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.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2022933200

ISBN-13: 978-0-13-784374-9
ISBN-10: 0-13-784374-7

ScoutAutomatedPrintCode

Contents

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/AI5wmnzzBqc?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!

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

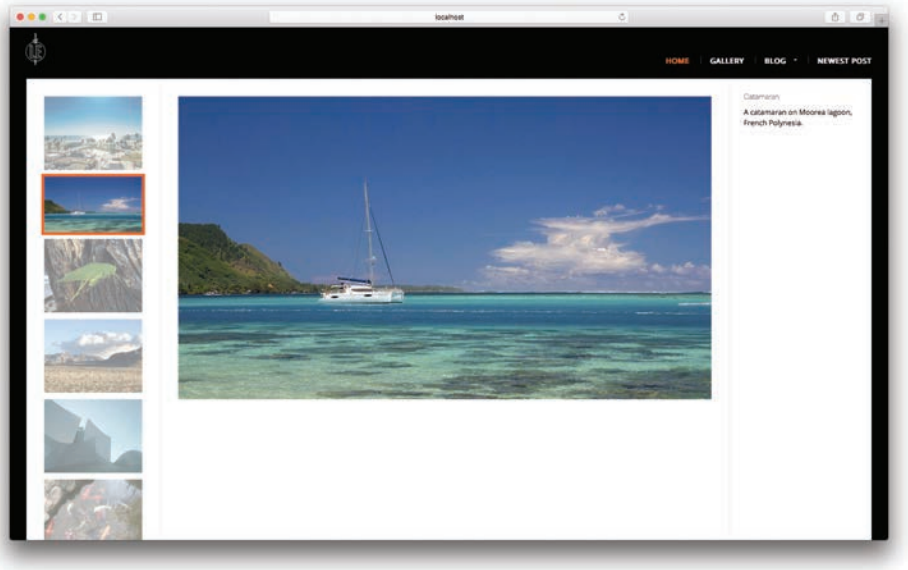

Figure 11.1: This is the gallery we're looking for.

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:

