4.4.8 Improve the UI with Dragleave Event

Congratulations on completing the drag-and-drop feature! It functions well, but after playing with it, we sensed the need for an improvement. Wouldn't it be nice if the user could see where the dragged element could be dropped? Let's change the border to a dashed line and highlight the background color upon this event.

To do this, we'll add some CSS properties to the task list element. As we've done several times in this lesson, we'll use the `event.target` property in combination with the `closest()` method to find the task list element or the `<ul>`. Can you guess which event we need to listen for— `dragstart`, `dragover`, or `drop`?

If you guessed the `dragover` event, you're correct, and here's why:

- We need access to the element being dragged over in the `event.target` property.
- We want the `style` property to change just as the dragged element is being dragged over the task list element.

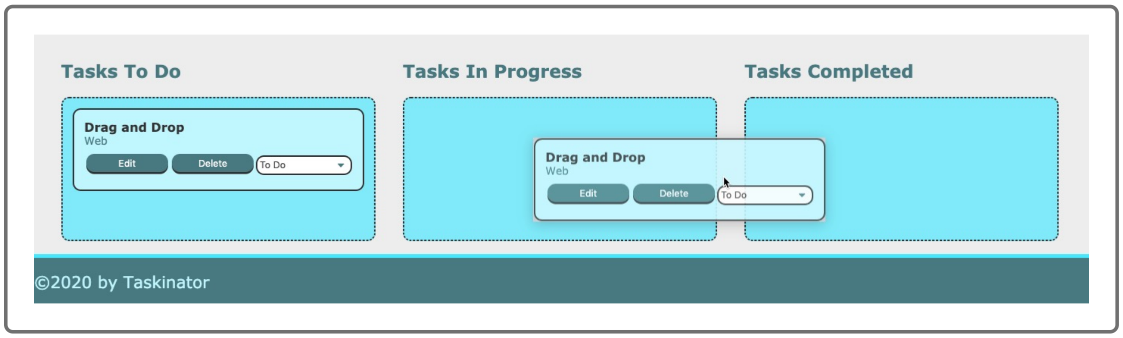
In the next step, because we know we'll be dealing with the `dragover` event listener, we'll add our style changes to the `dropZoneDragHandler`.

We'll use the `setAttribute()` method to add these to the selected task list.

Add the following expression to the `dropZoneDragHandler` inside the conditional statement beneath the `event.preventDefault();`:

```
taskListEl.setAttribute("style", "background: rgba(68, 233, 255, 0.7);
```

Save the file and refresh the browser to verify the code. You should see something similar to this:



As you can see, on the `dragover` event, the task list elements change and the style properties are added correctly to the task list element that was hovered over with the task item. However, if we continue to drag our task item over other task lists, they remain highlighted and dashed. Even the task list where we drop the item will retain the style attributes we assigned to it while dragging over it.

If the point of the extra styles is to indicate to the user where the task item could be dropped, then the styles should only appear when a task list is actively being hovered over by the task item. Once the task item is dropped or has moved to another task list, these style properties should be removed.

Can you find a method that can remove attributes? Let's check Google to see if there are some good options for us to choose.

HIDE HINT

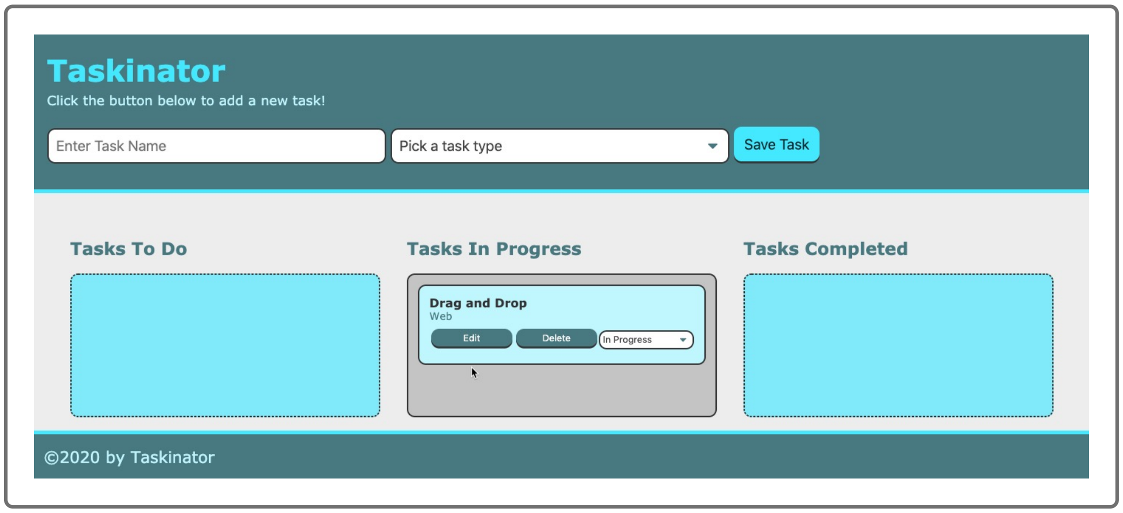
A good search term would be "remove attribute javascript".

Looking through the MDN web docs on JavaScript, we will find the `removeAttribute()` method. This method can remove the `style` attribute that we added in the `dragover` event. Let's add this to our `drop` event because we know that on a drop, these style properties should disappear.

Add this expression near the bottom of the `dropTaskHandler()` function above the append operation so the style removal can happen just before the task item is attached to the new task list.

```
dropZoneEl.removeAttribute("style");
```

Save and refresh the browser to see if the `style` attribute was removed from the task list element on the `drop` event:



Congrats! You've successfully removed the style attribute on the `drop` event.

