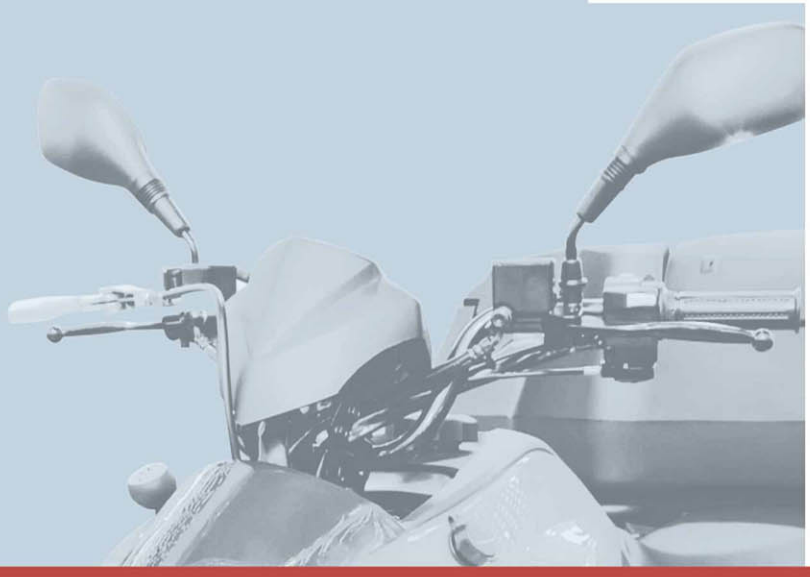
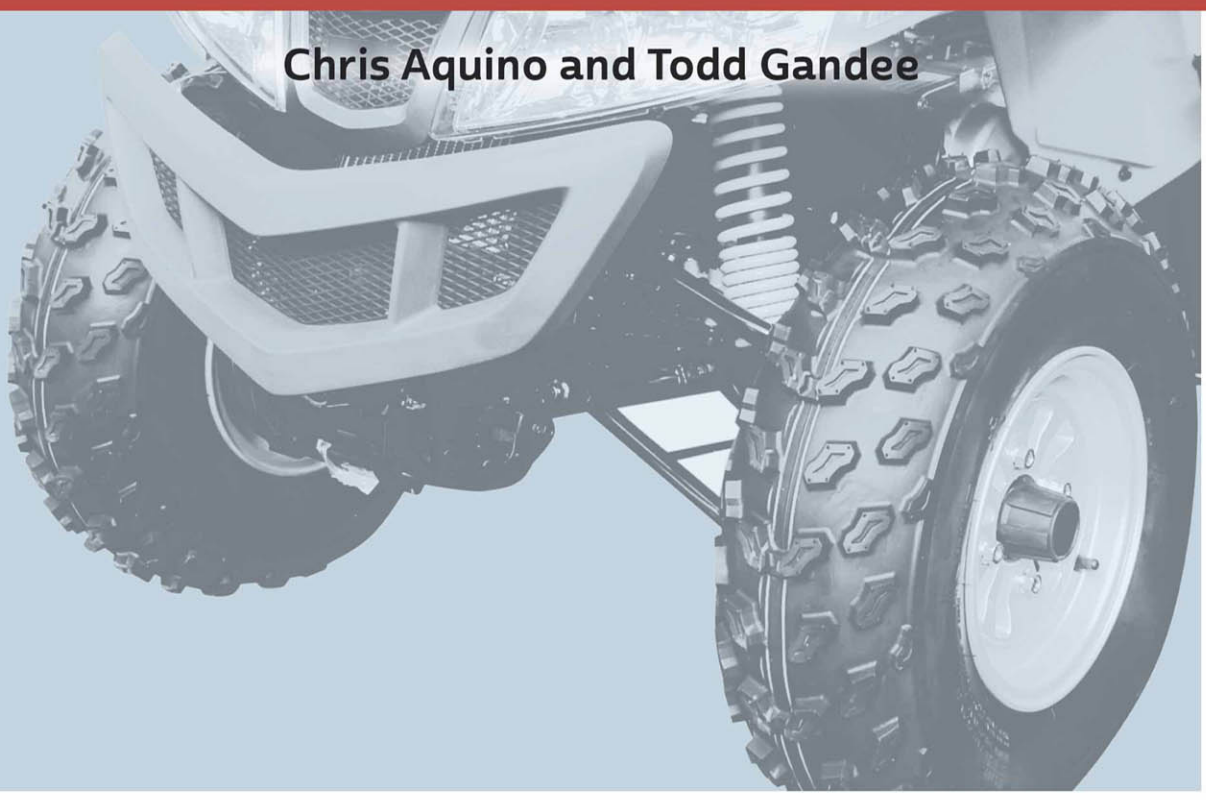
Big Nerd Ranch