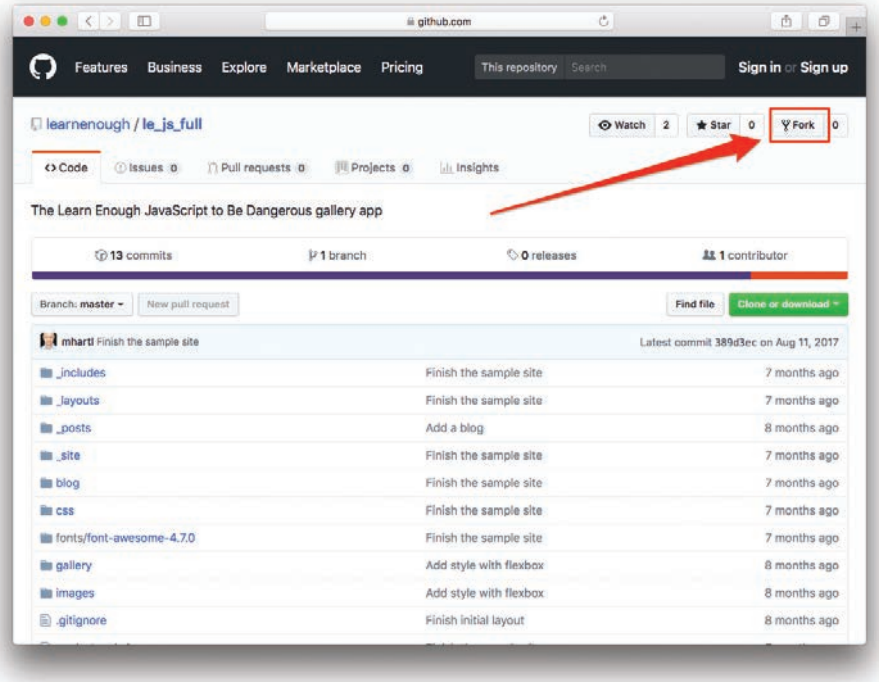

Figure 11.2: Forking the starting application at GitHub.

```
# 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*:

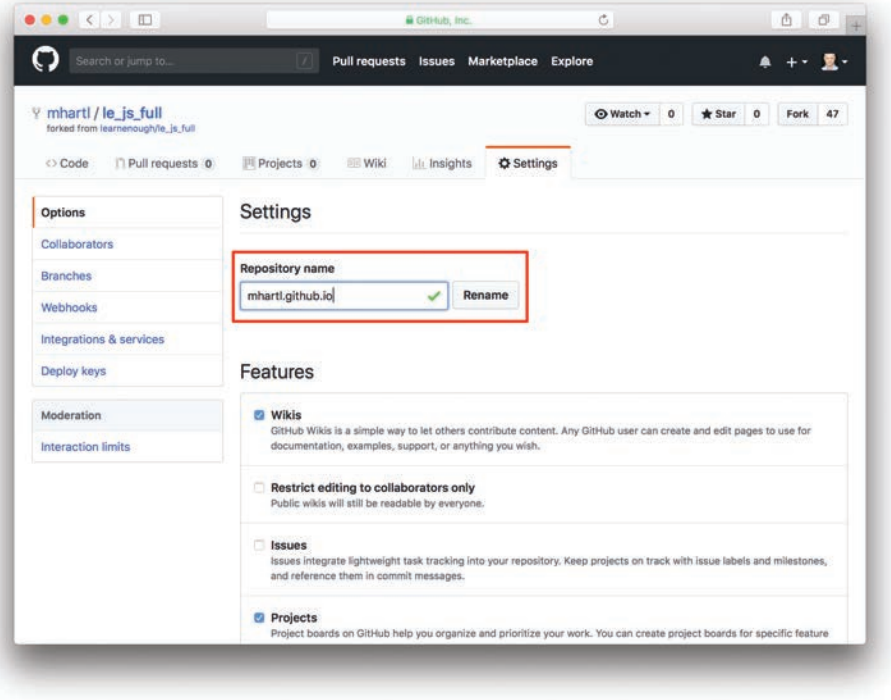

Figure 11.3: Renaming to the default GitHub Pages name.

```
$ 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.

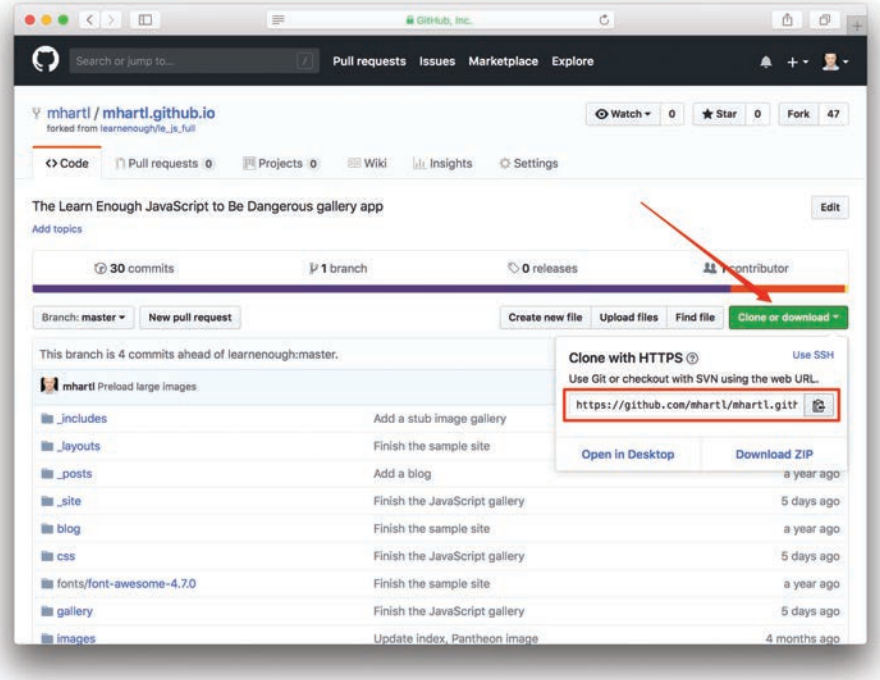

Figure 11.4: Getting the clone URL at GitHub.

