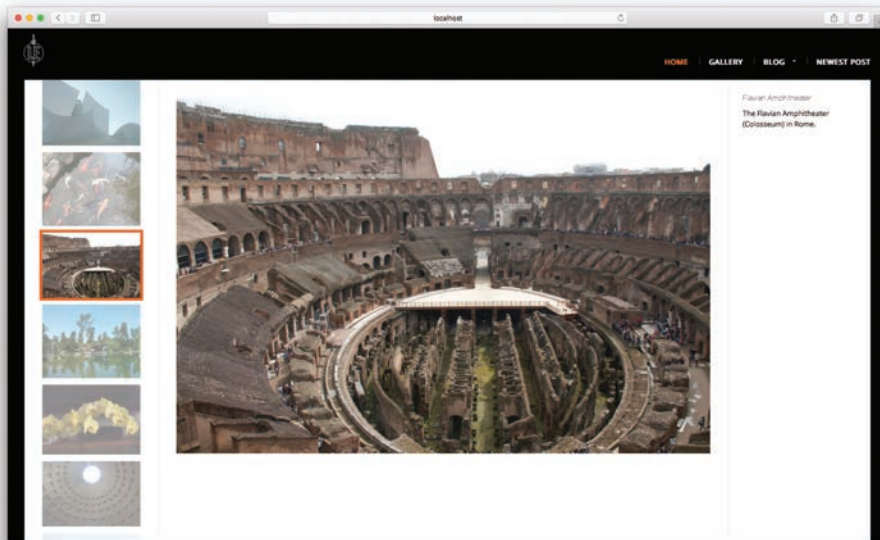


Figure 11.17: The Flavian Amphitheater (Colosseum) in Rome.

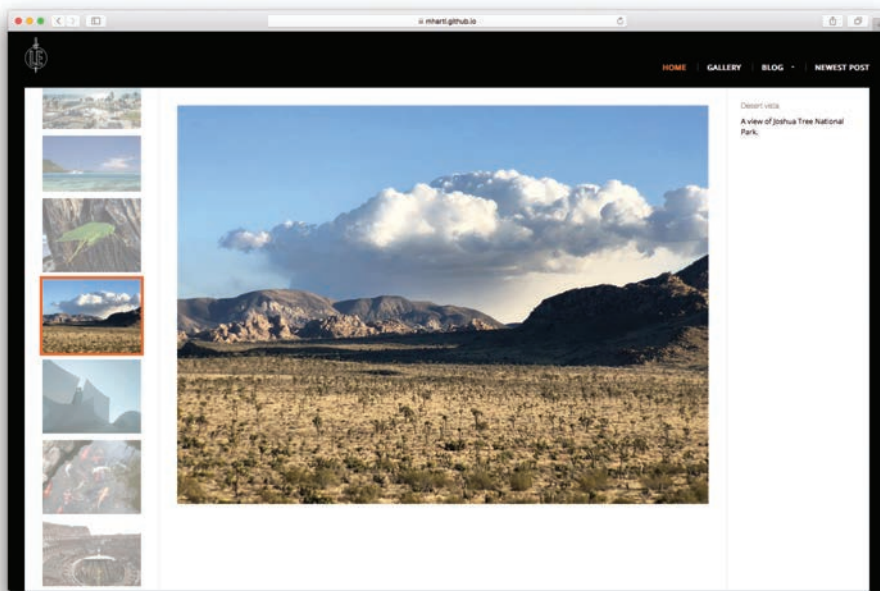


Figure 11.18: Our JavaScript gallery app on the live Web.

With the skills developed in this tutorial, you now have the preparation to go in multiple different directions. There are two in particular that I recommend. These are (1) learning more JavaScript and (2) making sure JavaScript isn't the only language you know.

