Lesson Wednesday

Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / Removing Event Listeners

Text Cheat sheet

In this lesson, we'll learn how to remove event listeners with the `removeEventListener()` method. In the applications that you build at Epicodus, there won't be a lot of cases in which we'll need to remove event listeners, but it is helpful to know when and why we might want or need to do that.

Removing Event Listeners with `removeEventListener()`

There is a method to remove an event listener, just like there is a method for adding an event listener. Let's see how it works. First, we'll set up a click event listener on a heading element. We'll use the same example from the previous lesson of an educational application that explains HTML and what each element on the page is called. Here's a snippet of our HTML:

```
<h1>Heading 1</h1>
```

And here is our event listener:

```
function alertHeading() {
  window.alert("This is a heading element.");
}
let h1 = document.querySelector("h1");
h1.addEventListener("click", alertHeading);
```

Notice that we're using a function declaration called `alertHeading()` as the handler function for the click event. `alertHeading()` is also called a callback function, because it is a function that is being used as an argument in another function. Because we're passing it in as the handler function for our click event listener, `alertHeading()` will be called any time the user clicks on the H1 element.

When we pass in `alertHeading()` to the `addEventListener()` method, we purposefully don't include parens so that we can pass in the definition of the function to the `addEventListener()` method. If we included parens then we'd be calling it now, and we don't want that. We want the `addEventListener()` method to be able to call the `alertHeading()` function only when it needs to, when the click on the H1 element happens.

Now let's remove the click event listener:

```
function alertHeading() {
  window.alert("This is a heading element.");
}
let h1 = document.querySelector("h1");
h1.addEventListener("click", alertHeading);
h1.removeEventListener("click", alertHeading);  // this lin
e is new!
```

Right after we create the event listener we now remove it with `h1.removeEventListener("click", alertHeading);`. Notice that we pass in **the exact same arguments** as we do into the `addEventListener()` method. This is by design and how the `removeEventListener()` method knows which event listener to remove.

Let's break this down the `removeEventListener()` method:

- This method takes two arguments:
  - The name of the event. In our example the event we're targeting is `"click"`.
  - The name of the function we are using to handle the event. This function is often called the "handler function". In our example, this handler function is `alertHeading`. (This function is also categorized as a callback function.)
- **The two arguments we supply to the `removeEventListener()` method call MUST match the same arguments that we supplied to the `addEventListener()` method that originally created the event listener that we want to remove.**

This last point is so important that we should look at scenarios of removing event listeners that fail.

(Some) Function Expressions Don't Work with `removeEventListener()`

When we set up an event listener with a function expression that's not saved to a variable, we can't target and remove it with `removeEventListener()`. Let's see an example. In the following code snippet, we will fail to remove the event listener, even though the arguments in both `addEventListener()` and `removeEventListener()` are exactly the same.

```
let h1 = document.querySelector("h1");
h1.addEventListener("click", function() {
  window.alert("This is a heading element.");
});
h1.removeEventListener("click", function() {
  window.alert("This is a heading element.");
});
```

In JavaScript, anonymous function expressions that are not stored into a variable are not identical even if they are defined using the same unchanging source-code. So even though each callback function has the exact same code:

```
function() {
  window.alert("This is a heading element.");
}
```

JavaScript considers them as two separate functions, and because of this, the `removeEventListener()` method can't match its arguments to the ones we passed into the `addEventListener()` method, and fails to find and remove the correct event listener.

We can still use function expressions that are stored in a variable, because the variable unequivocally represents one value, the function expression. The code snippet below demonstrates this.

```
const alertHeading = function() {
  window.alert("This is a heading element.");
};
let h1 = document.querySelector("h1");
h1.addEventListener("click", alertheading);
h1.removeEventListener("click", alertHeading);
```

So, just like when using function declarations, JavaScript knows that we're referencing the same function with the `alertHeading` variable. The arguments for both methods will match and enable the `removeEventListener()` method to find and remove the correct event listener.

When and Why We Remove Event Listeners

We remove event listeners in three cases:

- When we only want an event handler to run once. We can remove an event listener after it has been run to make it so the reaction happens once (or a different set number of times).
- To reuse elements. Being able to add and remove event listeners allows us to use one element in multiple scenarios. For example, we could use one button in three different scenarios by adding and removing event listeners. The alternative would be to use three separate buttons with three separate event listeners.
- To improve performance or efficiency. If we have many, many event listeners in an application, removing the ones that are no longer needed can improve efficiency. This aspect of efficiency has to do with memory management. We won't learn about performance optimization at Epicodus, but this is an important topic to learn about down the road as you continue your growth as a developer.

In terms of a workflow for removing event listeners, a good guideline is to remove event listeners when you've removed the target it belongs to. Not just hide the target, but remove it completely from the DOM. We haven't yet learned how to add and remove elements from the DOM, but we will in the next course section.