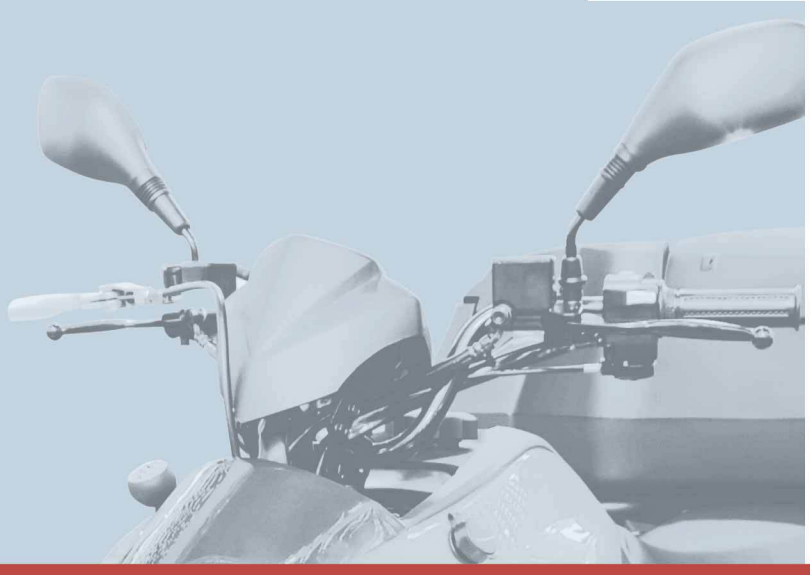
Front-End Web Development

The Big Nerd Ranch Guide

Chris Aquino and Todd Gandee



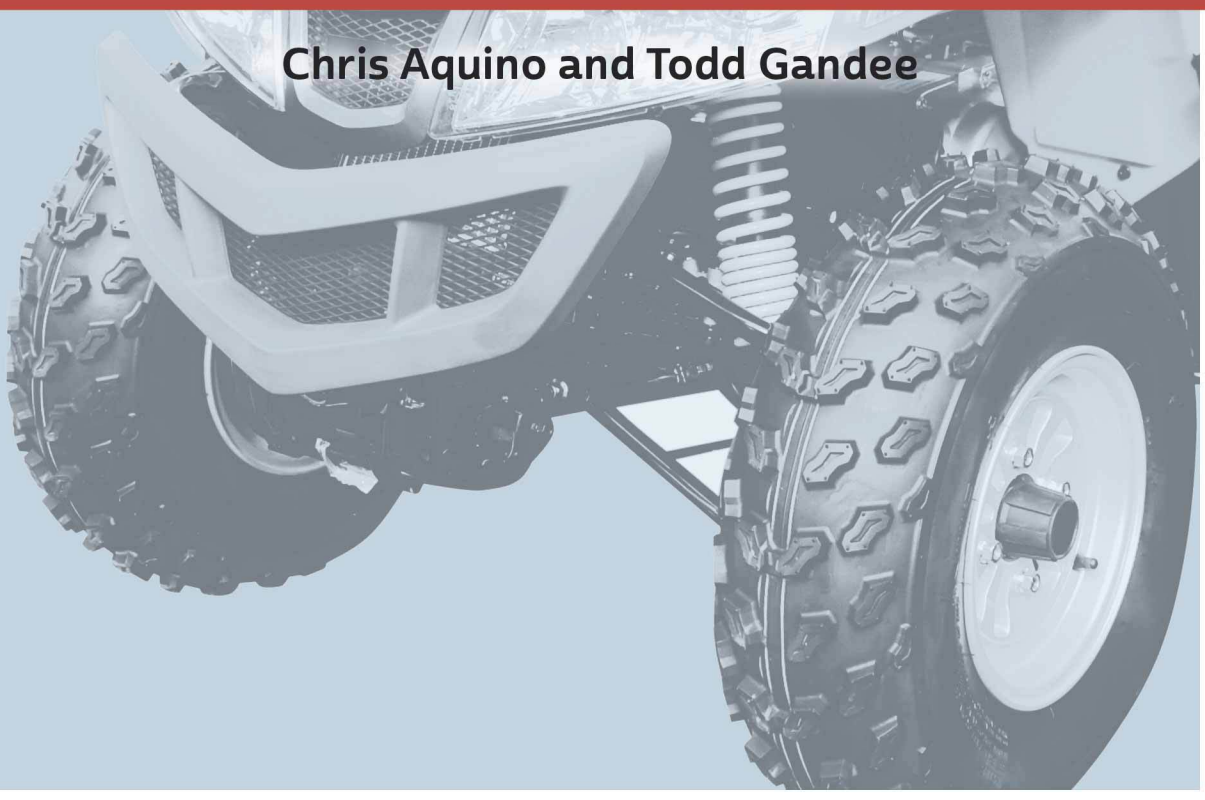
Big Nerd Ranch



Front-End Web Development

The Big Nerd Ranch Guide

Chris Aquino and Todd Gandee



Next, you need to go through the array of thumbnails, one item at a time. As you visit each one, you will call **`addThumbClickHandler`** and pass the thumbnail element to it. That may seem like several steps, but because `thumbnails` is a proper array, you can do all of this with a single method call.

Add a call to the `thumbnails.forEach` method in `main.js` and pass it the **`addThumbClickHandler`** function as a callback.

```
...
function initializeEvents() {
  'use strict';
  var thumbnails = getThumbnailsArray();
  thumbnails.forEach(addThumbClickHandler);
}
```

Note that you are passing a named function as a callback. As you will read later, this is not always a good choice. However, in this case it works well, because **`addThumbClickHandler`** only needs information that will be passed to it when **`forEach`** calls it – an item from the `thumbnails` array.

Finally, to see everything in action, add a call to **`initializeEvents`** at the very end of `main.js`.

