4.3.7 Delete a Task

Now we're ready to add interactivity to the buttons we just created, and we'll start with the Delete button. This won't be as straightforward as adding an event listener to the `formEl`.

In an ideal world, we could do something like this:

```
var allButtonEls = document.querySelector(".delete-btn");
allButtonEls.addEventListener("click", myFunction);
```

However, the Delete buttons don't exist when the page first loads, so this would give us an error. Also, `document.querySelector()` returns the first element it finds on the page, meaning only the first Delete button would work.

One solution would be to add individual event listeners to the Delete buttons as we make them:

```
var deleteButtonEl = document.createElement("button");
deleteButtonEl.addEventListener("click", myFunction);
```

This might make our code harder to follow, though, and the number of event listeners we would be creating could lead to memory leaks and performance issues down the road.

Fortunately, with **event delegation**, we can set up the click event listener on a parent element and then, through that single event listener, determine which child elements were clicked.

Clicks in JavaScript are a funny thing. If an element that has parent elements is clicked, the click event **bubbles**, or travels, upwards to its parents. Watch the following video to further explore this idea of event bubbling:

We'll eventually have Delete buttons in three different columns, so the most appropriate parent element to delegate this click responsibility to would be the `<main>` element. It currently has a `class="page-content"` attribute, which is used by our CSS to style the page. Let's add a new `id="page-content"` attribute to the main element, so that it has both `class` and `id` attributes with values of `page-content`, like this:

```
<main class="page-content" id="page-content">
```

In `script.js`, add a reference to the `page-content` element at the top and then an event listener at the bottom of the file:

```
var pageContentEl = document.querySelector("#page-content");

// other logic...

pageContentEl.addEventListener("click", taskButtonHandler);
```

The `addEventListener()` method references a function that doesn't exist yet, so let's create that now. Add the `taskButtonHandler()` function before the `pageContentEl.addEventListener()` call:

```
var taskButtonHandler = function(event) {
  console.log(event.target);
};
```

Notice that we're using a familiar friend, the `event` object. In this case, we're console logging `event.target`. `event.target` reports the element on which the event occurs, in this case, the `click` event. Test the app in the browser and click on different elements inside `page-content` to see what `event.target` is each time:

The event listener triggers not only when `page-content` itself is clicked but any element inside (the `<ul>` lists, the `<button>` elements, etc.). Thanks to `event.target`, though, we can know exactly which element triggered the listener.

Test the app again and click on an Edit button, then click on a Delete button. You'll see that `event.target` is different in both cases:

```
<button class="btn edit-btn" data-task-id="0">Edit</button>
<button class="btn delete-btn" data-task-id="0">Delete</button>
```

What uniquely identifies the Delete versus the Edit? The Delete button has a class called `.delete-btn` on it. While it's easy to see that distinction in the console, how can we translate that into code? How could we check the class name on the element that was clicked?

We could use a property that we've used in the past, `className`, or there's a catch-all method called `matches()` that was created specifically for checking if an element matches certain criteria. The `matches()` method is similar to using the `querySelector()` method, but it doesn't find and return an element. Instead, it returns `true` if the element would be returned by a

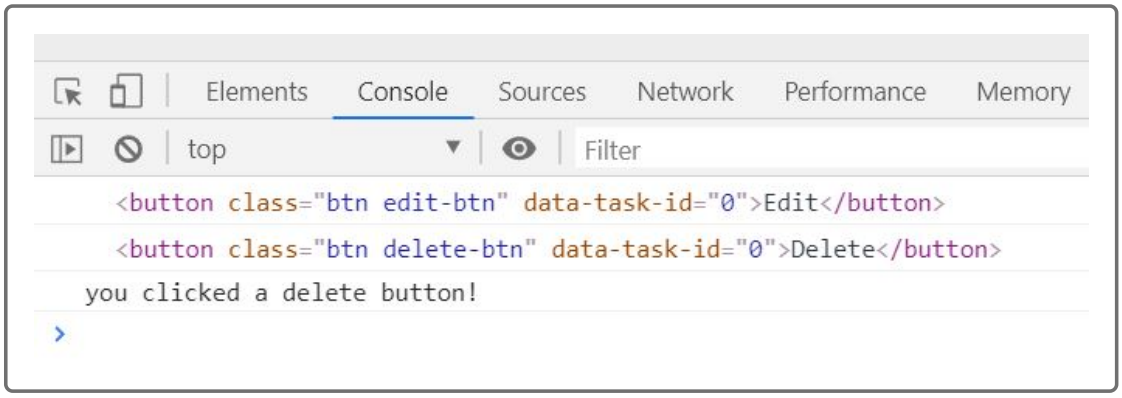
`querySelector()` with the same argument, and it returns `false` if it wouldn't.

In `taskButtonHandler()`, add the following `if` statement:

```
var taskButtonHandler = function(event) {
  console.log(event.target);

  if (event.target.matches(".delete-btn")) {
    console.log("you clicked a delete button!");
  }
};
```

Save `script.js` and refresh the browser. Click on different elements again and verify that the second `console.log()` only happens if a Delete button is clicked:



Remember, the click event is always firing regardless of which elements are clicked, so it's up to us to capture the element that we actually care about. With the `if` statement in place, we now know exactly when a Delete button is clicked. The next problem to solve is knowing which task the Delete button belongs to. There could be tens or hundreds of Delete buttons on the page, but think about what we set up earlier that helps us uniquely identify them.

PAUSE

What HTML attribute holds the task's ID?