There are cases where JavaScript's automatic "garbage collection" handles removing event listeners from the associated targets when they are removed from the DOM, but that is a subject that is not important to understand right now. If you want to learn about "garbage collection" and memory management, visit this reference page on MDN (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management). If you want to see a practical example of a memory issue in relation to event listeners, review this section on MDN's `addEventListener()` reference page (https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener#memory_issues).

Again, memory management and performance are topics that you should learn about after you leave Epicodus as part of your continuing education. The applications we build at Epicodus for the most part will be small and simple, and won't require cleaning up old, unused event handlers. The most common reason that you will remove an event listener is to reuse an HTML element in multiple scenarios/for different actions, or to react to an event only once.

A Practical Example

Let's review a more realistic example of removing an event listener.

In the lesson "Event Handling with Event Listeners" we expanded the functionality of our Mad Libs project to not only show the Mad Libs story (with user inputted values), but also to display an advertisement and a button to reset the form. Let's update the event handling that displays the advertisement so that it only runs

once. To make the code easier to follow, we'll first highlight the code just for the advertisement, and then we'll show all of the HTML and JS afterwards.

First, let's look only the code that creates the event listener on the form. We've omitted other code with ellipses `...`:

```
// User Interface Logic
window.addEventListener("load", function() {
  ...
  let form = document.querySelector("form");
  ...
  form.addEventListener("submit", function() {
    window.alert("Do you need a new computer? Visit www.sup
erextracomputersales.com to find the best deals!");
  });
  ...
});
```

In order to use the `removeEventListener()` method, we need to use a function declaration, or store our function expression into a variable that we can call on. Let's make that update first:

```
// User Interface Logic

// new function!
function advertisement() {
  window.alert("Do you need a new computer? Visit www.super
extracomputersales.com to find the best deals!");
}

window.addEventListener("load", function() {
  ...
  let form = document.querySelector("form");
  ...
  form.addEventListener("submit", advertisement);
  ...
});
```

We've added a new function definition called `advertisement()` that displays our ad. Take note of a few things about the placement of the `advertisement()` function:

- This is considered user interface logic, because it handles accessing, manipulating or updating the DOM. Code that does not access or update the DOM is considered business logic (we'll see more examples of this in an upcoming lesson).
- We define `advertisement()` outside of the `window.onload` event handler. We only need to *use* the function inside of the `window.onload` event handler.
- We locate the definition of `advertisement()` above where we use that function in our scripts.

Next, let's incorporate the `removeEventListener()` method. We'll want to add this method into the new `advertisement()` function so that we remove the event listener only after it has run once.

```
// User Interface Logic
function advertisement() {
  window.alert("Do you need a new computer? Visit www.super
extracomputersales.com to find the best deals!");
  // new line below!
  document.querySelector("form").removeEventListener("submi
t", advertisement);
}

window.addEventListener("load", function() {
  ...
  let form = document.querySelector("form");
  ...
  form.addEventListener("submit", advertisement);
  ...
});
```

The `form` variable that we created in the `window`'s load event listener is not available outside of that event listener. In other words, the `form` variable is "scoped" to the `window`'s load event listener. So, within the `advertisement()` function, we need to first use `document.querySelector("form")` to get the `HTMLFormElement` object, and then call the `removeEventListener()` method on it. We could separate this code onto two lines if that's easier to read and understand:

```
let form = document.querySelector("form")
form.removeEventListener("submit", advertisement);
```

And with that new line, we've successfully removed the event listener for the advertisement such that it will only display once and be removed. Take note that the issues of scope for the `form` variable does not exist for event listeners. We can remove an event listener in one function that is originally defined in another.

If you are feeling iffy about scope, don't worry, it *is* a tricky concept to understand. We'll be exploring scope again in an upcoming lesson, so there will be more opportunity to practice soon.

Completed Mad Libs Code

Check out the cheat sheet for this lesson for the completed HTML and JS for the Mad Libs project. This includes the update to only run the advertisement once. If you want to, you are welcome to use the code to recreate this project to test out the changes we've made to the advertisement's functionality. In the next lesson, you'll practice using `addEventListener()` and `removeEventListener()`.

Summary

In this lesson we learned how to remove event listeners with the `removeEventListener()` method. To use this method successfully, there's a few things to note:

- We must use a function declaration or a function expression that's stored into a variable when we create and remove an event listener.
- The two arguments we supply to the `removeEventListener()` method call MUST match the same arguments that we supplied to the `addEventListener()` method that originally created the event listener that we want to remove.
- The most common reason that you will remove an event listener is to reuse an HTML element in multiple scenarios/for different actions, or to react to an event only once.
- Issues related to scope that affect variables do not affect event listeners. We can remove an event listener in one function (or scope) that is originally defined in another function (or scope).

Previous (/introduction-to-programming/javascript-and-web-browsers/using-function-declarations-in-event-handling)
Next (/introduction-to-programming/javascript-and-web-browsers/practice-event-listeners)

Lesson 65 of 75
Last updated March 24, 2023

disable dark mode

(http://www.epicodus.com)