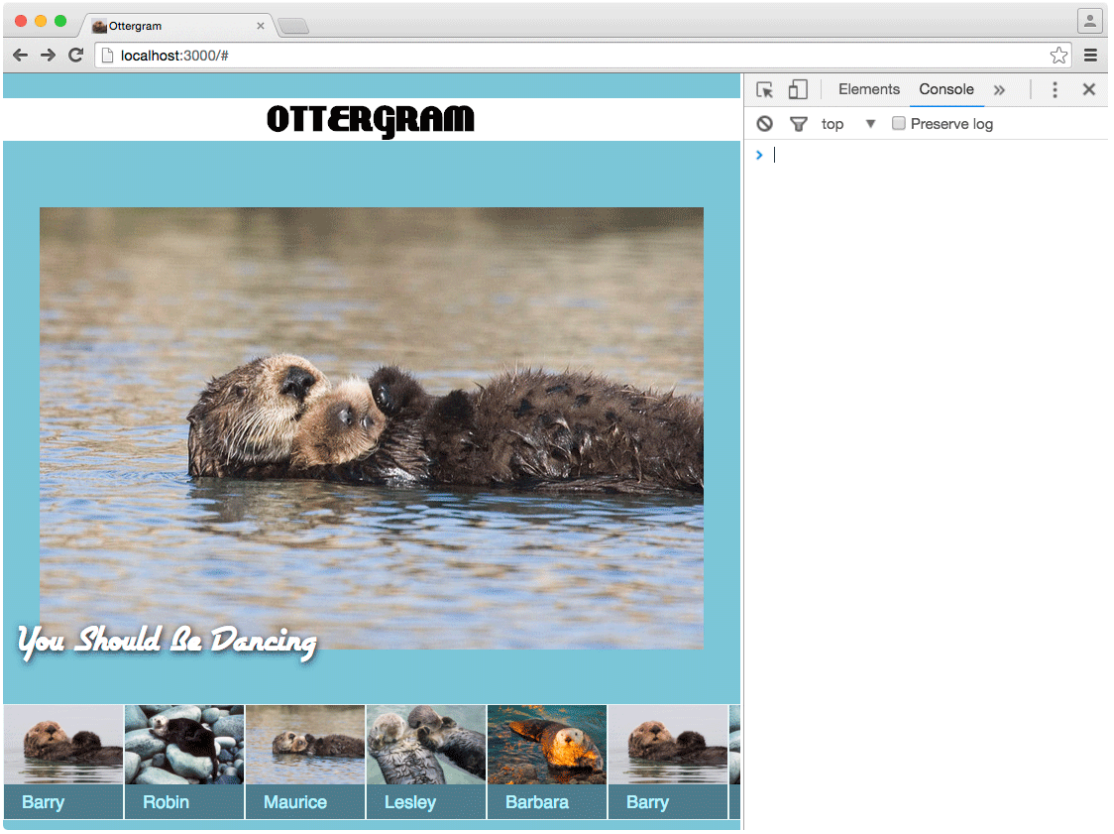
```
...
function initializeEvents() {
  'use strict';
  var thumbnails = getThumbnailsArray();
  thumbnails.forEach(addThumbClickHandler);
}

initializeEvents();
```

Remember, as the browser reads through each line of your JavaScript code, it runs the code. For most of `main.js`, it is only running variable and function declarations. But when it reaches the line `initializeEvents();`, it will run that function.

Save and return to the browser. Click a few different thumbnails and see the fruits of your labor (Figure 6.28).

Figure 6.28 You should indeed be dancing



Sit back, relax, and enjoy clicking some otters! There was a lot to work through and absorb while building your site's interactive layer. In the next chapter you will finish Ottergram by adding visual effects for extra pop.

Silver Challenge: Link Hijack

The Chrome DevTools give you a lot of power for toying with pages that you visit. This next challenge is to change all of the links on a search results page so that they do not go anywhere.

Go to your favorite search engine and search for "otters." Open the DevTools to the console. With the functions you wrote in Ottergram as a reference, attach event listeners to all of the links and disable their default click functionality.

Gold Challenge: Random Otters

Write a function that changes the `data-image-url` of a random otter thumbnail so that the detail image no longer matches the thumbnail. Use the URL of an image of your choosing (though a web search for "tacocat" should provide a good one).

For an extra challenge, write a function that resets your otter thumbnails to their original `data-image-url` values and changes another one at random.

For the More Curious: Strict Mode

What is strict mode, and why does it exist? It was created as a cleaner mode of JavaScript, catching certain kinds of coding mistakes (like typos in variable names), steering developers away from some error-prone parts of the language, and disabling some language features that are just plain bad.

Strict mode provides a number of benefits. It:

- enforces the use of the `var` keyword
- does not require `with` statements
- places restrictions on the way the `eval` function can be used
- treats duplicate names in a function's parameters as a syntax error

All this just for adding the `'use strict'` directive to the top of a function. As a bonus, the `'use strict'` directive is ignored by older browsers that do not support it. (These browsers simply see the directive as a string.)

