# Event Handling: Common Page Events

Event handling is the core of front-end development. When a user interacts with HTML elements on a website, those interactions are known as ***events***. Developers use Javascript to respond to those events. In this reading, we'll go over three common events and do exercises to add functionality based on those events:

- A button click
- A checkbox being checked
- A user typing a value into an input

## Handling a button click event

Let's start with a common event that occurs on many websites: a button click. Usually some functionality occurs when a button is clicked -- such as displaying new page elements, changing current elements, or submitting a form.

We'll go through how to set up a click eventlistener and update the click count after each click. Let's say we have a button element in an HTML file, like below:

**HTML**

```
<!DOCTYPE html>
<html>
  <head>
    <script src="script.js">
  </head>
  <body>
    <button id="increment-count">I have been clicked <span id="clicked-count">0
  </body>
```

```
</html>
```

We'll write Javascript to increase the value of the content of `span#clicked-count` by one each time `button#increment-count` is clicked. Remember to use the `DOMContentLoaded` event listener in an external script to ensure the button has loaded on the page before the script runs.

### Javascript

If you open up the HTML file in a browser, you should see the button. If you click the button rapidly and repeatedly, the value of `span#clicked-count` should increment by one after each click.

```
// script.js

window.addEventListener("DOMContentLoaded", event => {
  const button = document.getElementById("increment-count");
  const count = document.getElementById("clicked-count");
  let clicks = 0;
  button.addEventListener("click", event => {
    clicks += 1;
    count.innerHTML = clicks;
  });
});
```

## Using addEventListener() vs. onclick

Adding an event listener to the button element, as we did above, is the preferred method of handling events in scripts. However, there is another method we could use here: GlobalEventHandlers.onclick. Check out www.simonewebdesign.it for a breakdown of the differences between using `addEventListener()` and `onclick`. One distinction is

that `onclick` overrides existing event listeners, while `addEventListener()` does not, making it easy to add new event listeners.

The syntax for `onclick` is: `target.onclick = functionRef;` If we wanted to rewrite the button click event example using `onclick`, we would use the following:

```
let clicks = 0;
button.onclick = event => {
  clicks += 1;
  count.innerHTML = clicks;
};
```

There's also yet another way to register the `onclick` event, by putting it directly into the HTML markup and using a named function like so:

```
function clickHandler(event) {
  clicks += 1;
  count.innerHTML = clicks;
}


<button onclick="clickHandler(event)"/>
```

This way suffers from all the problems that the `onclick` property suffers from with the additional problem of relying on the global `window.event` object. You'll noticed that `clickHandler(event)` is passing in an `event` variable. Where is that coming from? Well everytime an event happens, the global variable `window.event` changes to be the current event so that's what is being passed to the function. It's a very messy approach and generally should be avoided.

We'll stick to using `addEventListener()` in our code, but it's important for front-end developers to understand the differences between the methods above and use cases for each one.

## Handling a checkbox check event

Another common event that occurs on many websites is when a user checks a checkbox. Checkboxes are typically recorded values that get submitted when a user submits a form, but checking the box sometimes also triggers another function.

Let's practice displaying an element when the box is checked and hiding it when the box is unchecked. We'll pretend we're on a pizza delivery website, and we're filling out a form for pizza toppings. There is a checkbox on the page for extra cheese, and when a user checks that box we want to show a `div` with pricing info. Let's set up our HTML file with a `checkbox` and `div` to show/hide, as well as a link to our Javascript file:

**HTML**

```
<!DOCTYPE html>
<html>
  <head>
    <script src="script.js">
  </head>
  <body>
    <h1>Pizza Toppings</h1>
    <input type="checkbox" id="on-off">
    <label for="on-off">Extra Cheese</label>
    <div id="now-you-see-me" style="display:none">Add $1.00</div>
  </body>
</html>
```

Note that we've added `style="display:none"` to the `div` so that, when the page first loads and the box is unchecked, the `div` won't show.

