4.3.8 Load the Task into the Form

Editing a task takes two parts:

1. Loading the task's current information into the form.
2. On form submit, updating the task element's content.

Visualizing the problem as two separate issues will keep it from feeling overwhelming! For now, we're only concerned with putting an existing task's data into the form. This happens any time an Edit button is clicked. However, Edit buttons present the same dilemma that Delete buttons did: there are too many of them.

PAUSE

How can you know when an Edit button is clicked?

Use event delegation and methods like `matches()` to check the class name.

Hide Answer

Fortunately, we can leverage the work we did with deleting a task to facilitate this process. Edit buttons already have a `data-task-id` attribute, after all. Editing can also use the same `taskButtonHandler()` handler function as a starting point.

Update `taskButtonHandler()` to accommodate both button clicks:

```
var taskButtonHandler = function(event) {
  // get target element from event
  var targetEl = event.target;

  // edit button was clicked
  if (targetEl.matches(".edit-btn")) {
    var taskId = targetEl.getAttribute("data-task-id");
    editTask(taskId);
  }
  // delete button was clicked
  else if (targetEl.matches(".delete-btn")) {
    var taskId = targetEl.getAttribute("data-task-id");
    deleteTask(taskId);
  }
};
```

Similar to `deleteTask()`, add a new function for editing that creates its own `taskSelected` variables based on the provided `taskId`:

```
var editTask = function(taskId) {
  console.log("editing task #" + taskId);

  // get task list item element
  var taskSelected = document.querySelector(".task-item[data-task-id='
};
```

If you haven't already, save and test the app in the browser. Clicking an Edit button should console log the appropriate task ID.

We've also defined a `taskSelected` variable that references the entire `<li>` element. We don't need everything from this element, though. The only pieces of information we care about are the task's name and type. In the DOM, these are an `<h3>` element and a `<span>` element respectively:

```
<h3 class="task-name">Sample Task</h3>
<span class="task-type">Mobile</span>
```

Hmmm, we didn't add `data-task-id` attributes to these elements, so how do we find them? We already have the parent `<li>` element. We can just use that as a `querySelector()` starting point.

Update the `editTask()` function to look like this:

```
// get task list item element
var taskSelected = document.querySelector(".task-item[data-task-id='"

// get content from task name and type
var taskName = taskSelected.querySelector("h3.task-name").textContent;
console.log(taskName);

var taskType = taskSelected.querySelector("span.task-type").textConter
console.log(taskType);
```

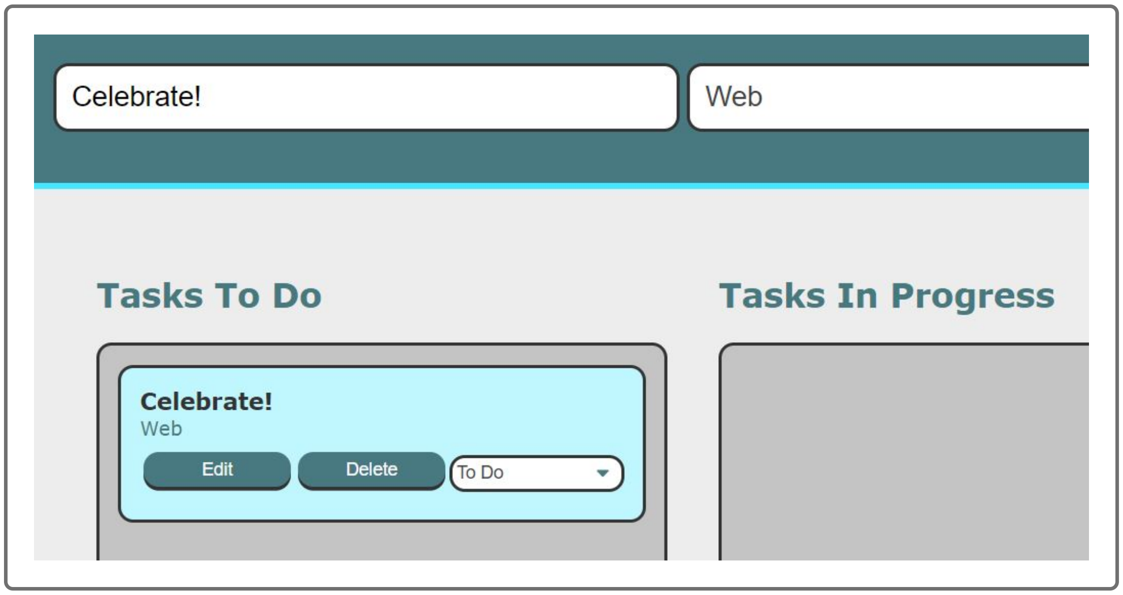
In the past, we've used `querySelector()` with the `document` object, but any DOM element can use this method. `document.querySelector()` searches within the `document` element, which is the entire page, while `taskSelected.querySelector()` only searches within the `tastSelected` element. Thus, we can narrow our search to the task item at hand to find its name (`h3.task-name`) and type (`span.task-type`).

