4.1.7 Capture the Button Click

You're almost done with this lesson. You've got this!

In the last step, we used the DOM to find the element object representation of the `<button>`. Now we need a way to observe the user's click of the `<button>`. Then we'll create a response from the button click, which will execute the operation of adding a task item to the task list. We must break down the process step-by-step to solve this problem.

We want to observe the click behavior specific to the `<button>`. We don't want the button-click response to occur if users click on the `<body>` or any other element in the document, because then we might inadvertently add tasks to the list—not a good user experience.

In web development, we refer to user behavior—the click, in this case—as an **event**. We refer to the observation of the event as the **event listener**. And we refer to the response to the event as the **event handler**.

There are many types of events because many user behaviors can happen on a webpage. Over the years, the number of events has grown in order to capture the interactions of the user and the webpage. Pointing the cursor at something (e.g. hovering over it with the mouse), clicking keys on the keyboard, and scrolling down the webpage are a few examples. We can even define events at the start or end of an action, like when a key is pressed or released.

DEEP DIVE ▲

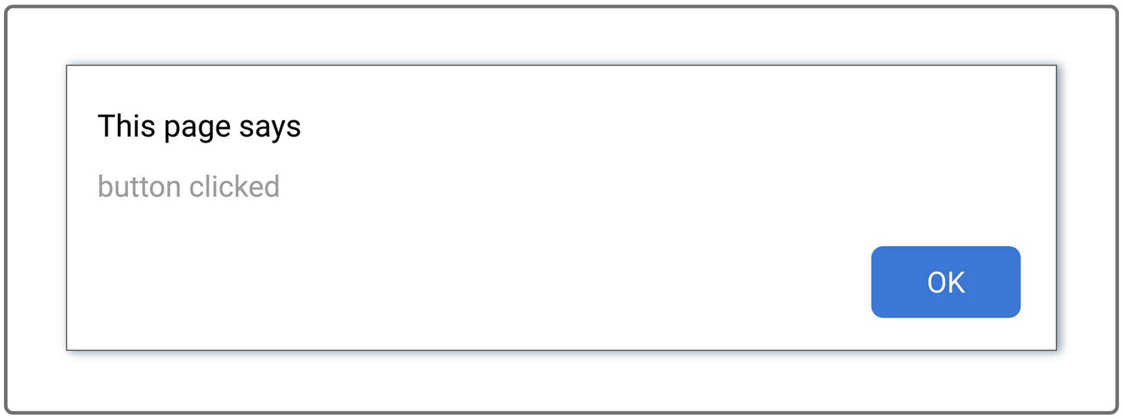
DEEP DIVE

For more information, see the **MDN web docs on all events possible (https://developer.mozilla.org/en-US/docs/Web/API/Element#Events)**.

Now it's time to get your hands dirty. Type the following into the console of the browser that has your `index.html` file:

```
buttonEl.addEventListener("click", function() {
  alert("button clicked");
});
```

Click the button to see what happens. You should see the following screenshot in the dialog box:



This page says

button clicked

Let's break down the expression to understand the concepts being demonstrated:

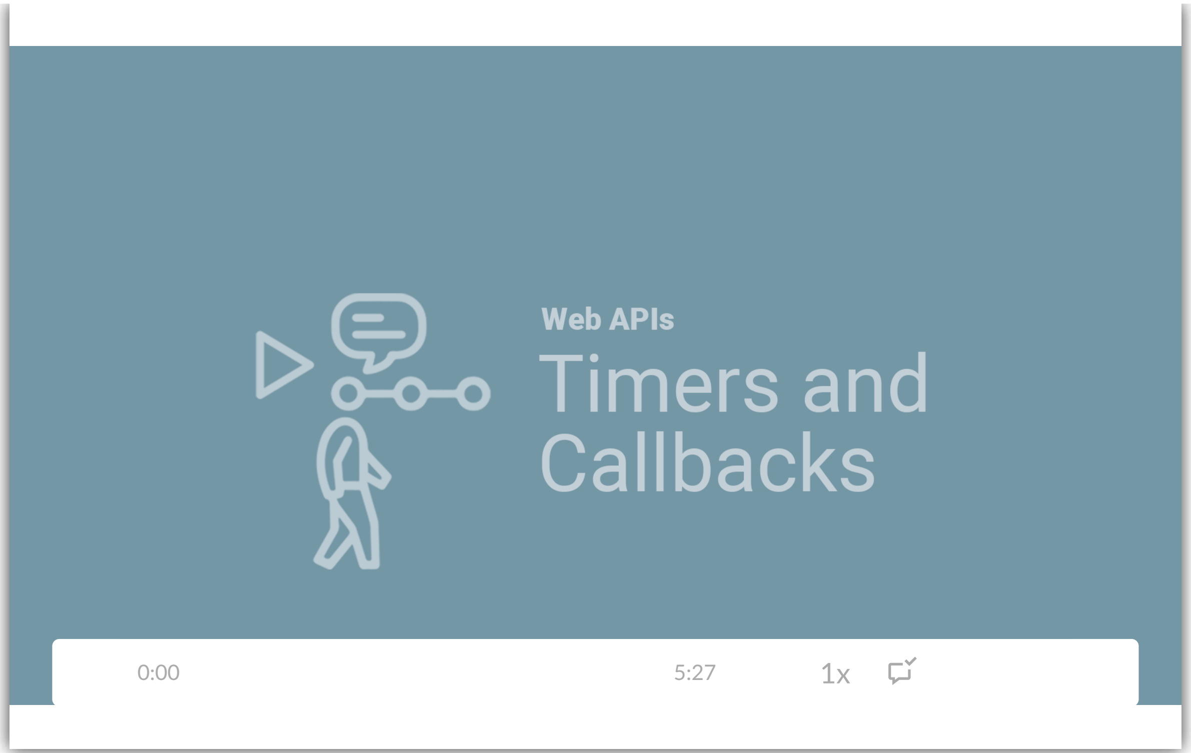
- The `addEventListener()` method can be used by an element object—in this case, the `buttonEl` object. This method adds an event listener to the `<button>` element. The `buttonEl` object was established in the previous step as the object representation of the `<button>` element.

- We pass two arguments into the `addEventListener()` function: the type of event we'll listen for—in our case, the click event—and the event response to execute once the event has been triggered. Here we've used an anonymous function that uses the window method `alert()`. An anonymous function doesn't have a named identifier and therefore can't be called outside the context of this expression. It only exists within the current context.

Introduction to the Callback

Let's dive a bit deeper into the second argument, as passing a function into a function may be confusing at first glance. This is called a **callback** function. This is one of the foundational pillars that makes JavaScript unique. In layperson's terms, we're passing a function to another function to execute. The callback function will execute once the event is triggered. In our case, the anonymous function will execute once a button click registers.

Callback functions are used in many other aspects of JavaScript, including when you want a block of code to execute after a given number of seconds have passed. To see this idea in action, watch the following video:



So far we've used the DOM to target the button object, then attached an event listener to this element object to capture the user's button click. The next crucial step will be to change the code block in the event listener to execute the job of adding a task item to the task list. Let's proceed with this step now.