Now we only need to remove the `style` attribute if the task item is dragged to another task list. It makes sense that if the style is indicating where the task item can be dropped, only one task list should be highlighted at a time. It doesn't appear that we can use any of the event listeners we currently have in our app because we are looking for a specific condition when the element is leaving the drop zone (in our case, the task list).

Luckily, we have an event aptly named `dragleave`. We'll need this event listener on each of the three task lists. Can you figure out how to do this without adding three listeners? If you said event delegation, you are correct! Similar to our other event listeners, we'll add the `dragleave` event listener to the `pageContentEl` element, which is the parent element to our three task lists.

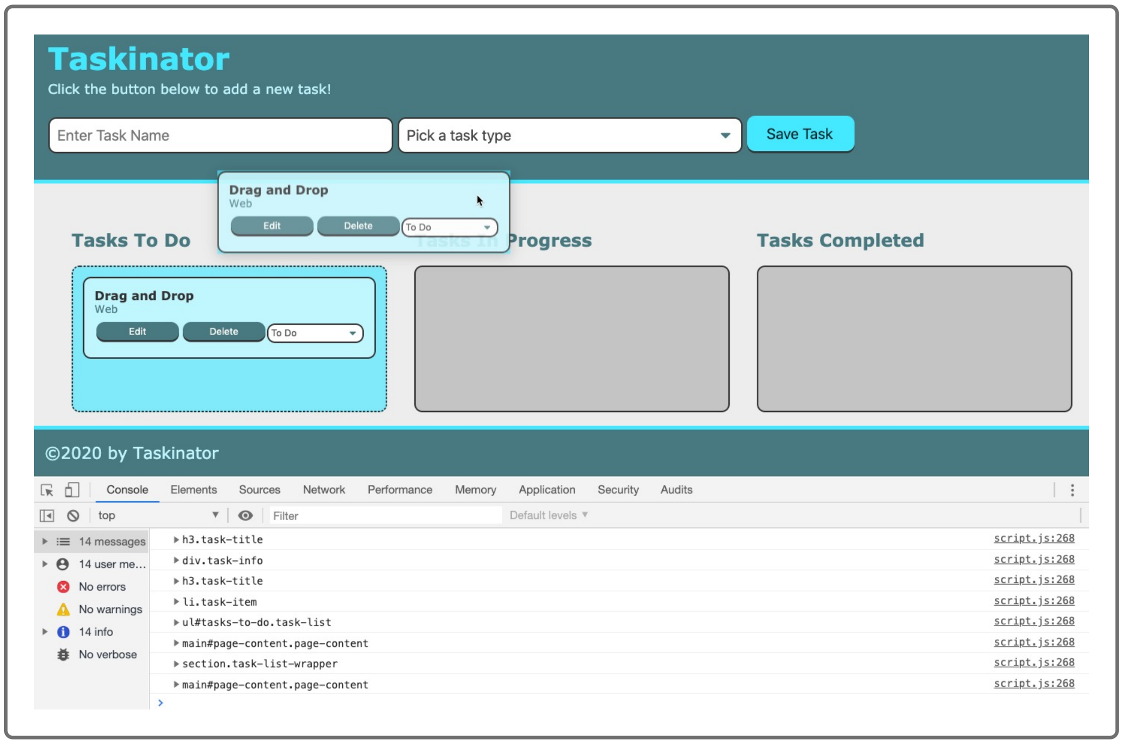
Let's add this statement to the bottom of the `script.js` file and include the callback function, `dragLeaveHandler()`, to execute on the event being triggered:

```
pageContentEl.addEventListener("dragleave", dragLeaveHandler);
```

Now let's define the `dragLeaveHandler()` by adding the following function expression above the event listeners in the event handler section of the `script.js` file:

```
var dragLeaveHandler = function(event) {
  console.dir(event.target);
}
```

Save the file and refresh the browser, then add a task and drag it over a task list to see the following results:



```
▸h3.task-title                                    script.js:268
▸div.task-info                                    script.js:268
▸h3.task-title                                    script.js:268
▸li.task-item                                     script.js:268
▸ul#tasks-to-do.task-list                         script.js:268
▸main#page-content.page-content                   script.js:268
▸section.task-list-wrapper                        script.js:268
▸main#page-content.page-content                   script.js:268
```

Let's explain the different DOM elements that are being displayed in the console in the image above. The `dragleave` target property is storing every element that the dragged task item has been dragged over but is no longer actively being dragged on. Once the dragged element has left the drag-over state, the target element is captured.

Let's remove this `console.dir()`, then use the same conditional statement we used previously in the `dropZoneDragHandler()` to narrow our `dragleaveTaskHandler()` to execute only when a dragged element leaves a task list or children of the task list, not every element on the page. To do this, let's first make a reference to the `target` property of the `dragleave` event, which is the task list that's being dragged over. Then add a conditional to check if the element we're drag-leaving is either the task list or a descendent of the task list. If it is, then remove the style attribute from it.

REWIND

If `closest()` returns null, then the selector was not found in the target element or an ancestor element, and the `style` attribute is not removed.

Here's how this code should look:

```
var taskListEl = event.target.closest(".task-list");
if (taskListEl) {
  taskListEl.removeAttribute("style");
}
```

This final step didn't seem as critical to the drag-and-drop process, but adding convenience measures to improve the user experience is an important metric when determining how user-friendly a website is.

We're done! Next, we'll make sure to commit our files to Git and GitHub.