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
```

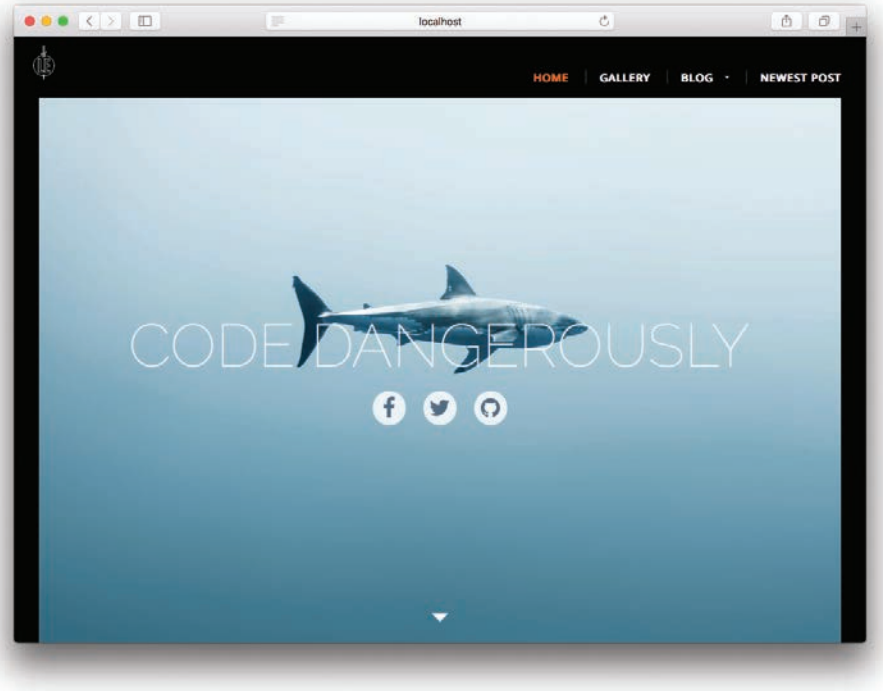

Figure 11.5: Our initial sample app.

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.

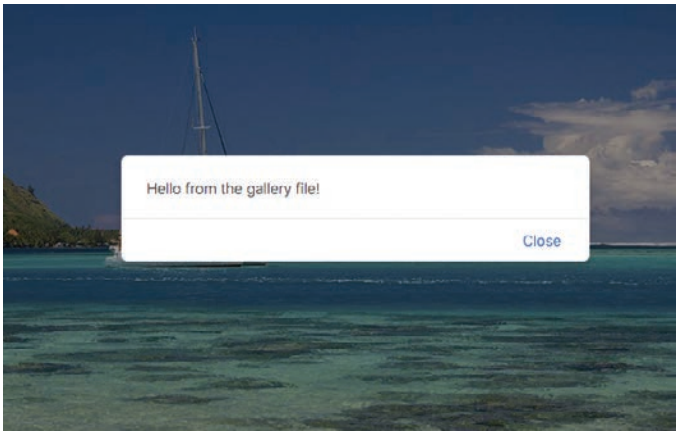

Figure 11.6: Hello from the gallery!

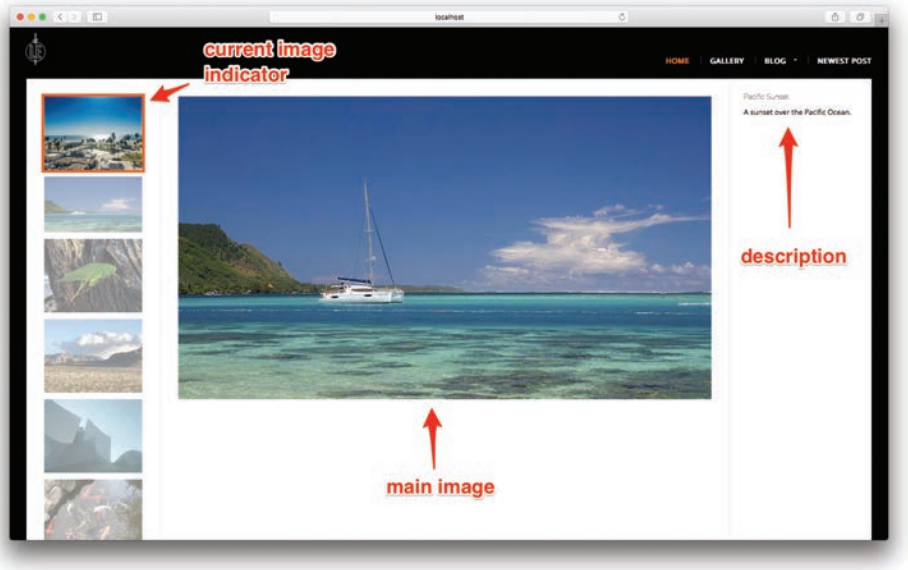

Figure 11.7: The initial gallery.

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

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

<div class="gallery col-three">
```

```
7     <div class="col col-nav gallery-thumbs" id="gallery-thumbs">
8       <div class="current">
9         <img src="/images/small/beach.jpg" alt="Venice Beach"
10             data-large-version="/images/large/beach.jpg"
11             data-title="Venice Beach"
12             data-description="An overhead shot of Venice Beach, California.">
13      </div>
14      .
15      .