11.5.1 Learning More JavaScript

There are approximately ∞ resources for learning more about JavaScript. Now that you know the basics, one good thing to focus on is expanding your command of the language syntax, as well as learning more advanced techniques (such as *async/await* and *promises*) and continuing to develop real applications. Here are a few resources that I've used or that have come highly recommended:

- Codecademy JavaScript (<https://www.codecademy.com/learn/introduction-to-javascript>): A guided in-browser introduction to JavaScript that’s highly complementary to the approach in *Learn Enough JavaScript to Be Dangerous*.
- Treehouse JavaScript (<https://teamtreehouse.com/library/topic:javascript>): Well-regarded interactive tutorials.
- Wes Bos JavaScript (<https://javascript30.com/>): A free course on vanilla JavaScript. Wes also offers a large number of premium courses (<https://wesbos.com/courses>), many of them focused on JavaScript topics like ES6 and Node.
- Learn JavaScript Essentials (<https://medium.com/javascript-scene/learn-javascript-b631a4af11f2#.lsb25e2f5>): An excellent list of resources compiled by Eric Elliott (https://medium.com/@_ericelliott), including links to additional courses and books.

11.5.2 Learning a New Language

Ask experienced devs if it’s important to know more than one programming language, and the answers will typically range from “yes!” to “extremely, indubitably yes!” Indeed, there are many reasons not to become a monoglot.

When it comes to building software for the greatest platform ever—the World Wide Web—the language I recommend (other than JavaScript) is *Ruby*, a powerful language designed for “programmer happiness”. In particular, Ruby is the language of two of the most popular frameworks for making web applications, *Sinatra* (used at companies like Disney and Stripe) and *Rails* (used at companies like GitHub, Hulu, and Airbnb).

Though suitable for bigger applications, Sinatra is the simpler framework, and is included as part of *Learn Enough Ruby to Be Dangerous* (<https://www.learnenough.com/ruby>). Rails is my preferred framework for making database-backed web applications, and is thoroughly covered by the *Ruby on Rails Tutorial* (<https://railstutorial.org/book>). Moreover, both can be used with JavaScript, with Rails/JavaScript integration being especially popular.

As a result, these are the recommended continuations of the *Learn Enough* sequence:

- *Learn Enough Ruby to Be Dangerous*
- *Ruby on Rails Tutorial*

Finally, for people who want the most solid foundation possible in technical sophistication, Learn Enough All Access (<https://www.learnenough.com/all-access>) is a subscription service that has special online versions of all the Learn Enough books and over 40 hours of streaming video tutorials. We hope you'll check it out!

Index

Symbols

\ (backslash), 75
" (double quotes), 25–29
[] (bracket) notation, 56
!! (bang bang), 43–44
' (single quote), 25
{ } (curly braces), 32, 37
(hash symbol), 195
% (modulo operator), 124
/ (slash character), 28–29
! operator, 42, 43
&& operator, 40, 41
+ operator, 27
|| operator, 41

A

accessing
 arrays, 56–58
 combining arrays, 63
 DOM (Document Object Model), 250
 string characters, 50
accumulators, 127
activating tools, 16
adding
 buttons, 193
 comments, 28–29
 event listeners, 195
 forms, 201, 207, 208
 HTML forms, 201
 notifications, 203
 pending tests, 162–163
 proof of concept, 187, 188
 stubs, 170
 testing, 169
alerts, 4, 34
alt attributes, updating, 247
anonymous functions, 110, 118, 196
applications
 code, 165 (*see also* code)
 deploying, 10–13
 functions from external files, 102
 image gallery (sample application), 235
 (*see also* image gallery)
 testing, 153
applying
 calculators, 66
 native assertions, 169
 REPLs (Read-Evaluate-Print Loops), 100
 technical sophistication, 28
 triple equals, 36
arguments, 22
 command-line, 226
 functions, 92, 93 (*see also* functions)
arrays, 55
 accessing, 56–58
 associative, 81
 creating URL-appropriate strings for, 118
 filter method, 122–125

