Lesson Wednesday

Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / Using Function Declarations in Event Handling

Text Cheat sheet

In this lesson we'll learn how to use function declarations in event handling. We'll look at examples with event handler properties and event listeners. We'll also learn that using function declarations really improves code reusability and is generally considered a good choice for applications that scale. In the next lesson when we learn how to remove event listeners, we will need to use function declarations.

Using Function Declarations with Event Handler Properties

Let's see how to use a function declaration in an `onclick` event handler property. We'll start by reviewing how to use an "onevent" property with a function expression. Below is the JS and a snippet of the HTML that we'll target. This code isn't from a specific project, just a small example to work with.

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

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

Now let's use a function declaration instead. Check out the updated JS:

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

Notice that we only set the name of the function declaration as the value of the `onclick` event handler property: `h1.onclick = alertHeading;`. This is a good reminder of the distinctions within function syntax between defining a function, calling a function, and printing the value/definition of a function. Let's review these distinctions with an example. Check out the following code snippets and pay attention to the comments. Optionally try out the following code in your DevTools console.

```
// we are defining a function with a function declaration
> function alertHeading() {
  window.alert("This is a heading element.");
}
```

```
// we are calling the alertHeading() function
> alertHeading();
```

```
// this is printing the value of a function
> alertHeading;
ƒ alertHeading() {
  window.alert("This is a heading element.");
}
```

When we call on the name of a function without parens, we get the value of the function's definition returned to us. This is just like accessing a variable name to get its value:

```
> let myAge = 55;
> myAge;
55
```

So, when we want to set the value of an object property to a function declaration that we've already defined, we need to only use the name of the function:

```
> h1.onclick = alertHeading;
```

If this concept is unclear, try out the code in the DevTools console, and talk about it with your pair, dev team, or teacher.

Using Function Declarations with Event Listeners

The syntax to use a function declaration with an event listener is much the same as using one with an event handler property. In the following code snippets, we've updated the example `onclick` event handler property to use an event listener instead. The first code snippet uses a function expression and the section code snippet uses a function declaration.

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

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

Just like with the event handler property example, we pass in the name of the `alertHeading` function that we are passing in as the second argument to the `addEventListner()` method call. Remember, a function that is passed in as the argument to another function or method is called a callback function. If you need a review of callback functions, revisit the "Event Handling with Event Listeners" lesson.

When Would I Use a Function Declaration in an Event Handler?

This is a great question. We prefer to reuse functionality by using function declarations or function expressions that are stored in variables, and we prefer to use function declarations because of hoisting. Let's look at an example for the first reason, and then cover the second.

Writing Functions So They Are Reusable

To understand how functions can improve code reusability, let's expand on our example in this lesson. Now let's say we are coding an educational website that teaches students the names of different HTML elements. For our HTML, we'll list all heading elements H1 through H6:

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
```

Now, anytime the user clicks on a heading element, we'll inform them `"This is a heading element."` in an alert. With the event handling we've learned so far, our JS would need to have a click event handler set up for each element. This is what our code would look like using event listeners:

```
let h1 = document.querySelector("h1");
let h2 = document.querySelector("h2");
let h3 = document.querySelector("h3");
let h4 = document.querySelector("h4");
let h5 = document.querySelector("h5");
let h6 = document.querySelector("h6");

h1.addEventListener("click", function() {
  window.alert("This is a heading element.");
});
h2.addEventListener("click", function() {
  window.alert("This is a heading element.");
});
h3.addEventListener("click", function() {
  window.alert("This is a heading element.");
});
h4.addEventListener("click", function() {
  window.alert("This is a heading element.");
});
h5.addEventListener("click", function() {
  window.alert("This is a heading element.");
});
h6.addEventListener("click", function() {
  window.alert("This is a heading element.");
});
```

However, we can drastically reduce the amount of code by using a function declaration or a function expression stored in a variable. Let's see what this looks like with a function declaration:

```
let h1 = document.querySelector("h1");
let h2 = document.querySelector("h2");
let h3 = document.querySelector("h3");
let h4 = document.querySelector("h4");
let h5 = document.querySelector("h5");
let h6 = document.querySelector("h6");

