Module 3-9

JavaScript Event Handling

Can you?

- Select DOM objects and attach anonymous functions with addEventListener()
- Describe event bubbling and how it works
- Describe default browser behavior and the various events that are triggered (w3schools HTML Event Reference)
- List possible events and the elements that may be associated with those events
- Describe how to add listeners to newly created DOM elements
- Remove an event listener with removeEventListener()

Events

- Events are changes that can occur within HTML DOM elements as a result of the browser’s page life cycle or user interaction.
- Examples of events:
  - A user hovering over a piece of text with the mouse cursor.
  - A user clicking on a link or button.
  - A HTML page loading for the first time.
  - A user double clicking somewhere on the page.

- JavaScript can be used to define actions that should take place when these events occur.

Common Events

Event	Description
click	user clicks once
mouseover	when the mouse cursor is over an element
dblclick	user clicks twice in rapid succession
change	user changes the value on a form (if it’s an input box, user needs to click somewhere else to complete the event)
focus	When an element is the currently active one, think again about a form, the field you are currently on is the one that’s focused.
blur	Opposite of focus, element has lost focus, something else is focused.

Adding Events Using JavaScript: *.addEventListener()*

Events are added using the addEventListener method, it is a method that can be called on any DOM element. We can quickly explore the method using the code below

HTML

```
<button type="button"
id='superBtn'>You are
awesome.</button>
```

JS

```
let button = document.getElementById('superBtn');
button.addEventListener('click', reply);

function reply() {
   window.alert('No.... you are awesome!');
}
```

The first parameter is the event we are trying to catch. The second parameter is the action it will take, most likely codified in a function.

In-line HTML Event Handlers vs. Event Listeners

<button onclick="btnClick()">Click Me!</button>

In-line Events, while convenient, have drawbacks:

1. They add complexity to the HTML document
2. They are attached to a fixed attribute value and can be overwritten
3. They violate the separation of concerns provided by the HTML - CSS - JS development model.

Event Listeners solve these in-line shortcomings and add one ***BIG*** benefit:

1. ***MULTIPLE EVENTS LISTENERS CAN BE ADDED TO A SINGLE EVENT!***

Once you become familiar with addEventListener method, dive deeper here: https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener

Let’s add some events

The “DOMContentLoaded” Event

DOMContentLoaded is a special event that is particularly helpful as one of the browser’s page lifecycle events that runs every time an HTML document is loaded. Specifically, this event fires once the browser has fully loaded the HTML and the DOM tree is built. *However*, external resources like pictures <img> and stylesheets may not yet be loaded.

We can use this event to write JS code that “preps” page content before a user starts interacting with the page:

- Set the values of some DOM elements (i.e. page titles, lists)
- Add event handlers for elements on the page

DOMContentLoaded Example

Walk through the following Code:

HTML

```
<html><body>
   <div id='content'>
       <input id='theButton' type='button' value='Surprise!'/>
   </div>

<script src="thisScript.js"></script>
</body></html>
```

JS

```
document.addEventListener('DOMContentLoaded', doAfterDOMLoads);

function doAfterDOMLoads() {

   let btn = document.getElementById('theButton');
   btn.addEventListener('click', buttonAction);
}

function buttonAction() {
   window.alert('surprise!');

}
```

1
2
3
4

1. First thing that happens, an event listener is defined that will be triggered by DOMContentLoaded to run the method **doAfterDOMLoads**.

1. Once the DOMContentLoaded event fires, the method **doAfterDOMLoads** is executed.

1. Note that in turn, this method defines another event handler for the button. If the button is clicked, the method **buttonAction** will execute.

1. If someone clicks the button, this method executes.

DOMContentLoaded Example

Alternatively, the previous block of JS code can be implemented using anonymous functions. Note that these two blocks of code are functionally identical.

```
document.addEventListener('DOMContentLoaded',
doAfterDOMLoads);

function doAfterDOMLoads() {

   let btn =
document.getElementById('theButton');
   btn.addEventListener('click', buttonAction);
}
function buttonAction() {
   window.alert('surprise!');

}
```

JS

```
document.addEventListener('DOMContentLoaded',
() => {
   let btn =
document.getElementById('theButton');
   btn.addEventListener('click',
        () => {
            window.alert('surprise!');
        }
   );
}
);
```

JS

The DOMContentLoaded Pattern

Having an event handler for DOMContentLoaded attach the content's listeners is a standard way to create event handlers:

1. Add an event listener that responds to the DOMContentLoaded event.

2. The DOMContentLoaded event then adds all other event listeners for the page elements (i.e. buttons, form elements, etc.)