- iteration, 62–64, 111, 112
- methods, 59–62
- popping, 61
- pushing, 61
- reversing, 60
- slicing, 58–59
- sorting, 60
- sorting numerical, 92–94
- splitting, 55–56
- undoing splits, 61–62
- asserting
 - applying active assertions, 169
 - equality, 168
- assigning
 - properties, 135
 - variables, 29, 31
- associative arrays, 81
- attributes, 35. *See also* string properties
 - src, 100
- automated tests, 132, 153, 159–164. *See also* testing
- auxiliary functions, 120, 121

B

- backslash (\), 75
- backtick syntax, 31–32
- bang bang (!!), 43–44
- Bash (Bourne-again shell), 6
- block structures, 38
- Boole, George, 367
- booleans
 - combining/inverting, 40–43
 - strings, 35–44
- Bourne-again shell. *See* Bash
- bracket ([]), notation, 56
- browserify utility, 188, 189, 191
- browsers. *See also* viewing
 - compatibility, 5
 - consoles, 14–19, 99
 - developer tools, 17
 - JavaScript in, 7–14
 - languages for, 1
- bugfixes, 157
- bugs, 166. *See also* errors; troubleshooting
- built-in objects, 3

- bundle command, 238
- bundles, 189
- buttons
 - adding, 193
 - wild, 194
- button tag, 192, 193, 206

C

- calculators, applying, 66
- calls, functions, 9
- CamelCase, 136
- cascading style sheet. *See* CSS (cascading style sheet)
- chains
 - methods, 104–110, 180
 - prototypes, 139
- changes, committing to, 11. *See also* modifying
- characters
 - iteration, 50
 - pushing, 173
 - string literals, 25 (*see also* strings)
- charAt method, 52, 53, 173
- chmod command, 23
- classes
 - current, 250, 251
 - equivalence, 124
- cloning, 235
- code. *See also* applications
 - applications, 165
 - DRY principle, 142
 - event listeners, 196
 - formatting, 38–39
 - palindromes, 153
 - refactoring, 53, 132, 165, 177–184
- columns, 38
- combining booleans, 40–43
- command lines, DOM manipulation at, 224–233
- commands
 - bundle, 238
 - chmod, 23
 - console.log, 18
 - node, 22, 23, 83
 - npm, 154, 156
 - which, 18, 22

- comments, 28–29. *See also* words
 - documentation, 106
 - JavaScript, 244
- comparing numbers, 93
- compatibility, browsers, 5
- concatenation, 27–32
- configuring
 - Jekyll, 237
 - JSON (JavaScript Object Notation), 157
 - repositories, 11
 - testing, 154–159
- `console.log`, 18, 33, 233
- consoles
 - browsers, 14–19, 99
 - JavaScript, 18
- constants, 66
- constructor functions, 135
- control, versions, 156
- control flow, strings, 35–44
- conventions
 - dates, 71
 - numbers, 158
 - regular expressions (regexes/regexps), 74
- converting numbers to strings, 67–69
- copying
 - files, 236
 - shell scripts, 232
- counting words, 86, 87
- creating. *See* configuring; formatting
- CSS (cascading style sheet), 229
 - current class, 250, 251
 - image gallery, 243
- curly braces (`{}`), 32, 37
- current, setting images as, 250–252
- customizing days of the week, 72

D

- dates, 69–73
- days of the week
 - customizing, 72 (*see also* dates)
 - factoring in (functions), 96
- debugging
 - JavaScript, 99
 - printing, 33

