JavaScript Event Listeners Lab

- Due No Due Date
- Points 1
- Submitting a website url

(https://github.com/learn-co-curriculum/phase-0-javascript-events-event-listening-lab) (https://github.com/learn-co-curriculum/phase-0-javascript-events-event-listening-lab/issues/new)

Learning Goals

- Create event listeners on DOM nodes using `addEventListener()`

Introduction

In this lab we will learn how to teach nodes to "listen" for an event using `addEventListener()`.

If you haven't already, **fork and clone** this lab into your local environment. Navigate into its directory in the terminal, then run `code .` to open the files in Visual Studio Code.

Create Event Listeners on DOM Nodes with `addEventListener()`

In order for JavaScript to handle an event, we first need to tell it to listen for that event. We do this by calling the `addEventListener()` method on the element we want to add the listener to, and passing it two arguments:

1. the name of the event to listen for, and
2. a *callback function* to "handle" the event

Open up `index.html` in the browser. When you click in the `<input>` area, nothing happens. Let's set up some *event handling*. Specifically, let's add an event listener for the `click` event on the `input#button` element in `index.html`.

Try out the following in the Chrome DevTools console:

```
const input = document.getElementById('button');
input.addEventListener('click', function() {
  alert('I was clicked!');
});
```

Now when you click inside of `input#button`, you will get an alert box.

Let's review what's happening in this code.

First, we grab the element that we want to add the event listener to and save a reference to it in the `input` variable.

Next, we call `addEventListener()` on that element to tell JavaScript to listen for the event. We pass two arguments to `addEventListener()`: the name of the event to listen for (in this case, `click`) and a *callback function* that will be executed when the event is "heard."

According to MDN (https://developer.mozilla.org/en-US/docs/Glossary/Callback_function):

> A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

That's exactly what's happening here: we're passing a callback function as the second argument to the `addEventListener()` function; the callback will be invoked as soon as the event occurs.

Let's pull out that second argument and take a look at it:

```
function() {
  alert('I was clicked!');
}
```

This function has all the components of functions we've seen before (the `function` keyword, a pair of parentheses, and the body of the function enclosed in curly braces) *except one*: it doesn't have a name assigned to it. This is what's called an *anonymous* function. Because it doesn't have a name, it can't be invoked directly. But the event listener knows to execute whatever function is passed as the second argument when it detects the event, so it doesn't need to be named.

If we are only calling our callback function in that one place, using an anonymous function makes sense. However, what if we wanted to use that same alert message on a bunch of elements? In that case, it would make more sense to create a separate, named function that could be called by all of our event listeners. With this approach, we would pass the *function name* as the second argument to `addEventListener()` rather than the function itself:

```
const input = document.getElementById('button');

function clickAlert() {
  alert('I was clicked!');
}

input.addEventListener('click', clickAlert);
```

We could then attach our `clickAlert` to as many elements as we'd like. Just as we did for the `input` element, we would first use our CSS selector skills to grab the desired element and save it to a variable, then add the `click` event listener to that element. Give it a try!

With this approach, even if we're using our `clickAlert` with a whole bunch of elements, if we decide later that we want to change the text of the alert to "Hee hee, that tickles!" instead, we would only need to make that change in one place: inside our `clickAlert()` function.

Note: we pass `clickAlert` as the argument, not `clickAlert()`. This is because we don't want to *invoke* the function in this line of code. Instead, we want to pass a *reference* to the function to `addEventListener()` so *it* can call the function when the time comes.

Refresh your browser and try out the latest version of the code in the console to verify that it works. Also try passing `clickAlert()` as the second argument rather than `clickAlert` and see what happens.

Passing the Tests

Now let's set up `index.js` to do the same thing so we can get our test passing. To do that, simply copy the code into the `index.js` file's `addingEventListener()` function and run the test. Either version should pass the test — just make sure that the code creating the event listener is **inside** the `addingEventListener()` function.

Checking the Code in the Browser

We know that the code works in the console and passes the test, but we should also check our changes to `index.js` in the browser. Because you've added the `addEventListener()` function *inside* the `addingEventListener()` function, recall that you will need to call the outer function in `index.js` to execute `addEventListener()` and activate the event listener. Be sure to refresh the page to load the new code in `index.js`.

Resources

- **MDN - Web Events (https://developer.mozilla.org/en-US/docs/Web/Events)**