Adding Events using Anonymous Functions

Instead of calling a named function, we can write it anonymously (no name):

HTML

```
<button type="button"
id='superBtn'>You are
awesome.</button>
```

JS

```
let button = document.getElementById('superBtn');
button.addEventListener('click', () => {
     window.alert('No.... you are awesome!');
});
```

Best practice is to first write named function, then call named function in the event listener.

JS

```
function awesomeFunction() {
   window.alert('No.... you are awesome!');
};

document.getElementById("superBtn").addEventListener("click", () => {
   awesomeFunction();
});
```

Event Object

```
function awesomeFunction() {
  window.alert('No... you are awesome!');
};

document.getElementById("superBtn").addEventListener("click", (event) => {
    awesomeFunction();
});
```

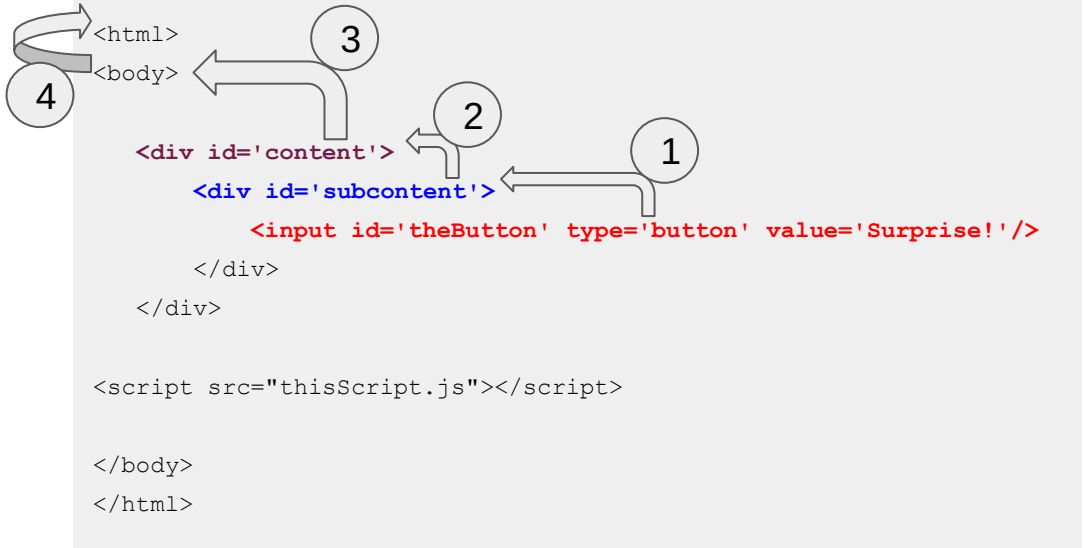
The event object holds some important properties that we can use. Example, if we want to know which mouse button was pressed in a mousedown event, where a mouse event happened (x, y, or page), if the mouse was moved, etc.

Before we continue working on our lecture code ...

… Let’s talk about bubbling

Event Bubbling

HTML elements have a hierarchy of parent child relationships An event tied to a child element will propagate (bubble up) and can fire corresponding event handlers associated with each parent.



Under the hood, an event triggered by the button travels upward in the following direction:

1. From the button to its immediate parent (#subcontent)
2. From *#subcontent* to *#content*
3. From *#content* to *body*
4. From *body* to *html*

Event Bubbling

If the following code demonstrates bubbling in the DOM

```
document.addEventListener('DOMContentLoaded', doAfterDOMLoads);

function doAfterDOMLoads() {

   let btn = document.getElementById('theButton');
   let sub = document.getElementById('subcontent');
   let main = document.getElementById('content');

   btn.addEventListener('click', buttonAction);
   sub.addEventListener('click', buttonAction);
   main.addEventListener('click', buttonAction);
}

function buttonAction() {
   window.alert('surprise!');
}
```

How to stop event bubbling (propagation)

If the following JS code were in place, we'd see the popup appear three times:

```
document.addEventListener('DOMContentLoaded', doAfterDOMLoads);

function doAfterDOMLoads() {
   let btn = document.getElementById('theButton');
   let sub = document.getElementById('subcontent');
   let main = document.getElementById('content');
   btn.addEventListener('click', buttonAction1);
   sub.addEventListener('click', buttonAction2);
   main.addEventListener('click', buttonAction2);
}
function buttonAction1(event) {
   window.alert('surprise!');
   event.stopPropagation();
}
function buttonAction2() {
   window.alert('surprise again!');
}
```