Lesson Tuesday

Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / Event Handling with Event Handler Properties

Text Cheat sheet

So far, we've made our webpages interactive with `window` methods:

- `window.alert()`
- `window.prompt()`
- `window.confirm()`

However, these methods are not commonly used for user interaction, and as we learned they are associated with hacky websites and malware. Now that we are equipped with a working understanding of the `document` object and how to access and manipulate HTML elements, we're going to start exploring event handling.

There are two methods of event handling that we will learn in this course section: event handler properties and event listeners. In this lesson, we'll learn how to handle events with **event handler properties**. As we'll soon learn, these are also commonly called **onevent properties**.

Let's start with a review.

Review: Events and Event Handling

We were first introduced to events in the lesson called "How Web Browsers Work." In computer programming an **event** is any action or occurrence that software can recognize. An event can be triggered by a human, like a mouse click, or by something that our program creates, like an error. When we write our programs to be interactive, we write our programs to react to events!

In web development, events arise from user interaction with specific browser structures — like scrolling in a browser window or clicking on an HTML element. Each event has a name, like 'click' or 'scroll', and belongs to a specific object, like `document` (our HTML) or `window` (our browser window or tab). This enables developers to easily write code that targets specific events.

When developers write JavaScript to react to an event, we call this process **event handling**. As an example, consider an online website that hosts a game like Tic Tac Toe (https://www.mathsisfun.com/games/tic-tac-toe.html). We play the game with mouse clicks: every time a player is ready to place their X or O on the game board, they must click on a square on the webpage. To make this Tic Tac Toe game functional, we need to have code that is able to listen for the click event and do something in response, like call on a JavaScript function to update the game board with an X or an O.

Next up, we'll learn how to handle a click event on HTML elements.

Event Handling with Event Handler Properties

To handle a click event on an HTML element, we'll use an event handler property. An **event handler property** is a built-in property of an object (a Web API) that represents an event. Our `window`, `document`, and HTML elements all have built-in event handler properties.

Event handler properties "listen" for an event on the object that it is attached to. When the event happens, the event handler property runs some code as a reaction. As developers, we get to decide what the reaction is.

Let's look at an example to demystify this new concept. We'll continue to use the cookie recipe example in this lesson. If you want to code along with this lesson (which is optional) then open up your cookie recipe project in your browser and open the DevTools console as well.

Here's our HTML for the H1 element:

```
<h1 id="specialHeader">Best Chocolate Chip Cookies</h1>
```

Here's the JS for adding an event handler property for a click event. Since we're using the angle bracket `>` in the following code snippet, it means we're entering the following code into our DevTools console.

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

With the above code, we've made our program respond with an alert with the message "I am a heading element." every time we click on the H1 element "Best Chocolate Chip Cookies". Let's break down this new code.

`onclick` is our event handler property. It belongs to our `h1` variable which is an `HTMLHeadingElement` object. All event handler properties follow the following syntax (in pseudocode):

```
target.eventHandler = function
```

Let's explain each part of the syntax:

- `target` is the HTML element, `document`, or `window` object that we're attaching an event handler to. In our examples the `h1` variable representing an `HTMLHeadingElement` object is what we're targeting.
- `eventHandler` is the name of the event handler property that we're accessing on the target. In our example, the event handler property is `onclick`.
- `function` is the value that we set for the event handler property. It is always set to a function. Every time the event happens, the function will be called and the code inside of it will be run. In our example, we used an anonymous function expression as the value of our `onclick` event handler:

```
h1.onclick = function() {
  window.alert("I am a heading element.");
};
```

Now, every time someone clicks on the H1 element in the DOM, the anonymous function expression will be called and the code inside of it will be run, causing an alert to pop up.

We always set the value of an event handler property to a function. Why? Well, event handler properties are expected to be assigned to a function. They have an internal mechanism that knows to call the function when the event happens. We can also manually call on our event handler properties like this and it will trigger the `onclick` function:

```
> h1.onclick();
```

Pretend that an alert pops up after we input this into our DevTools console (or, try it out now in your own Cookie Recipe project). Even though we can do this, it's not a common practice.