- tools, 16
- default behaviors, 209
- defining
 - functions, 91–95, 96
 - objects, 3, 135–138
 - prototypes, 143
 - TranslatedPhrase objects, 141
- deploying applications, 10–13
- `describe` function, 159, 160, 169
- detecting palindromes, 136, 138, 154, 155, 187–191, 216
- developer tools. *See also* tools
 - browsers, 17
 - MDN (Mozilla Developer Network), 139, 147, 148
- documentation
 - comments, 106
 - for File System, 216
 - JSDOM, 228
 - `urllib`, 220
- Document Object Model. *See* DOM (Document Object Model)
- document objects, 227
- documents, 196
- DOM (Document Object Model), 187, 197, 198, 215, 250
 - finding elements, 253
 - loading, 198, 199
 - manipulation, 4
 - manipulation at command lines, 224–233
- dot loads, 106
- dot notation, 17
- double quotes (`"`), 25
- DRY principle, 142
- duplicating code
 - DRY principle, 142
 - eliminating, 146–147
- dynamic HTML (Hypertext Markup Language), 202–205, 207. *See also* HTML (Hypertext Markup Language)

E

- ECMAScript, 5, 31. *See also* JavaScript
- editing GitHub Pages, 12
- Eich, Brendan, 5

- emojis, 108, 109
- empty strings, 26
- encapsulation, 96
- entering long strings, 211, 212
- equality, asserting, 168
- equivalence classes, 124
- errors
 - messages, 100, 101, 170
 - syntax, 27
 - testing, 165
- evaluation, short-circuit, 182
- events, 4
 - DOM (Document Object Model), 197, 198
 - listeners, 187, 192–201
- exec method, 76
- executable scripts, 22
- exponentiation, 66
- exporting
 - modules, 158
 - Phase objects, 158
- expressions, regular. *See* regular expressions

F

- factoring palindrome testers into functions, 195
- fat arrow, 94–95
- files
 - copying, 236
 - creating, 21
 - functions in, 95–104
 - JavaScript in, 21–22
 - reading from, 216–218
 - standalone JavaScript, 6
 - testing, 160 (*see also* testing)
- File System, documentation for, 216
- filter method, 116, 122–125, 180
- floating-point numbers, 65
- floats. *See* floating-point numbers
- forEach loops, 110–114, 116, 126, 178, 179
- fork capability (GitHub Pages), 235, 237
- for loops, 53, 58, 62
- formatting. *See also* configuring
 - code, 38–39
 - files, 21
 - indenting, 98

- lists of images, 243
 - printing, 33–35
 - quotes, 26
 - repositories, 11
- forms. *See also* documents
 - adding, 201, 207, 208
 - HTML (Hypertext Markup Language), 4, 200, 205–214
 - submitting, 209
- front-end JavaScript programs, 6
- functionality, non-standard, 150
- functional programming, 3, 95, 115–116, 179, 180
 - filter method, 122–125
 - map method, 116–122
 - reduce method, 126–133
 - TDD (test-driven development), 132–133
- functions, 3. *See also* methods; objects
 - anonymous, 110, 118, 196
 - arguments, 92, 93
 - auxiliary, 120, 121
 - calls, 9
 - console.log, 33
 - constructor, 135
 - defining, 91–95, 96
 - describe, 159, 160, 169
 - factoring palindrome testers into, 195
 - fat arrow, 94–95
 - in files, 95–104
 - forEach loops, 110–114
 - it, 159
 - method chaining, 104–110
 - nameless, 110
 - new, 69, 75, 77
 - palindrome, 104, 107, 132, 137, 140, 141, 142
 - Phrase, 135, 136, 137, 140, 142
 - prompt, 190, 205
 - querySelector, 229
 - return values, 92
 - sorting numerical arrays, 92–94
 - sum, 126
 - trigonometric, 66
 - urlify, 120

G

general-purpose programming languages, 1
getDay() method, 95
GitHub Pages, 11, 191
 editing, 12
 fork capability, 235
 renaming, 238
 saving settings, 14
 usernames, 12
Google Translate, 226, 232
Green, testing, 172–177

