Module 3

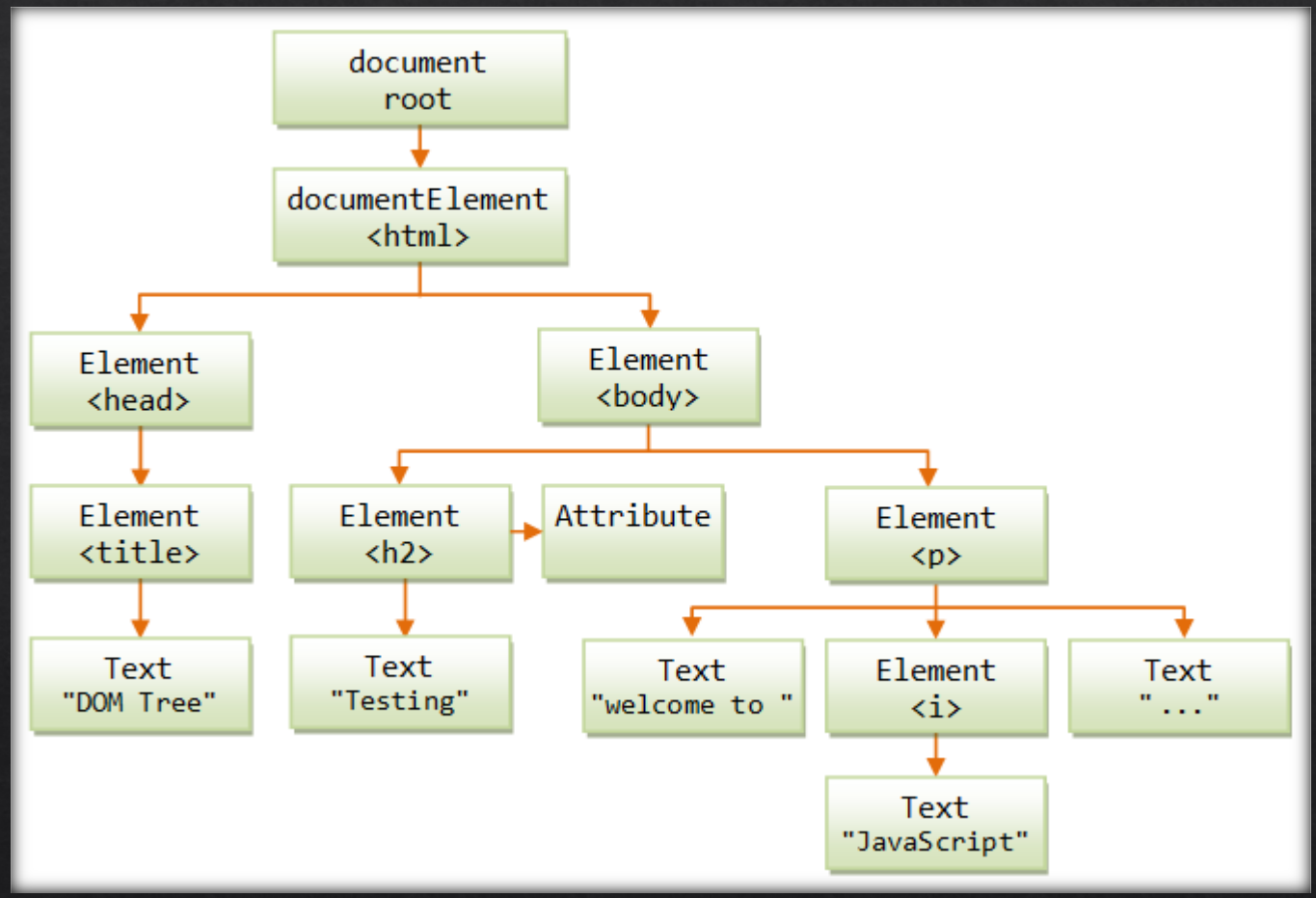
JS EVENT HANDLING

JAVA – MODULE 3, DAY 9

Review

RECAP

DOM



★ The DOM is modeled as a tree structure.