The `data-task-id` attribute

Hide Answer

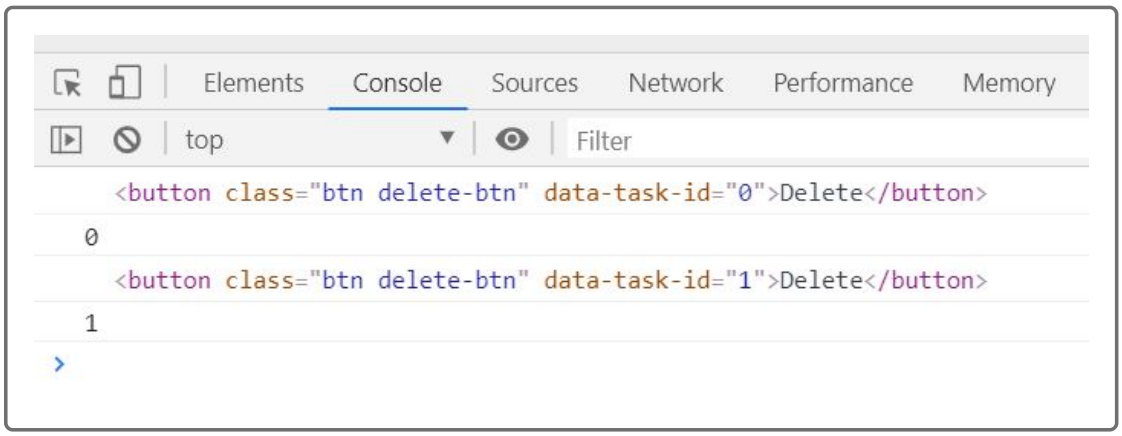
Previously, when we created these `<button>` elements, we used the `setAttribute()` method to set the `data-task-id`. Now we want to get that attribute. It's no coincidence that the method we'll use is `getAttribute()`. The `getAttribute()` method is available on any DOM element. Since `event.target` is a reference to a DOM element, we can use the method there as well.

Update `taskButtonHandler()` to console log what this method returns:

```
var taskButtonHandler = function(event) {
  console.log(event.target);

  if (event.target.matches(".delete-btn")) {
    // get the element's task id
    var taskId = event.target.getAttribute("data-task-id");
    console.log(taskId);
  }
};
```

Test the app in the browser again and notice that the second `console.log()` returns a number that corresponds to the `data-task-id` attribute on the HTML element itself:



Now that we have the ID, we can use it to delete/remove the entire task from the DOM. While we could do that right here in the `taskButtonHandler()` function, it's good practice to follow the separation of concerns and do the actual deletion in its own function. It might seem like we're creating too many functions, but this does make the code easier to maintain and test.

In `script.js`, create a new function that uses `taskId` as a parameter:

```
var deleteTask = function(taskId) {
  console.log(taskId);
};
```

Then call this function from `taskButtonHandler()`:

```
if (event.target.matches(".delete-btn")) {
  var taskId = event.target.getAttribute("data-task-id");
  deleteTask(taskId);
}
```

Save `script.js` and try running this in the browser again by clicking the Delete button on a task. We should see a similar result as before, but now the `console.log()` printing our task item's ID is coming from the `deleteTask()` function instead of our handler function.

HIDE PRO TIP

These incremental checks using `console.log()` statements are a great way to verify that your code is coming together nicely. If a `console.log()` suddenly stops working or prints unexpected results, you don't have to backtrack too far to catch the problem.

Now that we've captured the ID of the task we want to delete, how do we go about actually deleting it? Luckily for us, the `data-task-id` attribute wasn't only applied to the Delete button; it's on the task's `<li>` element as well.

Update the `deleteTask()` function to include this code:

```
var deleteTask = function(taskId) {
  var taskSelected = document.querySelector(".task-item[data-task-id='
  console.log(taskSelected);
};
```

See that we're selecting a list item using `.task-item`, and we're further narrowing the search by looking for a `.task-item` that has a `data-task-id` equal to the argument we've passed into the function. Also notice that there's no space between the `.task-item` and the `[data-task-id]` attribute, which means that both properties must be on the same element; a space would look for a element with the `[data-task-id]` attribute somewhere *inside* a `.task-item element`.

Now save `script.js` and run this code in the browser by clicking on a task item's Delete button.

```
▼<li class="task-item" data-task-id="0">
  ▼<div class="task-info">
     <h3 class="task-title">Sample Task</h3>
     <span class="task-type">Mobile</span>
   </div>
  ▼<div class="task-actions">
     <button class="btn edit-btn" data-task-id="0">Edit</button>
     <button class="btn delete-btn" data-task-id="0">Delete</button>
   ▶<select name="status-change" data-task-id="0" class="select-status">…</select>
   </div>
 </li>
```

Using `querySelector()` with a selector like `.task-item[data-task-id='0']` allowed us to find a different element with the same `data-task-id` attribute. We used a similar method earlier when we wanted to get the values from our form elements. With the form, we selected the element type (`input`) and then specified what attribute we were looking to match (`input[name='task-name']`). Here, we are selecting the element by its class name, and then checking to see if it also has the specific data attribute value.

Now that we have the correct element, let's update `deleteTask()` one more time to actually remove it from the page:

```
var deleteTask = function(taskId) {
  var taskSelected = document.querySelector(".task-item[data-task-id='
  taskSelected.remove();
};
```

Again, save `script.js` and run this code in the browser to test our functionality. When a Delete button is clicked, the corresponding task item should be removed from the page entirely thanks to the aptly named `remove()` method.

Now that we've handled the functionality for capturing button clicks in our task list and deleting them, we can move on to editing a task.