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

h1.addEventListener("click", alertHeadings);
h2.addEventListener("click", alertHeadings);
h3.addEventListener("click", alertHeadings);
h4.addEventListener("click", alertHeadings);
h5.addEventListener("click", alertHeadings);
h6.addEventListener("click", alertHeadings);
```

With this refactor, we've gone from 24 lines of code down to 15, which is a good sign that we've improved our code reusability. Instead of repeating the `window.alert("This is a heading element.");` in six different function expressions, we've opted to create just one function declaration and reuse it for each of the headings. When thinking about how to optimize our code a good guideline is to look for repeated code and add it to a single function that you can then call in multiple places.

Now, which code snippet do you think is easier to read and understand? There's actually no right answer here. I would pick the second option because it's easier to pick out the function definition, and the event listeners are all on single lines, which is easier for me to scan.

By the way, this code works the exact same when we store an anonymous function expression into a variable.

```
let h1 = document.querySelector("h1");
let h2 = document.querySelector("h2");
let h3 = document.querySelector("h3");
let h4 = document.querySelector("h4");
let h5 = document.querySelector("h5");
let h6 = document.querySelector("h6");

const alertHeadings = function() {
  window.alert("This is a heading element.");
}

h1.addEventListener("click", alertHeadings);
h2.addEventListener("click", alertHeadings);
h3.addEventListener("click", alertHeadings);
h4.addEventListener("click", alertHeadings);
h5.addEventListener("click", alertHeadings);
h6.addEventListener("click", alertHeadings);
```

So why would we pick a function declaration over a function expression? Generally speaking, we prefer to use function declarations because that code is **hoisted**. We learned about hoisting in a previous lesson, but now is a good time to review.

A Review of Hoisting

JS interpreters (programs that interpret JS code, like Google Chrome's V8 JS engine) perform **hoisting**, which is the process of moving the declaration of functions and variables to the top of their scope, prior to execution of the code. Let's understand this with an example. Copy and paste this code into your DevTools console

```
const result = add(3, 5);
function add(number1, number2) {
   return number1 + number2;
}
result;
```

Take note that we've declared our `add()` function with a function declaration, and we're calling the `add()` function before we've defined it in our code. Should this code execute properly?

When we hit 'enter' the code executes without error and we should get `8`. In this case we can call `add()` before it has been defined because of **hoisting**. This only happens with function declarations.

Now refresh your page and your DevTools console and copy/paste this code and hit enter:

```
const result = add(3, 5);
const add = function(number1, number2) {
   return number1 + number2;
}
result;
```

And what do we get? We get an error similar to this one:

```
Uncaught ReferenceError: add is not defined
```

The difference with the second code snippet is that we've defined our `add()` function with a function expression, which does not get hoisted. Code gets red from top to bottom, and in this case when we call `add(3,5)`, our JS interpreter can't find this function because it hasn't been hoisted and literally doesn't exist yet.

Because of hoisting, we generally recommend that you stick to using function declarations. However, you are welcome to use function expressions instead. Just remember the implications of hoisting!

Reusable Code that Scales

To write code that we can reuse, it's generally considered a good choice to set up event handlers to use function declarations or function expressions that are stored in variables. This code is considered scalable, because we can reuse those functions in multiple locations. The "scalability" of the code we write comes down to how quickly we can adapt our code to new requirements. In the previous example, we went from the requirement of alerting `"This is a heading element."` for one H1 tag to all 6 heading tags. Even though this example is a bit simplistic, we can see how using functions declarations (or function expressions stored in a variable) impact how easily we can reuse code.

Note that when you are just starting out with JavaScript and writing interactive programs, any code that works is good! When you get to your independent project for this course section, we really want to see a working project that follows JavaScript best practices (indentation, spacing, syntax, semicolons, descriptive variable names), and continues to implement the best practices for HTML and CSS. The next step up is considering code reusability and organizing your code based on the goal of making it easy to read and understand. This is something that we'll work on over multiple course sections.

Summary

In this lesson, we learned how to use function declarations in event handling, both with event handler properties and event listeners. We also learned that it's encouraged to set up event handlers to use function declarations or function expressions that are stored in variables, because it makes our code reusable. You can choose to use either function declarations or function expressions in your event handlers, just remember the implication of hoisting.

Previous (/introduction-to-programming/javascript-and-web-browsers/event-handling-with-event-listeners)
Next (/introduction-to-programming/javascript-and-web-browsers/removing-event-listeners)

Lesson 64 of 75
Last updated more than 3 months ago.

disable dark mode

Epicodus (http://www.epicodus.com)