TECH ELEVATOR

DOM QUERYING

Review

```
1 const element = document.getElementById('someElementId');
2
3 element.innerText = 'This is the new content of the element';
```

```
1 const element = document.querySelector('a.active');
2
3 element.innerText = 'This is the new content of the element';
```

```
1 const paragraphs = document.querySelectorAll('p');
2
3 element.innerText = 'This is the new content of the element';
```

DOM MANIPULATION

```
1 const element = document.querySelector('#someElementId');
2
3 element.innerText = 'This is the new content of the element';
```

```
1 const element = document.getElementById('someElementId');
2
3 element.innerHTML = '<p>This could be a security issue!</p>';
```

★ The DOM can be changed by JavaScript at anytime after it has been created.

Review

appendChild

```
1 const container = document.getElementById('someElementId');
2
3 // Create and populate the element. It does not yet exist in the DOM
4 const el = document.createElement('h1');
5 el.innerText = 'Hello JavaScript';
6
7 // Actually add the element to the DOM as the last child of container
8 container.appendChild(el);
```

insertAdjacentElement

Review

```
targetElement.insertAdjacentElement(position, element);
```

Parameters

`position`

A `DOMString` representing the position relative to the `targetElement`; must match (case-insensitively) one of the following strings:

- `'beforebegin'`: Before the `targetElement` itself.
- `'afterbegin'`: Just inside the `targetElement`, before its first child.
- `'beforeend'`: Just inside the `targetElement`, after its last child.
- `'afterend'`: After the `targetElement` itself.

`element`

The element to be inserted into the tree.

```html
<div>Some Other Element</div>

<!-- beforeBegin -->
<div id="container">
  <!-- afterBegin -->

  <div>Some child</div>

  <!-- beforeEnd -->
</div>
<!-- afterEnd -->

<div>Some Other Element</div>
```

cloneNode()

The cloneNode() method of the Node interface returns a duplicate of the node on which this method was called. Its parameter controls if the subtree contained in a node is also cloned or not.

```
1 const htmlContainer = document.getElementById('someContainerId');
2 const template = document.getElementById('myTemplate');
3
4 // Create a clone of the template - this isn't part of the DOM yet
5 const clone = template.content.cloneNode(true);
6
7 // Get the actual content of the template
8 const el = clone.querySelector('div');
9
10 htmlContainer.appendChild(el); // Or insertAdjacentElement
```

Warning: cloneNode() may lead to duplicate element IDs in a document!

TECH ELEVATOR

Lecture

AGENDA

- Templates
- Events
- Event Listeners
- Event Bubbling
- Default Browser Behavior

Lecture

TEMPLATE TAG

```
1 <template id="myTemplate">
2   <div>
3     <h2>A heading</h2>
4     <p>Some text goes here</p>
5   </div>
6 </template>
```

Lecture

BROWSER EVENT MODEL

Browser events work in what is commonly called a Publish and Subscribe manner. Publish and Subscribe is a programmatic way to pass messages between different parts of a system while keeping those different parts decoupled from each other, meaning that the parts don't have to know about each other, they just need to know which messages to watch out for.

Publishing means that the parts of the system can send messages out for other parts to act on and Subscribe means that a part can listen for certain messages to be published and perform logic in response to it.

Lecture

EVENT HANDLING

Event Handling is a style of user interaction that browsers use to allow developers to react and interact with a user's use of their site.

Lecture

EVENT HANDLER

An **Event Handler** is a function that can be attached to a DOM element and run when a defined type of event happens on that DOM element.

EVENT OBJECTS

An **Event Object** is an object given to every Event Handler that defines properties about that event, like location, which DOM element the event happened on, and key presses or mouse click information.

Listening for events in JavaScript

Listening for events in JavaScript

Reacting to/handling events in JavaScript requires three things:

1. A DOM element that you want to listen to events on
2. A specific event that you want to listen to
3. A function that holds the logic that you want to execute

All DOM elements can receive the following events:

1. Mouse Events
    1. `click` - a user has clicked on the DOM element
    2. `dblclick` - a user has double clicked on the DOM element
    3. `mouseover` - a user has moved their mouse over the DOM element
    4. `mouseout` - a user has moved their mouse out of the DOM element

Input elements, like `<input>`, `<select>`, and `<textarea>`, also trigger these events:

1. Input Events
    1. `keydown` - a user pressed down a key (including shift, alt, etc.) while on this DOM element
    2. `keyup` - a user released a key (including shift, alt, etc.) while on this DOM element
    3. `change` - a user has finished changing the value of this input element
    4. `focus` - a user has selected this input element for editing
    5. `blur` - a user has unselected this input element for editing

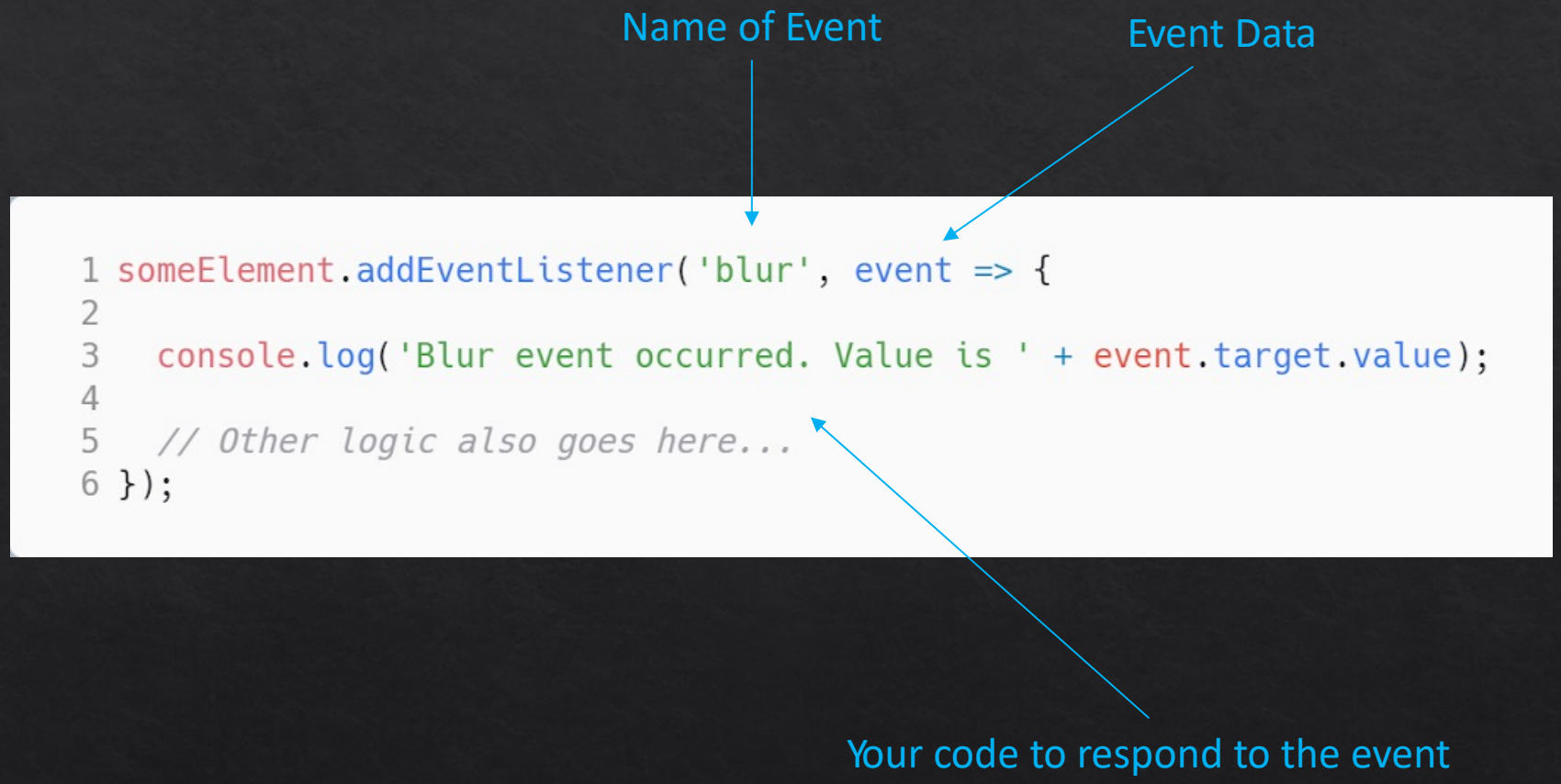
Form elements have these events:

1. Form Events
    1. `submit` - a user has submitted this form using a submit button or by hitting Enter on a text input element
    2. `reset` - a user has reset this form using a reset button

There are many more events that can be listened for, but these are the ones you'll use most of the time. You can find more at the MDN documentation for events.