H

handling HTML forms, 205–214
Hansson, David Heinemeier, 148, 149
hash symbol (#), 195
hello, world!, 6, 8, 9–10
 adding proof of concept, 187, 188
 live on web pages, 15
HTML (Hypertext Markup Language)
 adding forms, 201
 button tag, 192, 193, 206
 dynamic, 202–205, 207
 forms, 4, 200, 205–214
 image gallery (sample application), 242–243
 methods unrelated to, 47
 skeletons, 7, 8

I

identifiers, 29
image gallery (sample application), 235
 changing image info, 252–259
 deploying, 256–259
 fork capability (GitHub Pages), 235, 237
 HTML (Hypertext Markup Language),
 242–243
 modifying images, 242–249
 prepping, 235–242
 setting images as current, 250–252
images
 modifying, 242–249
 updating, 254
img tags, 243
includes method, 48, 60

increment statements, 51
indenting, 98. *See also* formatting
 code, 38
indexes, 51
inheritance, 142
initializing NPM (Node Package Manager)
 modules, 157
inserting comments, 28–29. *See also* adding
Inspect Element, activating tools via, 16
installing
 Jekyll, 237
 Mocha, 154
 NPM (Node Package Manager), 219
instances
 methods, 57
 strings, 44
instantiating objects, 135
integers, summing, 127, 128
interpolation, 27–32
 backtick syntax, 31–32
inverting booleans, 40–43
iteration
 arrays, 62–64, 111, 112
 forEach loops, 110–114
 strings, 50–53, 112, 113
it function, 159

J

JavaScript
 applications, 10–13 (*see also* applications)
 in browsers, 7–14
 comments, 244
 consoles, 18
 debugging, 99
 in files, 21–22
 objects, 17 (*see also* objects)
 overview of, 5–7
 prepping, 239–240
 in REPLs (Read-Evaluate-Print Loops),
 14–20
 in shell scripts, 22–23
 submitting forms, 209
JavaScript Object Notation. *See* JSON
 (JavaScript Object Notation)

- Jekyll static site builder
 - configuring, 237
 - installing, 237
- joining, 27. *See also* concatenation
 - undoing splits, 61–62
- jQuery library, 195
- JSDOM
 - adding, 227 (*see also* DOM [Document Object Model])
 - documentation, 228
- JSON (JavaScript Object Notation), 157
- K**
- keys, 81
- key–value pairs, 81
- keywords, `return`, 119
- Knuth, Donald, 57
- L**
- length property, 35, 36, 52
- lengths object, 129
- letters method, 168, 169, 173, 174, 178, 180, 181
- listeners, events, 187, 192–201. *See also* events
- lists of images, 243
- literals, templates, 31–32
- LiveScript. *See* JavaScript
- loading
 - DOMs (Document Object Models), 198, 199
 - modules, 190
- logarithms, 66
- long strings, entering, 211, 212
- loops
 - `for`, 53, 58, 62
 - alternatives to, 116
 - `forEach`, 110–114, 116, 126, 178, 179
 - indexes, 51
 - iteration, 50
 - REPLs (Read-Evaluate-Print Loops), 6
- lowercase letters, 46
- M**
- main branch, serving websites from, 13
- main gallery images, setting, 245
- map method, 116–122
- Map object, 87–89
- matchers, regex, 172
- match method, 78, 85
- mathematics
 - floating-point numbers, 65
 - mathematical operations, 65–66
 - + operators, 27
- Math object, 66–67
 - converting numbers to strings, 67–69
- MDN (Mozilla Developer Network), 3, 139, 147, 148
- messages, error, 100, 101, 170
- methods, 18, 91
 - arrays, 59–62
 - chaining, 104–110, 180
 - `charAt`, 52, 53, 173
 - `exec`, 76
 - filter, 116, 122–125, 180
 - `getDay()`, 95
 - `includes`, 48, 60
 - instances, 57
 - letters, 168, 169, 173, 174, 178, 180, 181
 - map, 116–122
 - match, 78, 85
 - overriding, 143–144
 - palindrome, 175
 - `palindromeTester`, 209
 - `querySelector`, 195
 - `querySelectorAll`, 244
 - `reduce`, 116, 126–133
 - regular expressions (regexes/regexps), 75–76
 - `remove`, 231
 - `reverse`, 108, 109, 137, 150
 - `slice`, 59
 - `split`, 55, 79
 - strings, 44–50
 - `toLowerCase`, 107
 - `toString()`, 67
 - unrelated to HTML, 47
- mixed-cased palindromes, 164
- Mocha testing tool
 - installing, 154
 - pending tests, 162–163
 - settings, 154
 - starting, 156