Rewriting or Removing an Event Handler

If messed up our original `h1.onclick` event handler in the DevTools console, something like this:

```
> h1.onclick = function() {
  window.alert("I m a hding element."); // with typos in the message
};
```

We can easily fix it by reassigning `h1.onclick` to a new function expression.

```
> h1.onclick = function() {
  window.alert("I am a heading element."); // the typos are fixed!
};
```

Doing this replaces the previous function, just like with reassigning the value of a JavaScript variable.

We can remove an event handler by reassigning the value of the property to `null`, which intentionally represents an absence of a value (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/null). Using `null` thereby gives the event handler no value, and stops the event handler from reacting to the event.

```
> h1.onclick = null;
```

One reason to remove an event handler is when you only want to react to an event once, or only a select number of times. When we learn about event handling with event listeners later in this course section, we'll review other reasons to remove event handlers.

Property Naming Conventions

The name of the `onclick` property suggests that we're saying to our program, "on a click event, do X", where "X" is how we want the click event to be handled. For this reason, these event handler properties are also called **onevent properties**. The name "onevent property" conveniently reminds us of the naming convention for these event handler properties: on + name of event. For example, consider these two additional onevent properties that we can set for our H1 element:

```
// the oncopy property lets us respond to when
// all or part of the H1 element has been copied.
> h1.oncopy = function() {
  window.alert("The heading element has been copied.");
};
```

```
// the onmouseover property lets us respond to when
// a mouse has hovered over the H1 element.
> h1.onmouseover = function() {
  window.alert("The mouse has hovered over the heading element.");
};
```

The `oncopy` property lets us respond to when all or part of the H1 element has been copied, and the `onmouseover` property lets us respond to when a mouse has hovered over the H1 element. As we can see, the names of these properties follow the intuitive naming convention of on + name of event.

Events Are Categorized by Type and by the Object it Belongs To

Events are categorized in two ways. First, events are grouped into categories that we call "event types". For example, the `onclick` and `onmouseover` events are a part of Mouse Events. The `oncopy` event is a part of Clipboard Events. Typically for every event type, there's multiple events. For example, Clipboard Events also include cut and paste events. For a full list of event types, visit this reference page on MDN (https://developer.mozilla.org/en-US/docs/Web/Events#event_index). In the next lesson, we'll explore this reference page more in depth.

Second, events are also categorized as belonging to an object. This is really important! Events don't exist by themselves — they always happen on or to something. For example:

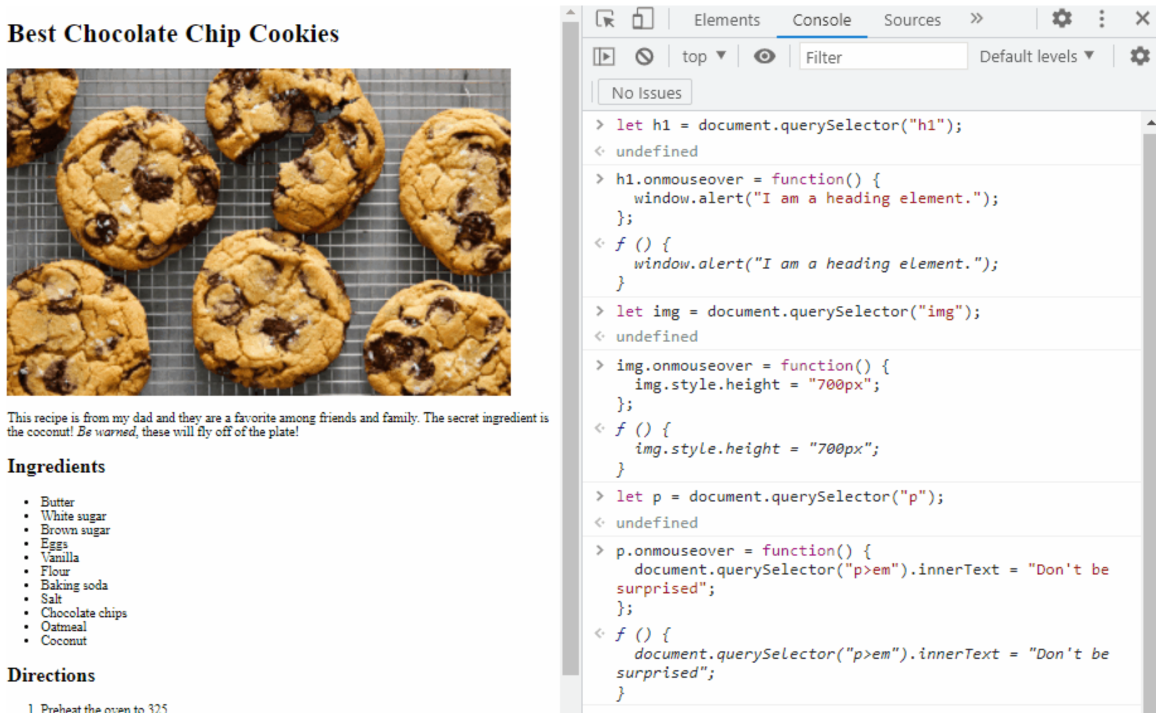
- The screen resize event happens to the window.
- The copy event happens on the HTML element.