16      .
17      <div>
18        <img src="/images/small/turtle.jpg" alt="turtle"
19             data-large-version="/images/large/turtle.jpg"
20             data-title="Sea Turtle"
21             data-description="A friendly sea turtle.">
22      </div>
23    </div>
24    <div class="col col-content">
25      <div class="gallery-photo" id="gallery-photo">
26        <img src="/images/large/boat.jpg" alt="Catamaran">
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>
  <img src="/images/small/sunset.jpg" alt="sunset"
       data-large-version="/images/large/sunset.jpg"
       data-title="Pacific Sunset"
       data-description="A sunset over the Pacific Ocean.">
</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.

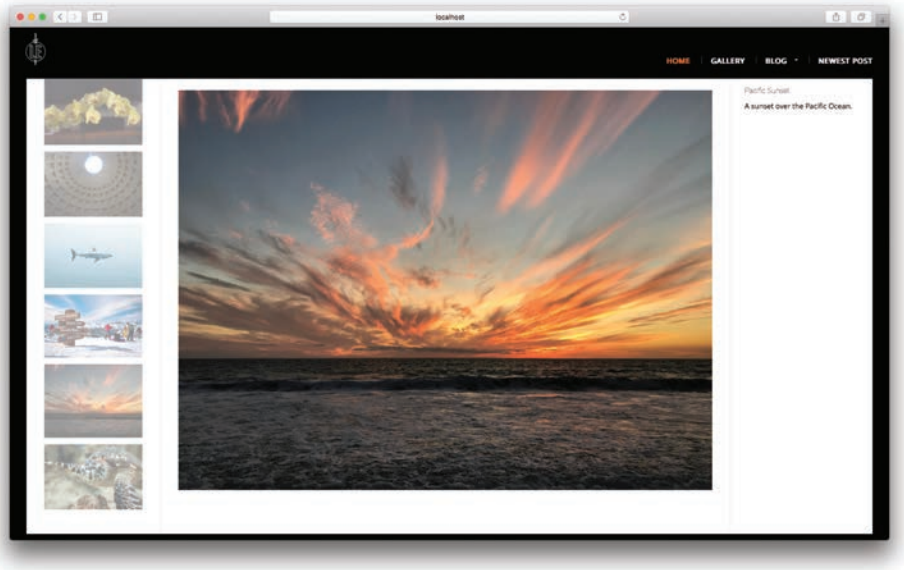

Figure 11.8: A Pacific sunset.

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);
    });
  });
}
```