Lecture

addEventListener



addEventListener

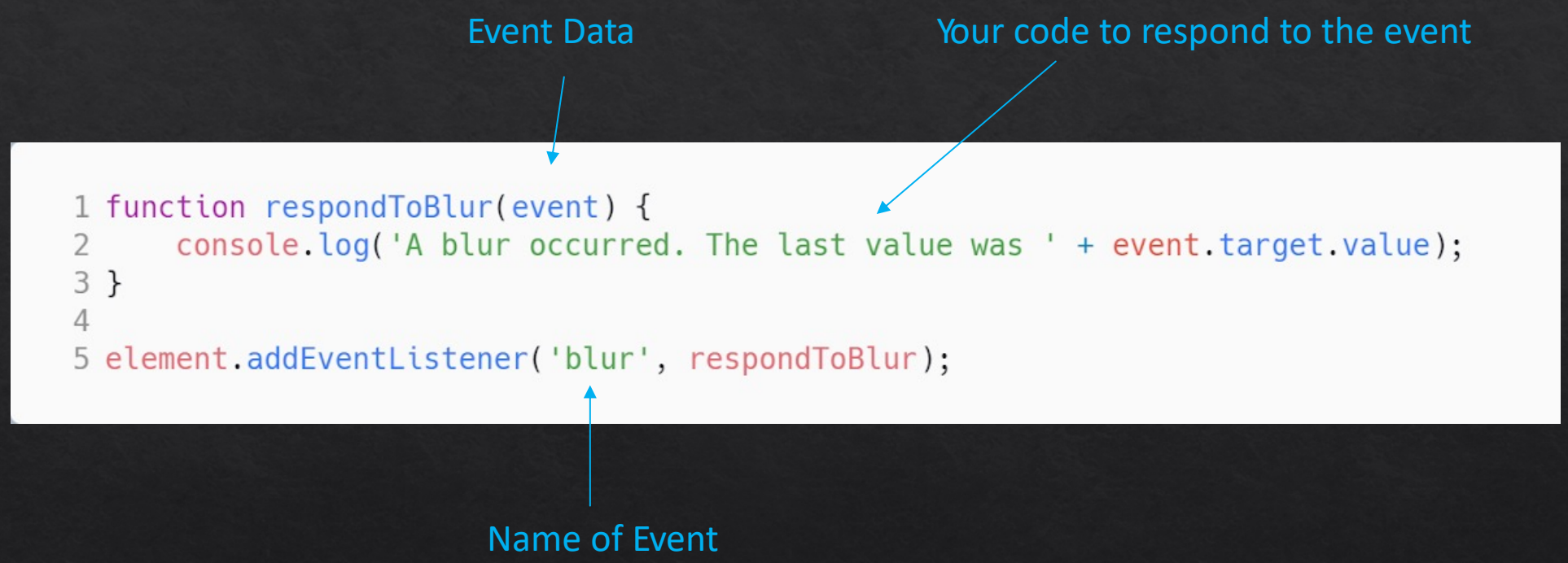
Name of Event

Your code to respond to the event

```
1 someElement.addEventListener('blur', event => console.log('Field blurred. Value is ' + event.target.value));
```

Event Data

addEventListener



Lecture

addEventListener

Adding event handlers to the DOM *

*Best practice

```
function changeGreeting() {
  let greetingHeader = document.getElementById('greeting');
  greetingHeader.innerText = 'Goodbye';
}


let changeButton = document.getElementById('change-greeting');

changeButton.addEventListener('click', (event) => {
  changeGreeting();
});
```

Event handling using anonymous functions

You could also get the same functionality by attaching an anonymous function as the event listener instead of calling a named function:

```
changeButton.addEventListener('click', (event) => {
  let greetingHeader = document.getElementById('greeting');
  greetingHeader.innerText = 'Goodbye';
});
```