Connecting events to objects enables different objects to respond to the same event. For example, we could have an `onmouseover` event set up for every HTML element in the DOM, and each `onmouseoever` event could trigger an entirely different function to execute.

```
> let h1 = document.querySelector("h1");
> h1.onmouseover = function() {
  window.alert("I am a heading element.");
};
> let p = document.querySelector("p");
> p.onmouseover = function() {
  document.querySelector("p>em").innerText = "Don't be surprised";
};
> let img = document.querySelector("img");
> img.onmouseover = function() {
  img.style.height = "700px";
};
```

In this example, we can see that we've set 3 different elements in the DOM to react to the same `onmouseover` event — however, each reaction is distinct. The gif below shows the results of adding the above event handlers to the cookie recipe.

Optionally, open your cookie recipe project and type in the above code into your DevTools console to try it out. Make sure to remove the `>` angle brackets, as those will throw an error. The angle brackets are just to denote that the code is meant for the DevTools console. We encourage you to type out your code instead of copy/pasting so that you can develop your typing ability and muscle memory.



Best Chocolate Chip Cookies

This recipe is from my dad and they are a favorite among friends and family. The secret ingredient is the coconut! *Be warned*, these will fly off of the plate!

Ingredients

- Butter
- White sugar
- Brown sugar
- Eggs
- Vanilla
- Flour
- Baking soda
- Salt
- Chocolate chips
- Oatmeal
- Coconut

Directions

1. Preheat the oven to 325

```
> let h1 = document.querySelector("h1");
< undefined
> h1.onmouseover = function() {
    window.alert("I am a heading element.");
  };
< ƒ () {
    window.alert("I am a heading element.");
  }
> let img = document.querySelector("img");
< undefined
> img.onmouseover = function() {
    img.style.height = "700px";
  };
< ƒ () {
    img.style.height = "700px";
  }
> let p = document.querySelector("p");
< undefined
> p.onmouseover = function() {
    document.querySelector("p>em").innerText = "Don't be surprised";
  };
< ƒ () {
    document.querySelector("p>em").innerText = "Don't be surprised";
  }
```

Keyboard Events: `onkeydown` and `onkeyup`

Let's look at two keyboard events: `onkeydown` and `onkeyup`. We'll update our cookie recipe to react to these events by changing the background and text color of the document body. So what do these keyboard events target?

- `onkeydown` enables our program to react when a keyboard key is pressed down.
- `onkeyup` enables our program to react when a keyboard key is let go after it has been pressed.

We'll continue to input code into our DevTools console for our Cookie Recipe project:

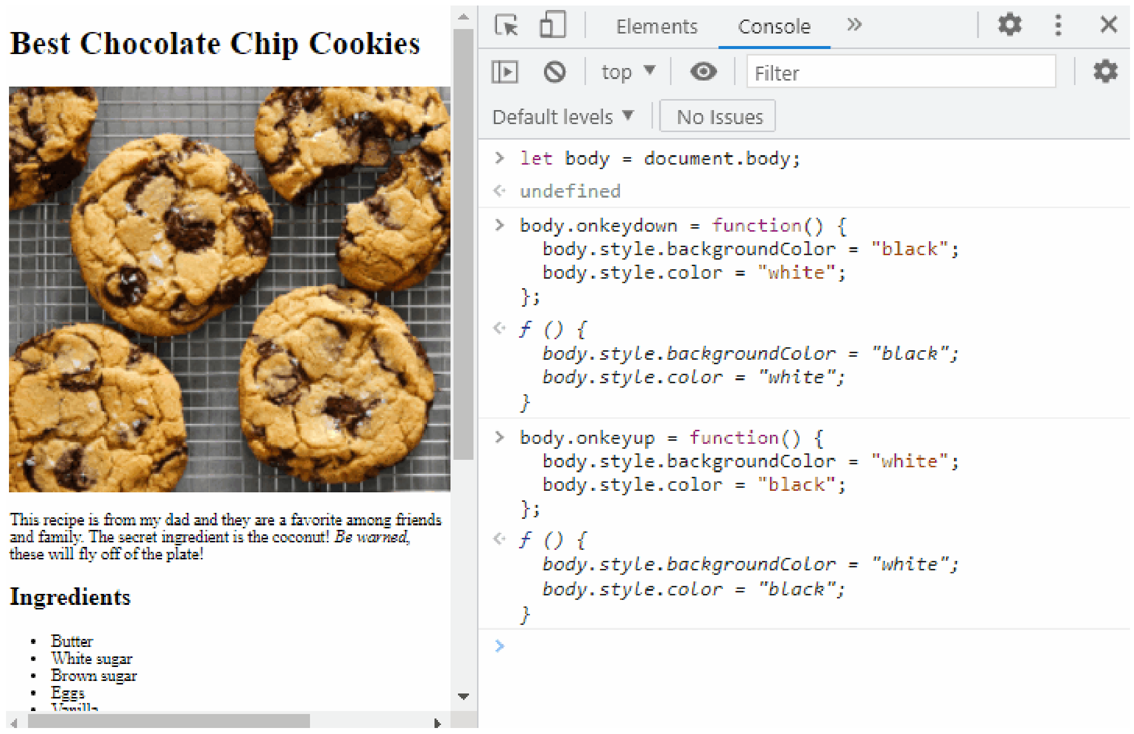
```
> let body = document.body;
> body.onkeydown = function() {
  body.style.backgroundColor = "black";
  body.style.color = "white";
};
> body.onkeyup = function() {
  body.style.backgroundColor = "white";
  body.style.color = "black";
};
```

Notice that we're using the `document` object property called `body` to target the HTML body element: `let body = document.body;` . The `document` object has these special properties for both the `<head>` and `<body>` tags. We could also do this instead:

```
> let body = document.querySelector("body");
```

Next, we create two different reactions to the body element's `onkeydown` event and `onkeyup` event: when we press a key, we set the background color to black and the text color to white, and when we let a key go we set the reverse values for each style property. When we type on our body (even though there's nowhere to type), we should see the colors of our HTML text and background change! If you are trying this out, make sure to click back into the webpage before you start pressing on your keys.

The gif below demonstrates the results of this new code. The code is input into the DevTool console on the right, then the cursor clicks back into the webpage, then some keys are pressed down and let go.


Best Chocolate Chip Cookies
This recipe is from my dad and they are a favorite among friends and family. The secret ingredient is the coconut! Be warned, these will fly off of the plate!
Ingredients
Butter
White sugar
Brown sugar
Eggs
Elements
Console
top
Filter
Default levels
No Issues

```
> let body = document.body;
< undefined
> body.onkeydown = function() {
    body.style.backgroundColor = "black";
    body.style.color = "white";
  };
< ƒ () {
    body.style.backgroundColor = "black";
    body.style.color = "white";
  }
> body.onkeyup = function() {
    body.style.backgroundColor = "white";
    body.style.color = "black";
  };
< ƒ () {
    body.style.backgroundColor = "white";
    body.style.color = "black";
  }
>
```

Outmoded: Inline HTML Event Handler Attributes

As you explore and research event handler properties, you'll come across using inline HTML attributes to handle events. These look remarkably similar to event handler properties, only they are HTML attributes that are added directly inline to our HTML. Here's an example of two event handlers added to the `<body>` tag in our cookie recipe:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title> Best Chocolate Chip Cookies </title>
    <link rel="stylesheet" href="css/styles.css" type="text/css">
  </head>
  <body onkeydown="keyDown()" onkeyup="document.body.style.backgroundColor='wh
ite';document.body.style.color='black'">
    <h1 id="specialHeader">Best Chocolate Chip Cookies</h1>
    <img src="https://static01.nyt.com/images/2022/02/12/dining/JT-Chocolate-C
hip-Cookies/JT-Chocolate-Chip-Cookies-articleLarge.jpg" alt="An image of a coo
kie"/>

    ...