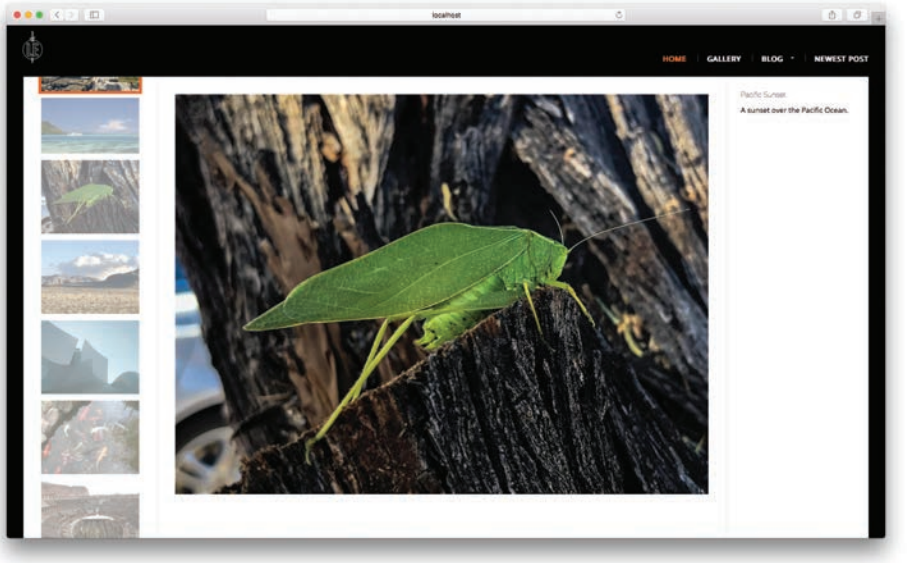

Figure 11.9: The image/description match in Figure 11.8 was a coincidence.

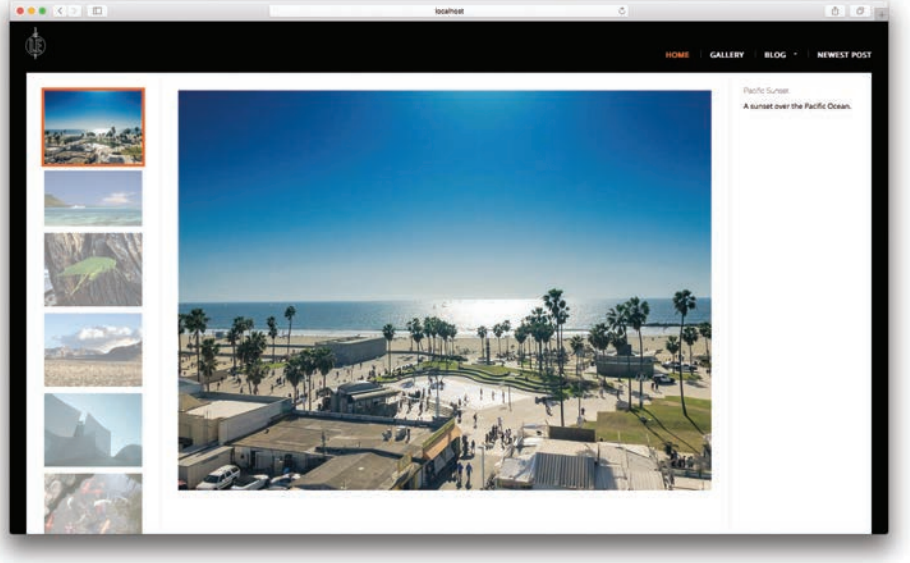

Figure 11.10: The "current image" match here is also a coincidence.

