Event Handling (W4D3) - Learning Objectives

Event Handling

1. Given an HTML page that includes
   `<button id="increment-count">I have been clicked <span id="clicked-count">0</span> times</button>`
   , write JavaScript that increases the value of the content of span#clicked-count by 1 every time button#increment-count is clicked.

```
window.addEventListener("DOMContentLoaded", event => {
  // getElementById returns a single element (the first that it finds that has the specified id)
  const button = document.getElementById("increment-count");
  const count = document.getElementById("clicked-count");
  button.addEventListener("click", event => {
    count.innerHTML = `${event.detail}`;//track the number of clicks in a short period of time....
  });
});
```

- event.detail will reset if we pause clicking on the button. If we want a persistent count, we can keep track of a variable:

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

2. Given an HTML page that includes
   `<input type="checkbox" id="on-off"><div id="now-you-see-me">Now you see me</div>` , write JavaScript that sets the display of div#now-you-see-me to "none" when input#on-off is checked and to "block" when input#on-off is not checked.

```
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";
    }
  });
});
```

- We also could have used a CSS file that had styling for `show` and `hide` classes, then added and removed the classes instead of changing the style directly.

```
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.className = "show";
      // else hide the div
    } else {
      divShowHide.className = "hide";
    }
  });
});
```

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

3. Given an HTML file that includes `<input id="stopper" type="text" placeholder="Quick! Type STOP">`, 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".

```
  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";
    } else {
      // If we successfully typed in "STOP" I added an indicator to show that it worked
      console.log("Thank you for stopping the madness");
    }
  };
  setTimeout(stopCyanMadness, 5000);
});
```

4. Given an HTML page with that includes `<input type="text" id="fancypants">` , write JavaScript that changes the background color of the textbox to #E8F5E9 when the caret is in the textbox and turns it back to its normal color when focus is elsewhere.

```
  window.addEventListener("DOMContentLoaded", event => {
  const input = document.getElementById("fancypants");

  input.addEventListener("focus", event => {
    event.target.style.backgroundColor = "#E8F5E9";
  });
  input.addEventListener("blur", event => {
    event.target.style.backgroundColor = "initial";
  });
});
```

- One important thing to note here is that the event listeners for the `focus` and `blur` events are on the input tag directly. In our project this week we had these event listeners on a parent element because we wanted to apply the same action to many child input tags. In order for us to do this we had to pass in the third argument `true` to `addEventListener` because the `focus` and `blur` events do not bubble. The `true` argument tells `addEventListener` to trigger the callback on the capture phase instead of on the standard bubble phase. (We could also use the `focusin` and `focusout` events, which do bubble.)

5. Given an HTML page that includes a form with two password fields, write JavaScript that subscribes to the forms submission event and cancels it if the values in the two password fields differ.

```
  window.addEventListener("DOMContentLoaded", event => {
  // get the form element
  const form = document.getElementById("signup-form");

  const checkPasswordMatch = event => {
    // get the values of the pw field and pw confirm field
    const passwordValue = document.getElementById("password").value;
    const passwordConfirmValue = document.getElementById("confirm-password")
      .value;
    // if the values are not equal, alert the user
    // otherwise, submit the form
    if (passwordValue !== passwordConfirmValue) {
      // prevent the default submission behavior
      event.preventDefault();
      alert("Passwords must match!");
    } else {
      alert("The form was submitted!");
    }
  };
  // listen for submit event and run password check
  form.addEventListener("submit", checkPasswordMatch);
});
```

6. Given an HTML page that includes a div styled as a square with a red background, write JavaScript that allows a user to drag the square around the screen.

- Important aspects of this LO are knowing that we have to add the 'draggable' property to the element, as well as knowing the basics of what events are associated with a drag-n-drop.
-
- `dragstart` : listened for on the element that is being dragged, fired when the user clicks and starts to move
- `dragenter` : fired when a draggable element crosses the boundary into this element
- `dragleave` : fired when a draggable element crosses the boundary out of this element
- `dragover` : fired every few hundred milliseconds while a draggable object is over this element
- `drop` : fired when a draggable object is released over this element
- Reference the example on aaonline for how these are applied: https://open.appacademy.io/learn/js-py--aug-2020-online/week-4-aug-2020-online/drag-n-drop-api
- Reference the MDN docs for more info on all of the different events: https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API
- Don't worry about implementation of a drag event in an assessment scenario, it is a good reference to see that there are many different types of events, not just clicks and input.

7. Given an HTML page that has 300 DIVs, create one click event subscription that will print the id of the element clicked on to the console.

```
  window.addEventListener("DOMContentLoaded", event => {
  // Add a click event listener on the document's body
  document.body.addEventListener("click", event => {
    // console.log the event target's ID
    console.log(event.target.id);
  });
});
```

8. Identify the definition of the bubbling principle.

- When an event occurs on an element, any handlers that respond to that event on the target are invoked, then on the parent element, then on its parent, all the way up the DOM hierarchy. We can prevent this bubbling by invoking `event.stopPropogation()` .