- modifying
 - current class, 250, 251
 - image info, 252–259
 - images, 242–249
 - native objects, 147–152
- modules
 - exporting, 158
 - installing NPM (Node Package Manager), 219
 - loading, 190
 - NPM (Node Package Manager), 4, 153, 157
 - publishing, 184–186
- modulo operator (%), 124
- moving processedContent into methods, 140, 150
- Mozilla Developer Network. *See* MDN (Mozilla Developer Network)
- N**
- nameless functions, 110
- names
 - GitHub Pages, 12, 238
 - repositories, 236
 - variables, 29, 30 (*see also* identifiers)
- native assertions, applying, 169
- native objects, 65. *See also* objects
 - dates, 69–73
 - Map object, 87–89
 - mathematical operations, 65–66
 - Math object, 66–67
 - modifying, 147–152
 - numbers, 65–66
 - plain objects, 81–82
 - regular expressions, 73–81
 - unique words, 83–89
- Netscape Navigator, 5
- networks, MDN. *See* MDN (Mozilla Developer Network)
- new function, 69, 75, 77
- node command, 22, 23, 83
- Node.js, 18–20
 - shell scripts, 215 (*see also* shell scripts)
- Node package Manager. *See* NPM (Node Package Manager)
- Node REPL, 23, 38, 69, 83, 106. *See also* REPLs (Read-Evaluate-Print Loops)
- non-standard functionality, 150
- notation, 244
 - bracket ([]), 56
 - JSON (JavaScript Object Notation), 157
- notifications, adding, 203
- NPM (Node Package Manager), 4, 6
 - browserify utility, 188, 189, 191
 - installing, 219
 - modules, 153, 157
 - publishing, 184–186
- npm command, 154, 156
- null objects, 197
- numbers, 65–66
 - comparing, 93
 - conventions, 158
 - converting strings, 67–69
 - dates, 69–73
 - floating-point, 65
- numerical arrays, sorting, 92–94
- O**
- object-oriented languages, 17, 44
- objects
 - built-in, 3
 - defining, 3, 135–138
 - document, 227
 - functions attached to, 91 (*see also* functions; methods)
 - instantiating, 135
 - JavaScript, 17
 - JSON (JavaScript Object Notation), 157
 - lengths, 129
 - Map, 87–89
 - modifying native, 147–152
 - native, 65 (*see also* native objects)
 - null, 197
 - plain, 81–82
 - prototypes, 30
- operators
 - +, 27
 - !, 42, 43
 - &&, 40, 41
 - ||, 41
 - modulo operator (%), 124
- overriding methods, 143–144

P

palindrome function, 104, 107, 132, 137, 140, 141, 142, 175
 palindromes
 adding forms, 207, 208
 adding HTML for results, 202
 code, 153
 creating pages, 187–191
 detecting, 136, 138, 154, 155, 187–191, 216
 factoring testers into functions, 195
 long strings, 212
 mixed-cased, 164
 punctuated, 167
 testing, 160, 166
 translating, 144
 palindromeTester method, 209
 paragraphs
 pulling out, 230
 shell scripts, 231
 passwords, 40
 pasting shell scripts, 232
 pending tests, 162–163
 Perl, 215
 Phrase function, 135, 136, 137, 140, 142
 phrases, 3
 piping, 232
 plain objects, 81–82
 popping arrays, 61
 powers, 66
 prepping
 image gallery (sample application), 235–242
 JavaScript, 239–240
 printing strings, 33–35
 processedContent, moving into methods, 140, 150
 programming languages
 functional programming, 95
 general-purpose, 1
 HTML (Hypertext Markup Language), 4
 (see also HTML [Hypertext Markup Language])
 object-oriented languages, 17
 programs. *See also* applications
 front-end JavaScript, 6

