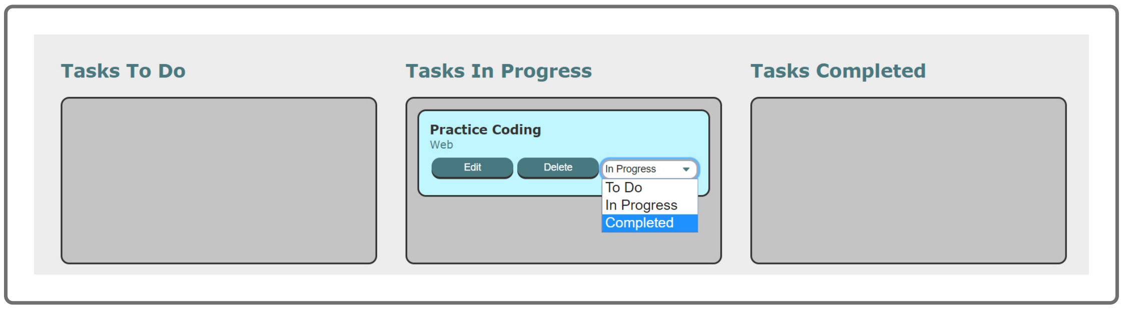
4.3.10 Move the Task Based on Status

There's still one last method in which a task can be updated. Tasks have a status—To Do, In Progress, Completed—that isn't updated through the main form but rather through individual `<select>` elements. Changing the value of the `<select>` should automatically move the task from one column to another, as you can see here:



Since we'll be interacting with the remaining two columns, add two new variables at the top of `script.js` to reference them:

```
var tasksInProgressEl = document.querySelector("#tasks-in-progress");
var tasksCompletedEl = document.querySelector("#tasks-completed");
```

We technically already have a click event listener set up for the tasks' `<select>` elements thanks to the event delegation of `taskButtonHandler()`, which listens to events on the entire `<main>` element of the document. This click event doesn't really help us here, though, because a `<select>` click only fires on the initial click. Depending on the browser, the second click to choose an option fires on the `<option>` element instead of the `<select>` element. That sounds like a bigger headache to sort out than its worth.

Thankfully, there are more events besides click and submit that we can tap into. There's also a change event that triggers, as the name implies, any time a form element's value changes. Like our Delete and Edit clicks, though, we'll need to delegate this event since we have multiple elements that it applies to.

At the bottom of `script.js`, add a new event listener:

```
pageContentEl.addEventListener("change", taskStatusChangeHandler);
```

Alongside your other functions, define a new function to handle this event:

```
var taskStatusChangeHandler = function(event) {

};
```

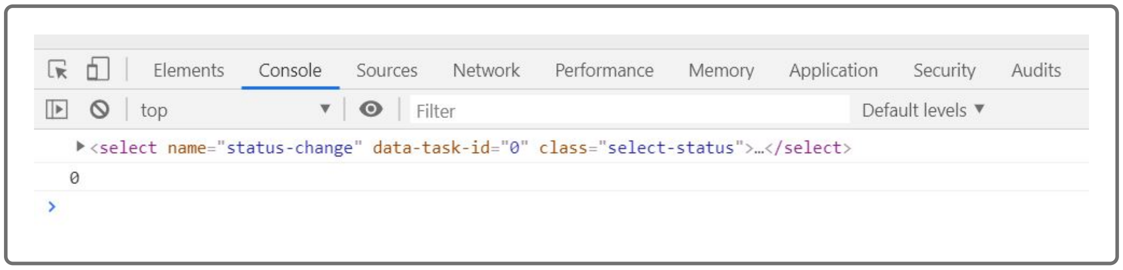
PAUSE

On the `event` object, how do you get the element that triggered the event?

`event.target`

Hide Answer

In the `taskStatusChangeHandler()` function, console log `event.target` and `event.target.getAttribute("data-task-id")`. When you click on and change the value of the `<select>` element, you should see the following in the DevTools:



```
▸<select name="status-change" data-task-id="0" class="select-status">…</select>
0
>
```

`event.target` is a reference to a `<select>` element, meaning we can use additional DOM methods to get this element's properties. We'll want the ID so we can find the overall task item, and we'll want the value of the `<select>` element to know which other column to move it to.

Update the function to create the following variables:

```
var taskStatusChangeHandler = function(event) {
  // get the task item's id
  var taskId = event.target.getAttribute("data-task-id");

  // get the currently selected option's value and convert to lowercas
  var statusValue = event.target.value.toLowerCase();

  // find the parent task item element based on the id
  var taskSelected = document.querySelector(".task-item[data-task-id='
};
```

Converting the value to lowercase might seem unnecessary, but it helps future proof the app in case we ever changed how the status text is displayed. Code-wise, we know we'll always check against the lowercase version. Based on whatever the value is, we can move the `taskSelected` element from one column/list to another.

Add the following `if` statements to the end of the `taskStatusChangeHandler()` function:

```
if (statusValue === "to do") {
  tasksToDoEl.appendChild(taskSelected);
}
else if (statusValue === "in progress") {
  tasksInProgressEl.appendChild(taskSelected);
}
else if (statusValue === "completed") {
  tasksCompletedEl.appendChild(taskSelected);
}
```

Remember that `tasksToDoEl`, `tasksInProgressEl`, and `tasksCompletedEl` are references to the `<ul>` elements created earlier. Thus, if the user selects "In Progress" from the dropdown, it will append the current task item to the `<ul id="tasks-in-progress">` element with the `tasksInProgressEl.appendChild(taskSelected)` method.

The interesting thing about the use of `appendChild()` here is that it didn't create a copy of the task. It actually moved the task item from its original location in the DOM into the other `<ul>`. It's important to note that the variable `taskSelected` didn't create a second `<li>`. That would only be the case if we used `document.createElement()`. Instead, it's a reference to an existing DOM element, and we simply appended that existing element somewhere else.

Save and test the app in the browser to make sure the tasks move to the correct column/list when the dropdown changes. Now would be a good time to make a "Master JavaScript" task and move it to Completed!