# Module 4 Day 9

Event Handling

# Events

- Events are changes that can occur within HTML DOM elements.
- 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.

- We use JS to help define the actions that should take place if 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

Events are added using the addEventListener method, it is a method of a DOM element, consider the following example:

HTML

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

JS

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

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

This method is straightforward, 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.

# Let’s add some events

# The “DOMContentLoaded” Event

There is one event that is particularly helpful as it fires every time a HTML page is loaded. More specifically, the event fires when all DOM ***Elements*** have been loaded.

We can use this event to write JS code that “preps” anything on the page before a user starts interacting with the page...even before styles are applied and images are loaded. Some examples:

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

# DOMContentLoaded Example

Consider 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**.
2. Now, once the DOMContentLoaded event fires, the method **doAfterDOMLoads** is executed.
3. This method defines another event handler that is attached to the button. A button click will now fire the **buttonAction** method.
4. When the user clicks the button, they are alerted with a surprise.

# DOMContentLoaded Example

You can structure the previous block of JS code with anonymous functions if you want. Note that these two blocks of code are 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” Event

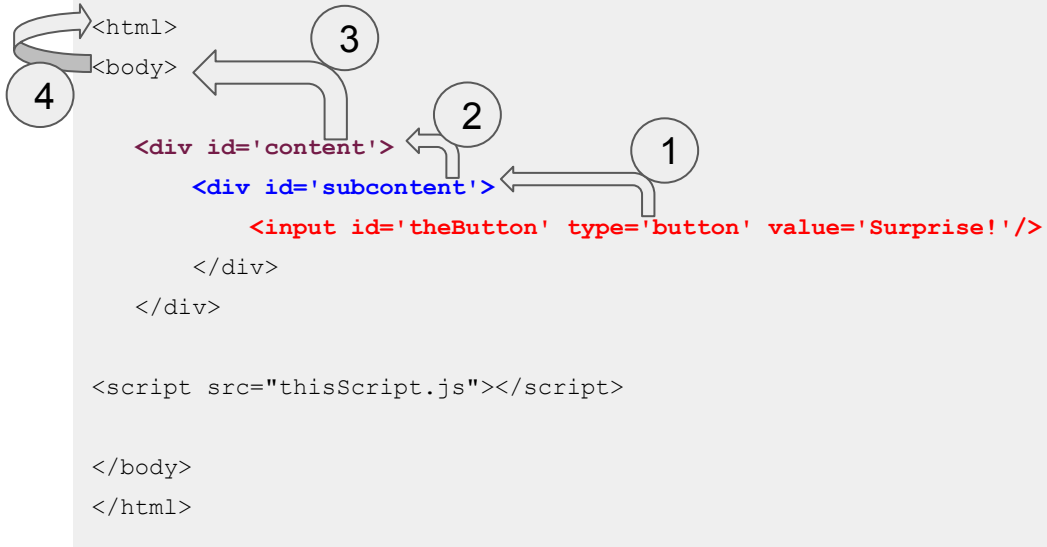
This is the standard pattern we want to use when creating event handlers:

1. Add an event listener that responds to the DOMContentLoaded event.
2. To the method attached to step 1, add all other event listeners for other page elements (i.e. buttons, form elements, etc.)

# Let’s work on a more comprehensive example

# Event Bubbling

If your HTML elements have a hierarchy of parent child relationships, an event tied to a child element will propagate up (bubble), possibly activating any events tied to the parent. Consider the following:




Under the hood, an event attached to 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 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', buttonAction);
   sub.addEventListener('click', buttonAction);
   main.addEventListener('click', buttonAction);
}

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

We can control this behavior using the event.stopPropagation() method !

Field Trip!
https://www.w3schools.com/JSREF/tryit.asp?filename=tryjsref_event_stoppropagation