Lecture

removeEventListener

```
1 function respondToBlur(event) {
2     console.log('A blur occurred. The last value was ' + event.target.value);
3 }
4
5 element.addEventListener('blur', respondToBlur);
6
7 // Some time later...
8
9 element.removeEventListener('blur', respondToBlur);
```

DOMContentLoaded

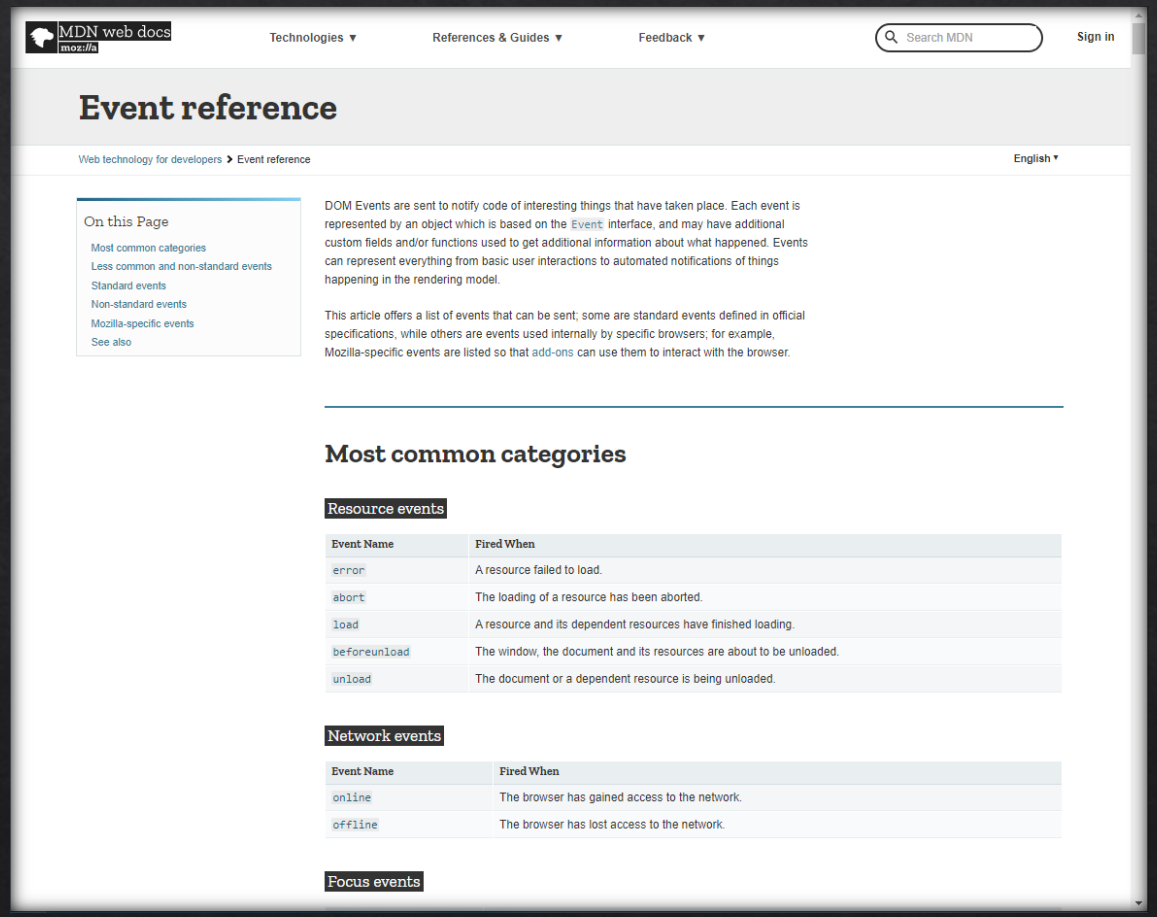
```
1 // This function is called when the DOM is ready to interact with
2 // If we tried to interact with it before this, DOM interaction would fail
3 document.addEventListener('DOMContentLoaded', () => {
4
5   // Your code goes here
6
7 });
```