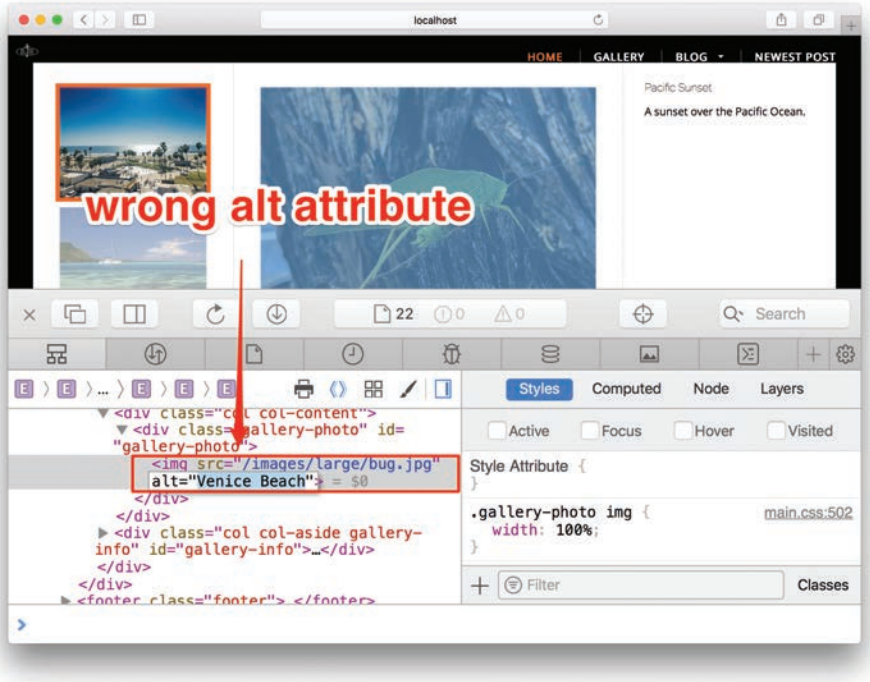

Figure 11.11: The `alt` attribute doesn't match the image `src`.

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">
  <img src="/images/small/beach.jpg" alt="Venice Beach"
       data-large-version="/images/large/beach.jpg"
       data-title="Venice Beach"
       data-description="An overhead shot of Venice Beach, California.">
</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.

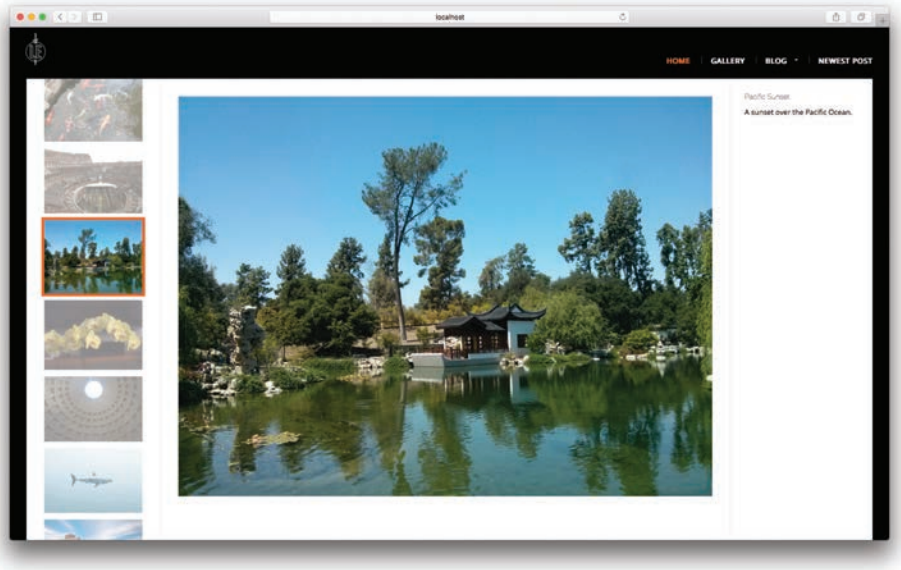

Figure 11.13: The Chinese Garden at The Huntington.