 hello, world!, 6, 8, 9–10 (*see also* hello, world!)
 image gallery (*see* image gallery [sample application])
 wikip, 224
 writing, 6
 prompt function, 190, 205
 prompts, 4
 Node.js, 18–20
 properties, 82
 assigning, 135
 length, 35, 36, 52
 strings, 35–44
 prototype-based languages, 139
 prototypes, 139–147
 chains, 139
 defining, 143
 objects, 30, 139 (*see also* objects)
 publishing NPM (Node Package Manager)
 modules, 184–186
 punctuated palindromes, testing, 167
 pushing, 179
 arrays, 61
 characters, 173
 Python, 215

Q

querySelectorAll method, 244
 querySelector method, 195, 229

R

Rails, 205
 Read-Evaluate-Print Loops. *See* REPLs (Read-Evaluate-Print Loops)
 reading
 from files, 216–218
 from URLs, 218–223
 Real Programming, 21
 Red, testing, 164–172
 reduce method, 116, 126–133
 refactoring code, 53, 132, 165, 177–184
 references
 regular expressions, 75
 viewing, 230
 regex matchers, 172. *See also* regular expressions
 regressions, 165

- regular expressions (regexes/regexps), 56, 73–81
 - methods, 75–76
 - online builders, 74
 - references, 75
 - string methods, 77–80
 - reloading
 - pages, 101
 - palindrome function, 107
 - remove method, 231
 - renaming
 - GitHub Pages, 238
 - repositories, 236
 - repeating, DRY principle, 142
 - REPLs (Read-Evaluate-Print Loops), 6, 100.
 - See also* Node REPL
 - applying calculators, 66
 - code in, 39
 - JavaScript in, 14–20
 - loading files into, 138
 - quotes and, 26
 - shell scripts, 217 (*see also* shell scripts)
 - strings and, 25
 - repositories
 - creating, 11
 - image gallery, 235, 236 (*see also* image gallery [sample application])
 - resources, MDN (Mozilla Developer Network), 3
 - result areas, 204
 - return keyword, 119
 - return values, 92
 - reverse method, 108, 109, 137, 150
 - reversing
 - arrays, 60
 - strings, 106
 - roots, 66
 - Ruby, 215
- S**
- sample applications. *See* image gallery (sample application)
 - saving GitHub settings, 14
 - scope, variables, 63
 - scripting languages, 22, 215
 - scripts
 - executable, 22
 - shell, 4, 6 (*see also* shell scripts)
 - script tags, 9
 - semantic versioning, 158
 - sequences, string literals, 25. *see also* strings
 - settings. *See also* configuring; formatting
 - editing GitHub Pages, 12
 - Mocha, 154
 - saving (GitHub), 14
 - shell scripts, 4, 6, 215
 - Bash (Bourne-again shell), 6
 - copying, 232
 - DOM manipulation at command lines, 224–233
 - JavaScript in, 22–23
 - paragraphs, 231
 - pasting, 232
 - reading from files, 216–218
 - reading from URLs, 218–223
 - short-circuit evaluation, 182
 - Sinatra, 205
 - single quote ('), 25
 - slash (/) character, 28–29
 - slice method, 59
 - slicing arrays, 58–59
 - sorting
 - arrays, 60
 - numerical arrays, 92–94
 - split method, 79
 - splitting
 - arrays, 55–56
 - undoing splits, 61–62
 - src attribute, 100
 - standalone JavaScript files, 6
 - starting
 - image gallery (sample application), 235–242
 - Mocha, 156
 - state/length correspondence, troubleshooting, 130, 131
 - statements
 - console.log, 233
 - increment, 51
 - strict equality, asserting, 168
 - strings