TECH ELEVATOR

OTHER EVENTS

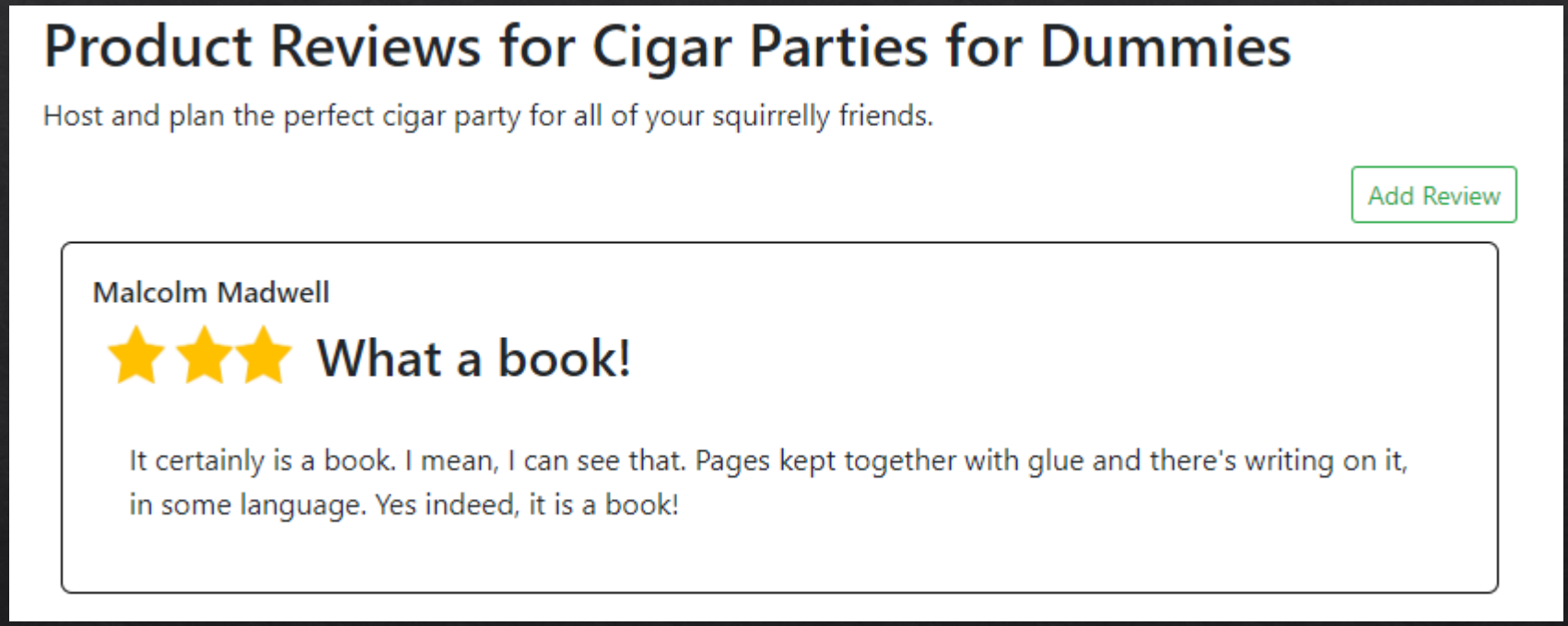
Lecture

- click
- dblclick
- change
- focus
- mouseover
- mouseleave



TECH ELEVATOR

EDITING THE DESCRIPTION, LET'S CODE!



Lecture Code

Lecture

DEFAULT ACTION

The **Default Action** is the default process that a browser will perform if no Event Handler prevents it from happening.

DEFAULT ACTION

What elements have default actions?

- **A click on a link – initiates navigation to its URL.**
- **A click on a form submit button – initiates its submission to the server.**
- Pressing a mouse button over a text and moving it – selects the text.
- Clicking on an input text box selects it
- Double-clicking on text selects it
- (the list goes on)

Lecture

PREVENTING BEHAVIOR

```
1 // Stops default behavior
2 event.preventDefault();
3
```

TECH ELEVATOR

ADDING A NEW REVIEW, LET'S CODE!

Lecture Code

ELEVATE YOURSELF

Product Reviews for Cigar Parties for Dummies

Host and plan the perfect cigar party for all of your squirrelly friends.

Add Review

Malcolm Madwell

★★★ What a book!

It certainly is a book. I mean, I can see that. Pages kept together with glue and there's writing on it, in some language. Yes indeed, it is a book!