You can read more about strict mode on the MDN at `developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode`.

For the More Curious: Closures

Earlier we mentioned that developers often prefer to use anonymous functions as callbacks instead of named functions. **`addThumbClickHandler`** illustrates why an anonymous function is a better solution.

Let's say you tried to use a named function, **`clickFunction`**, for the callback. Inside of that function, you have access to the `event` object because it will be passed in by **`addEventListener`**. But the body of **`clickFunction`** has no access to the `thumb` object. That parameter is only accessible *inside* the **`addThumbClickHandler`** function.

```
function clickFunction (event) {
  event.preventDefault();

  setDetailsFromThumb(thumb); // <--- This will cause an error
}

function addThumbClickHandler(thumb) {
  thumb.addEventListener('click', clickFunction);
}
```

On the other hand, using an anonymous function does give it access to the `thumb` parameter, because it is also inside of **`addThumbClickHandler`**. When a function is defined inside of another function, it can use any of the variables and parameters of this outer function. In computer science terms, this is known as a *closure*.

When the **`addThumbClickHandler`** function runs, it calls **`addEventListener`**, which associates the callback function with the click event. The browser keeps track of these associations, internally holding a reference to the callback function and running the callback when the event occurs.

Technically, when the callback is eventually executed, the variables and parameters of **`addThumbClickHandler`** no longer exist. They went away when **`addThumbClickHandler`** finished running. But, the callback "captures" the values of **`addThumbClickHandler`**'s variables and parameters. The callback uses these captured values when it runs.

For a deeper dive, read up on closures in the MDN.

For the More Curious: NodeLists and HTMLCollections

There are two ways to retrieve lists of elements that live in the DOM. The first one is **`document.querySelectorAll`**, which returns a NodeList. The other is **`document.getElementsByTagName`**, which differs from **`document.querySelectorAll`** in that you can only pass it a string with a tag name, like `"div"` or `"a"`, and also in that it returns an *HTMLCollection*.

Neither NodeLists nor HTMLCollections are true arrays, so they lack array methods such as **`forEach`**, but they do have some very interesting properties.

HTMLCollections are *live nodes*. This means that when changes are made to the DOM, the contents of an HTMLCollection can change without you having to call **`document.getElementsByTagName`** again.

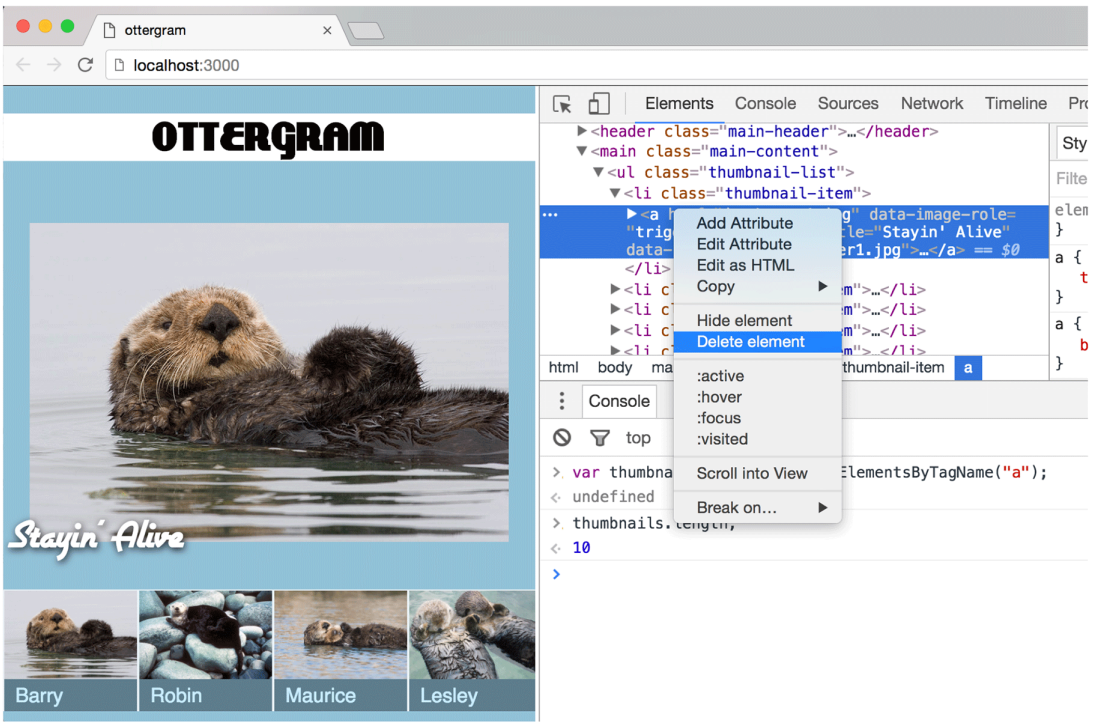
To see how this works, try the following in the console.

```
var thumbnails = document.getElementsByTagName("a");
thumbnails.length;
```