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">
      <img src="/images/small/beach.jpg" alt="Venice Beach"
           data-large-version="/images/large/beach.jpg"
           data-title="Venice Beach"
           data-description="An overhead shot of Venice Beach, California.">
    </div>
    .
    .
    .
  </div>
  <div class="col col-content">
    <div class="gallery-photo" id="gallery-photo">
      <img src="/images/large/beach.jpg" alt="Venice Beach">
    </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>
```

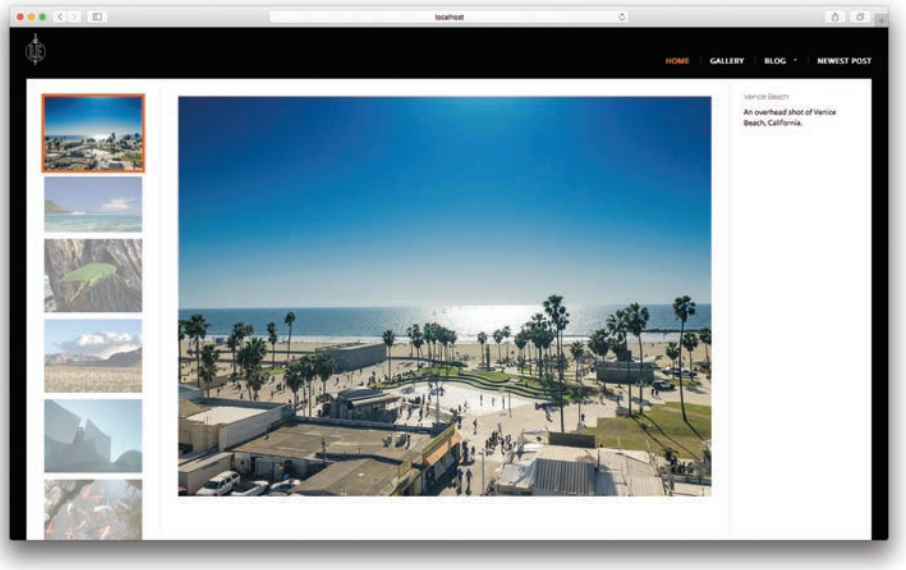

Figure 11.14: An overhead shot of Venice Beach, California.