Lecture

EVENT PROPAGATION

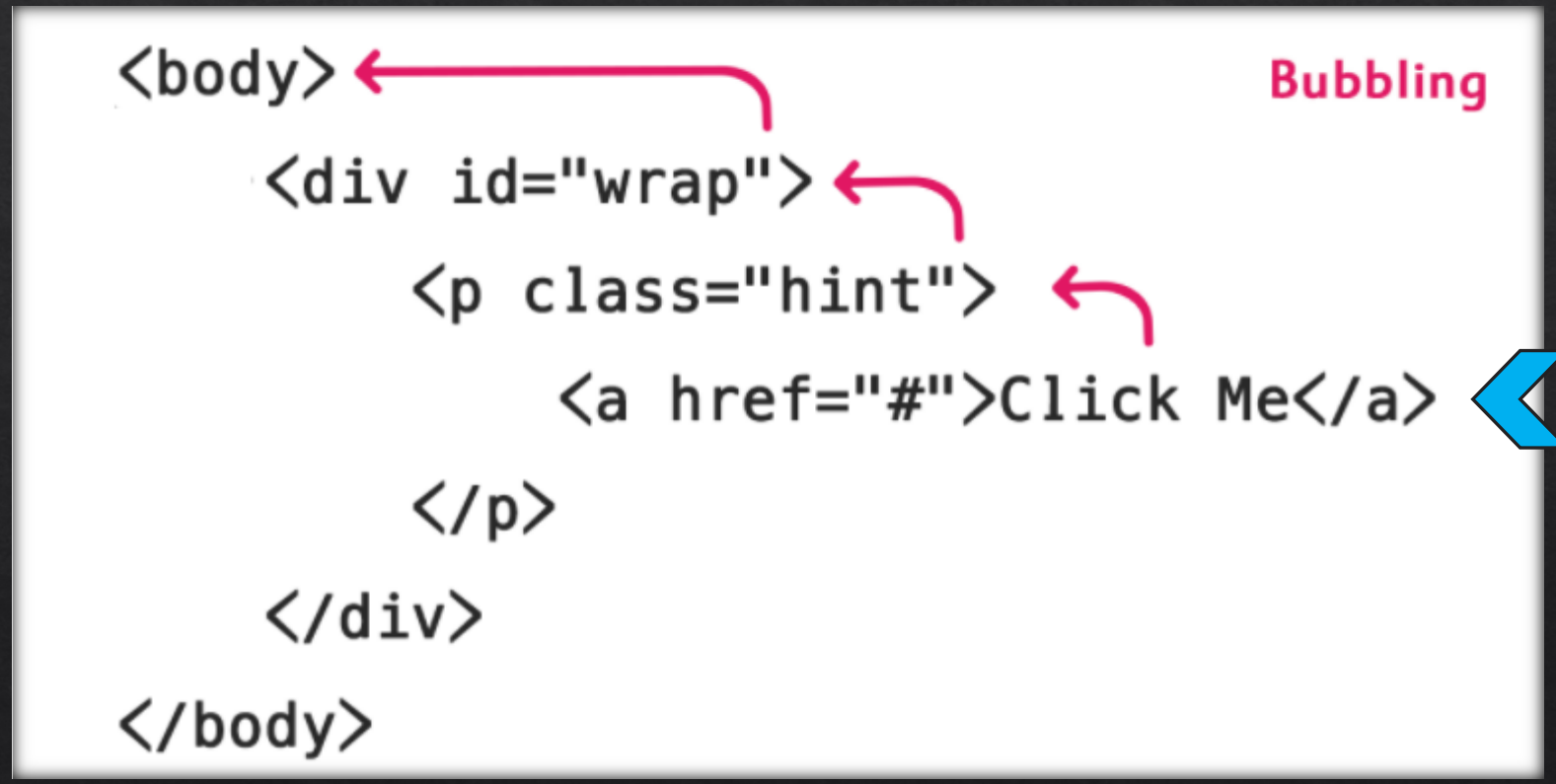
Event Propagation is the process of an event firing on a DOM element and every parent of the DOM element up to the window object.