Test the app in the browser to verify if the `console.log()` statements display the correct data.

Now that we have the information we want, we can reuse the selectors from before to update the form. Delete the `console.log()` statements in the function, and then add these lines after the `var taskType` expression:

```
document.querySelector("input[name='task-name']").value = taskName;
document.querySelector("select[name='task-type']").value = taskType;
```

Save the `script.js` file again and refresh the browser. After creating a task, click the Edit button. If all's well, the task's name and type should appear in the form inputs:



To make it clear to the user that the form is now in "edit mode," we should also update the text of the submit button.

Add this line to `editTask()`:

```
document.querySelector("#save-task").textContent = "Save Task";
```

In the browser, the button text will change once an Edit button has been clicked:

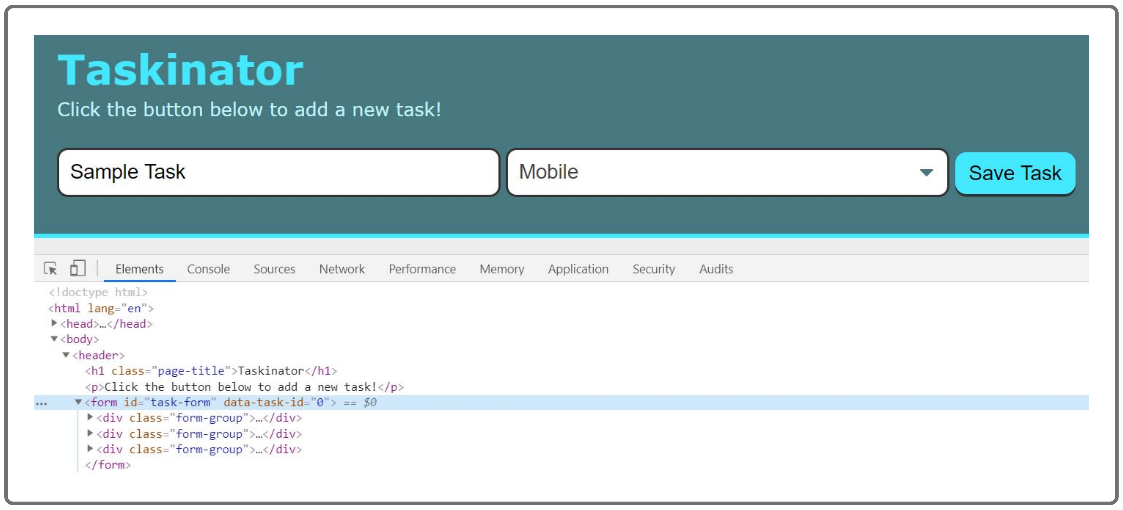
This is a nice UI improvement for the user, but we'll also need some way for us, the developer, to know which task is currently being edited. So far, we've only added the task's name and type to the form. The ID is lost, meaning when the user presses Save Task, where does that information go?

Let's make one more addition to the `editTask()` function:

```
formEl.setAttribute("data-task-id", taskId);
```

This will add the `taskId` to a `data-task-id` attribute on the form itself. It's a new attribute that users won't see but that we can use later on to save the correct task.

Test out the app again and make sure the `<form>` element in the DevTools receives a `data-task-id` attribute with the right value:


Taskinator
Click the button below to add a new task!
Sample Task
Mobile
Save Task
Elements Console Sources Network Performance Memory Application Security Audits

```
<!doctype html>
<html lang="en">
 ▶<head>…</head>
 ▼<body>
   ▼<header>
      <h1 class="page-title">Taskinator</h1>
      <p>Click the button below to add a new task!</p>
...  ▼<form id="task-form" data-task-id="0"> == $0
       ▶<div class="form-group">…</div>
       ▶<div class="form-group">…</div>
       ▶<div class="form-group">…</div>
      </form>
```

Congratulations, we're halfway through this new feature! We haven't finished the ability to edit a task, of course, but this is a good milestone that's worth committing with Git.