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)!

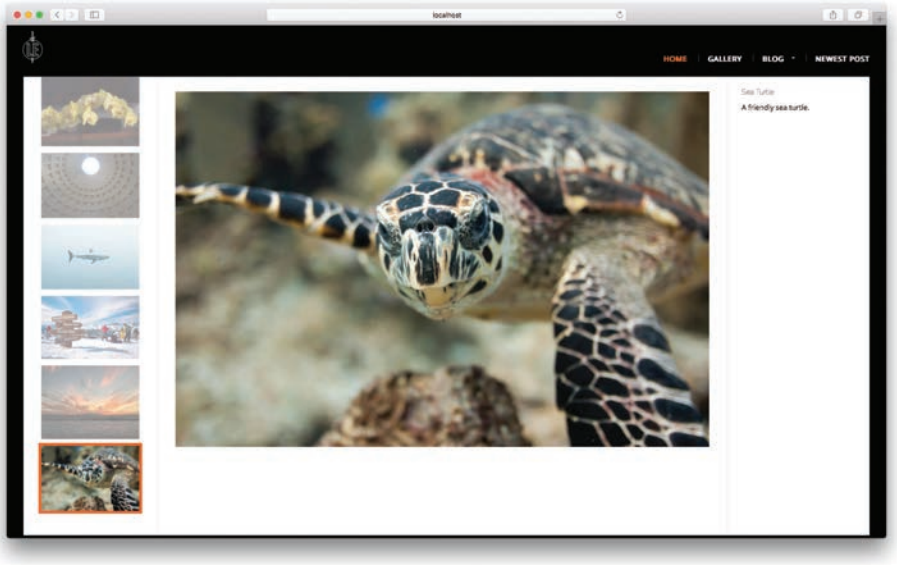

Figure 11.15: A friendly sea turtle.

(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

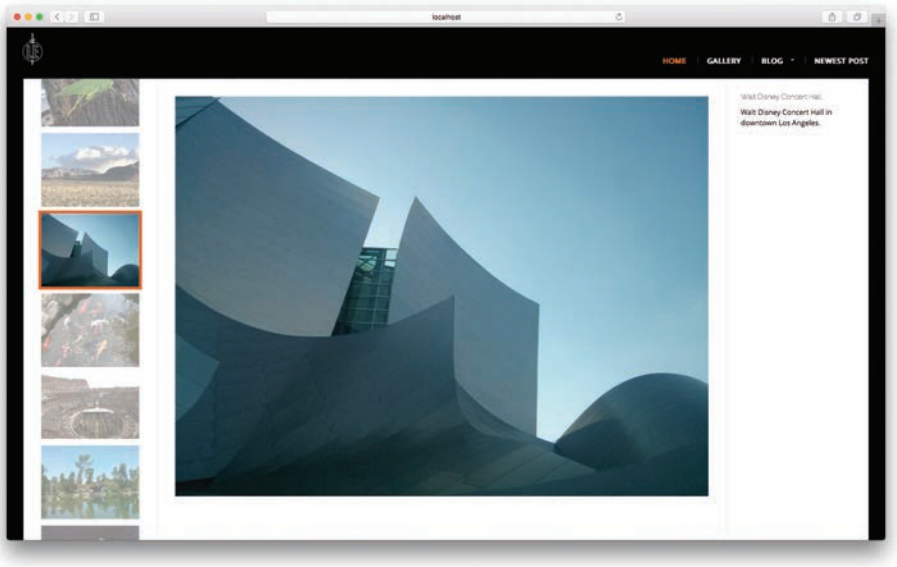

Figure 11.16: Walt Disney Concert Hall in downtown Los Angeles.

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*.

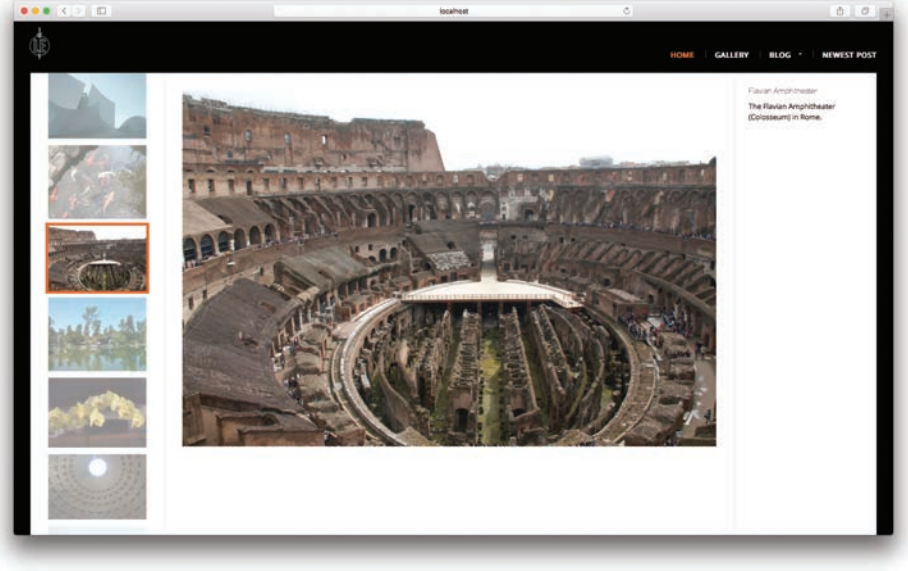

Figure 11.17: The Flavian Amphitheater (Colosseum) in Rome.

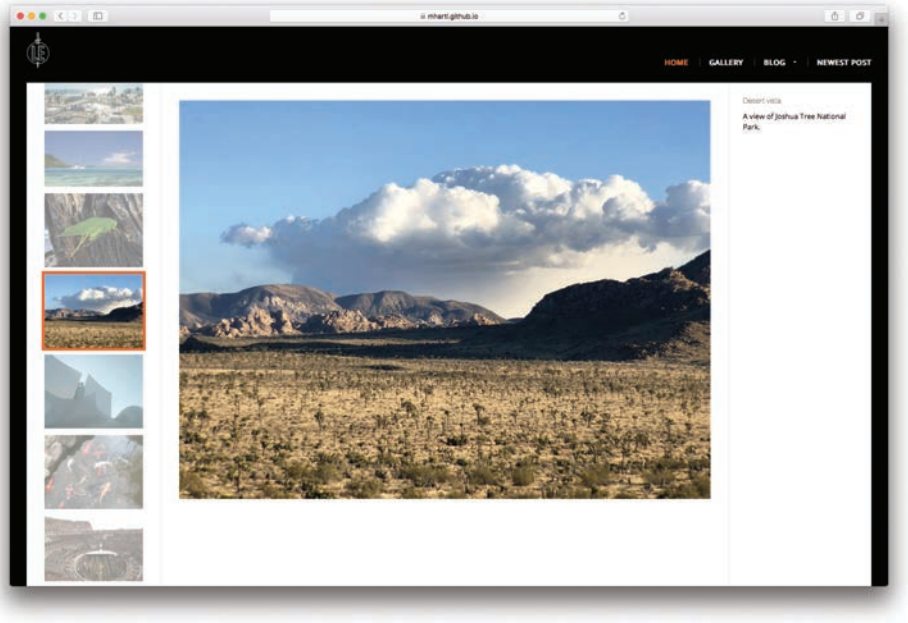

Figure 11.18: Our JavaScript gallery app on the live Web.

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

D

E

F

G

H

I

J

P

Q

R