    <script>
      function keyDown() {
        document.body.style.backgroundColor = 'black';
        document.body.style.color = 'white';
      }
    </script>
  </body>
</html>
```

Note — the `...` represents additional HTML that we are abbreviating in this code snippet. Here's the event handler attributes separated out:

```
<body onkeydown="keyDown()" onkeyup="document.body.style.backgroundColor='whit
e';document.body.style.color='black'">
```

```
<script>
  function keyDown() {
    document.body.style.backgroundColor = 'black';
    document.body.style.color = 'white';
  }
</script>
```

The above example shows us two ways that we can set up an inline HTML event handler attribute:

- With `onkeydown`, we're setting the value of that attribute to a function call `keyDown()` which is defined within the `<script>` tags at the bottom of our HTML body. Anything between the `<script>` tag does not get printed to the DOM. Instead, the code gets read and processed as JavaScript. Indeed, we can write any JavaScript within the `<script>` tags.
    - **Take note:** it's less common to use the `<script>` tags to write JavaScript in our HTML file. It's considered best practice to write our JavaScript in separate files. Doing so makes our code easier to read and maintain because we're separating our concerns: HTML is in our `.html` files, and JS is in our `.js` files.
- With `onkeyup`, we're writing JS commands directly within the quotes as the value of the `onkeyup` attribute. Notice that there are two commands that are separated by a semi colon `;`. This method of assigning a value for an HTML event handler attribute is less favorable because it's considered hard to read.

Don't Use HTML Event Handler Attributes

This method of handling events with inline HTML event attributes is outmoded and you shouldn't use them. It's important to be aware that they exist so you can recognize them in examples from online resources, and know to avoid them.

Why are these considered outmoded and bad practice?

1. Developers should always try to separate their JavaScript from their HTML. When we combine them it makes both harder to read and understand. Code that's harder to understand is code that is harder to maintain and debug. It's a best practice to keep JavaScript in a separate file that's connected to the HTML via the `<script>` tag.
2. What if we had a long webpage with a long form on it with a series of 10 "save" buttons. Each save button does the same thing — it saves all form fields. If we wanted to target a click event on all of those buttons to save the forms, we'd need to add an `onclick` event handler attribute to each button in our HTML. Conversely, with an event handler property we can target all button elements at once with a single `onclick` event handler property that saves the forms. In this example, using inline event handler attributes would bloat our HTML, while using event handler properties would separate our code (HTML from JS) and thereby make it easier to understand and maintain.

There's an equivalent event handler property for every inline HTML event handler attribute, so there should be no reason to use HTML event handler attributes.

To read more about HTML event handler attributes see this resource on MDN (https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events#inline_event_handlers_%E2%80%94_dont_use_thes

Summary

In this lesson we learned about event handler properties:

- These properties are also called "onevent" properties because they follow the naming convention on + event name.
- We set the value of an event handler property to a function. Any time the event is triggered, the function will be called. We can put whatever code we want to have run into the function.
- Event handler properties belong to Web API objects (also called "interfaces"), like `Element`, `HTMLElement`, `document`, and `window`. This lesson covered events that targeted the HTML body, paragraph, H1, and image elements.
- Event handler properties are grouped by type, and often an event type is made up of many events. For example:
  - Keyboard events are an event type.
  - Keyboard events are made up of three events: `onkeypress`, `onkeyup`, `onkeydown`.
- **Never use inline HTML event handler attributes.**

In the next lesson, we'll learn how to use event handler properties in a project. Then we'll practice using event handler properties. In upcoming lessons, we'll use this new skill to make more complicated applications that take user input via forms.

Previous (/introduction-to-programming/javascript-and-web-browsers/function-expressions)
Next (/introduction-to-programming/javascript-and-web-browsers/event-handler-properties-in-a-project-using-window-onload)
Lesson 51 of 75
Last updated more than 3 months ago.

disable dark mode