- backtick syntax, 31–32
- booleans, 35–44
- concatenation, 27–32
- control flow, 35–44
- creating URL-appropriate for arrays, 118
- empty, 26
- entering, 211, 212
- `filter` method, 122–125
- instances, 44
- interpolation, 27–32
- iteration, 50–53, 112, 113
- literals, 25, 30
- methods, 44–50
- + operators, 27
- overview of, 25–27
- printing, 33–35
- properties, 35–44
- regular expressions (regexes/regexps), 77–80
- reversing, 106
- stubs, adding, 169
- submitting forms, 209
- `sum` function, 126
- summing integers, 127, 128
- synchronous versions, 216
- syntax
 - backtick, 31–32
 - defining functions, 96
 - errors, 27

T

- tables, truth, 40, 41
- tags. *See also* HTML (Hypertext Markup Language)
 - `button`, 192, 193, 206
 - `img`, 243
 - `script`, 9
- TDD (test-driven development), 3, 53, 153. *See also* testing
 - functional programming, 132–133
 - when to use, 165
- technical sophistication, 1, 47
 - applying, 28
 - definition of, 2–3
- templates, literals, 31–32
- test-driven development. *See* TDD (test-driven development)

- testing, 153
 - adding, 169
 - breaking, 164
 - configuring, 154–159
 - errors, 165
 - Green, 172–177
 - initial coverage, 159–164
 - Mocha (*see* Mocha)
 - palindromes, 160, 166
 - pending tests, 162–163
 - publishing NPMs (Node Package Managers), 184–186
 - Red, 164–172
 - refactoring code, 177–184
 - when to test, 165
- tests
 - automated, 132
 - suites, 160, 161, 167 (*see also* testing)
 - suites, running, 179
- text-to-speech. *See* TTS (text-to-speech)
- thumbnails, 249. *See also* images
- titles, updating images, 254
- `toLowercase` method, 107
- tools
 - activating, 16
 - browser consoles, 14–19
 - browser developer, 17
 - browserify utility, 188, 189, 191
 - debugging, 16
 - Mocha, 154 (*see also* testing)
- `toString()` method, 67
- `TranslatedPhrase` objects, 141, 142
- translating palindromes, 144
- trigonometric functions, 66
- triple equals, 36
- troubleshooting
 - bugfixes, 157
 - filtering, 123, 125
 - state/length correspondence, 130, 131
- truth tables, 40, 41
- TTS (text-to-speech), 224

U

- undoing splits, 61–62
- unique words, 83–89

updating

- alt attributes, 247
- images, 254

uppercase letters, 46

urlify function, 120

urlib documentation, 220

URLs (Uniform Resource Locators), reading from, 218–223

usernames, GitHub Pages, 12. *See also* names**V**

values

- boolean, 36 (*see also* booleans)
- key–value pairs, 81

Vanier, Mike, 64, 121, 122

variables, 29

- assigning, 29, 31
- creating, 110, 111
- interpolation, 27–32
- names, 29, 30
- scope, 63
- string concatenation and, 29 (*see also* concatenation)

versions, 5

- control, 156
- semantic versioning, 158
- synchronous, 216

viewing

- JavaScript, 7–14
- references, 230

Wweb applications, testing, 153. *See also* applications

web inspectors, 230

web pages, viewing JavaScript in, 7–14

which command, 18, 22

wikip program, 224

wild buttons, 194

words

- counting, 86, 87
- unique, 83–89

writing

- comments, 28–29
- to console logs, 34
- programs, 6
- shell scripts, 215 (*see also* shell scripts)

Z

zeros, 157

Zip codes, 75, 76, 77