In our `script.js` file, we'll set up an event listener for `DOMContentLoaded` again to make sure the `checkbox` and `div` have loaded. Then, we'll write Javascript to show `div#now-you-see-me` when the box is checked and hide it when the box is unchecked.

### Javascript

```
// script.js

window.addEventListener("DOMContentLoaded", event => {
  // store the elements we need in variables
  const checkbox = document.getElementById("on-off");
  const divShowHide = document.getElementById("now-you-see-me");
  // add an event listener for the checkbox click
  checkbox.addEventListener("click", event => {
    // use the 'checked' attribute of checkbox inputs
    // returns true if checked, false if unchecked
    if (checkbox.checked) {
      // if the box is checked, show the div
      divShowHide.style.display = "block";
      // else hide the div
    } else {
      divShowHide.style.display = "none";
    }
  });
});
```

Open up the HTML document in a browser and make sure that you see the `checkbox` when the page first loads and not the `div`. The `div` should show when you check the box, and appear hidden when you uncheck the box.

The code above works. However, what would happen if we had a whole page of checkboxes with extra options inside each one that would show or hide based

on whether the boxes are checked? We would have to call `Element.style.display = "block"` and `Element.style.display = "none"` on each associated `div`.

Instead, we could add a `show` or `hide` class to the `div` based on the checkbox and keep our `display:block` and `display:none` in CSS. That way, we could reuse the classes on different elements, as well as see class names change in the HTML. Here's how the code we wrote above would look if we used CSS classes:

### Javascript

```
// script.js
// we need to wait for the stylesheet to load
window.onload = () => {
  // store the elements we need in variables
  const checkbox = document.getElementById("on-off");
  const divShowHide = document.getElementById("now-you-see-me");
  // add an event listener for the checkbox click
  checkbox.addEventListener("click", event => {
    // use the 'checked' attribute of checkbox inputs
    // returns true if checked, false if unchecked
    if (checkbox.checked) {
      // if the box is checked, show the div
      divShowHide.classList.remove("hide");
      divShowHide.classList.add("show");
      // else hide the div
    } else {
      divShowHide.classList.remove("show");
      divShowHide.classList.add("hide");
    }
  });
};
```

### CSS

```
.show {
  display: block;
}
.hide {
  display: none;
}
```

**HTML (Remove the style attribute, and add the "hide" class)**

```
<div id="now-you-see-me" class="hide">Add $1.00</div>
```

## Handling a user input value

You've learned a lot about event handling so far! Let's do one more exercise to practice event handling using an input. In this exercise, we'll write JavaScript that will change the background color of the page to cyan five seconds after a page loads unless the field `input#stopper` contains only the text "STOP".

Let's set up an HTML file with the input and a placeholder directing the user to type "STOP": **HTML**

```
<!DOCTYPE html>
<html>
  <head>
    <script src="script.js">
  </head>
  <body>
    <input id="stopper" type="text" placeholder="Quick! Type STOP">
  </body>
</html>
```

Now let's set up our Javascript:

**Javascript**

```
// script.js
// run when the DOM is ready
window.addEventListener("DOMContentLoaded", event => {
  const stopCyanMadness = () => {
    // get the value of the input field
    const inputValue = document.getElementById("stopper").value;
    // if value is anything other than 'STOP', change background color
    if (inputValue !== "STOP") {
      document.body.style.backgroundColor = "cyan";
    }
  };
  setTimeout(stopCyanMadness, 5000);
});
```

The code at the bottom of our function might look familiar. We used`setInterval`along with the Javascript Date object when we set up our current time clock. In this case we're using `setTimeout`, which runs `stopCyanMadness`after 5000 milliseconds, or 5 seconds after the page loads.

## What we learned:

- How to add an event listener on a button click
- How to add an event listener to a checkbox
- Styling elements with Javascript vs. with CSS classes
- How to check the value of an input