After getting all of the anchor elements as an HTMLCollection, you print the `length` of that list to the console.

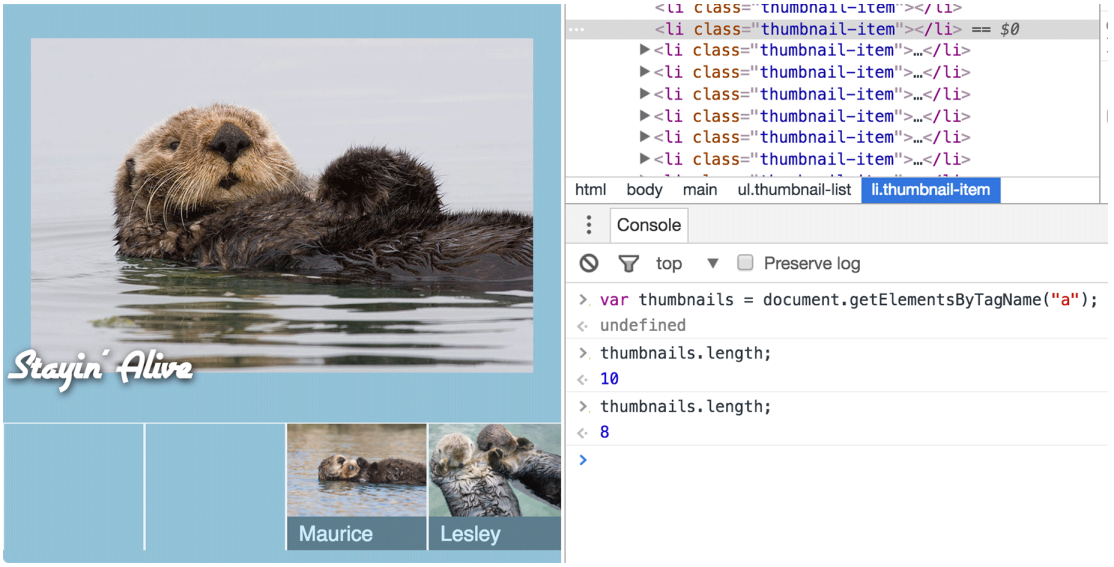
Now, remove some of the anchor tags from the page using the elements panel in the DevTools: Control-click (right-click) one of the list items and choose Delete element (Figure 6.29).

Figure 6.29 Deleting a DOM element with the DevTools



Do this several times, then enter `thumbnails.length` into the console again. You should see that the `length` is different (Figure 6.30).

Figure 6.30 The length value changes after deleting elements



Converting NodeLists and HTMLCollections to arrays not only makes them more convenient to work with via array methods, but you also have the guarantee that the items in the array will not change, even if the DOM is modified.