TECH ELEVATOR

Lecture

EVENT BUBBLING

EVENT BUBBLING



STOPPING PROPAGATION

```
1 // Stops default behavior
2 event.preventDefault();
3
4 // Stops the event from propagating upwards
5 event.stopPropagation();
```

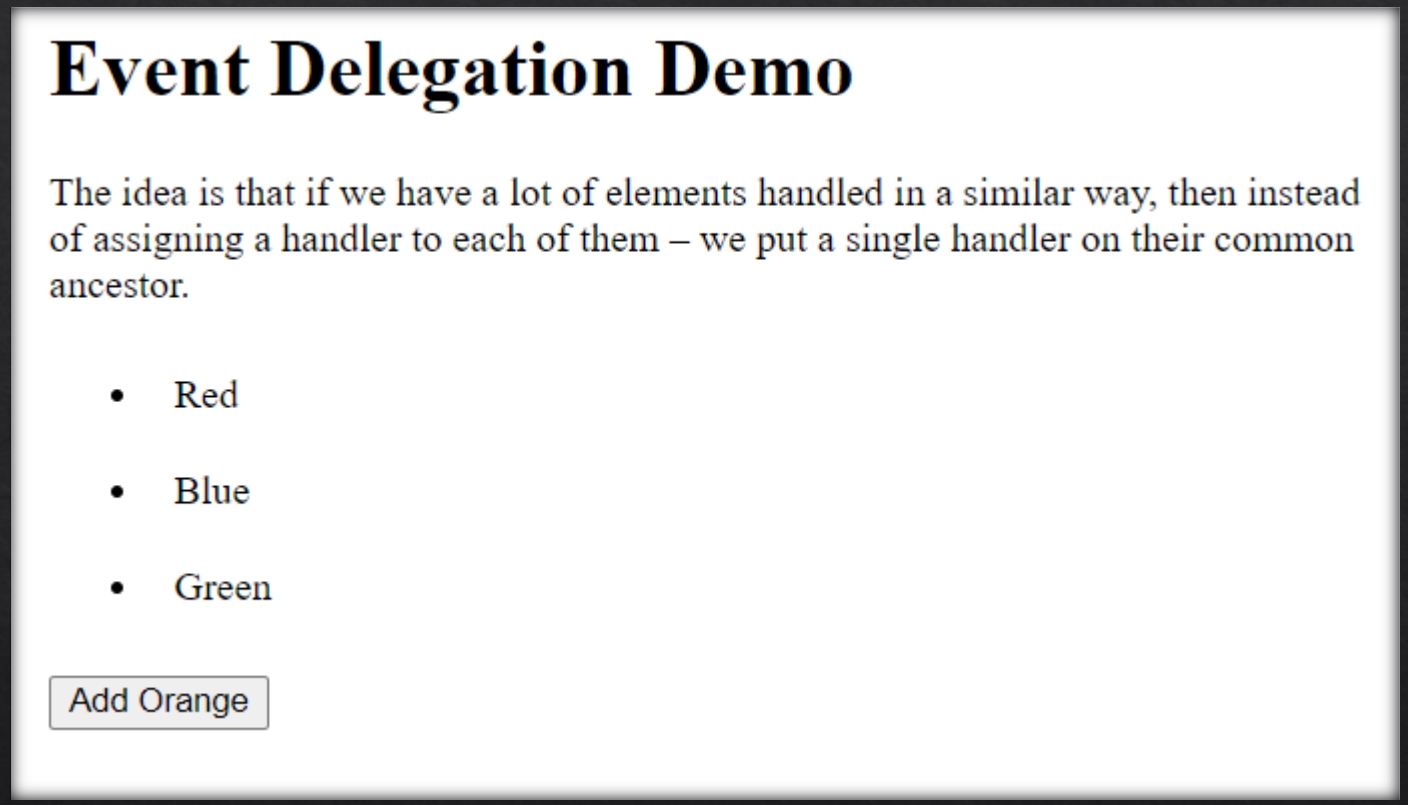
EVENT DELEGATION

Event Delegation – Placing a single handler on a common ancestor when we have a lot of elements that are handled in a similar way. This is done instead of assigning a handler to each of them.


```
<ul id="colors">
  <li>Red</li>
  <li>Blue</li>
  <li>Green</li>
</ul>
```

TECH ELEVATOR

EVENT DELEGATION DEMO, LET'S CODE!



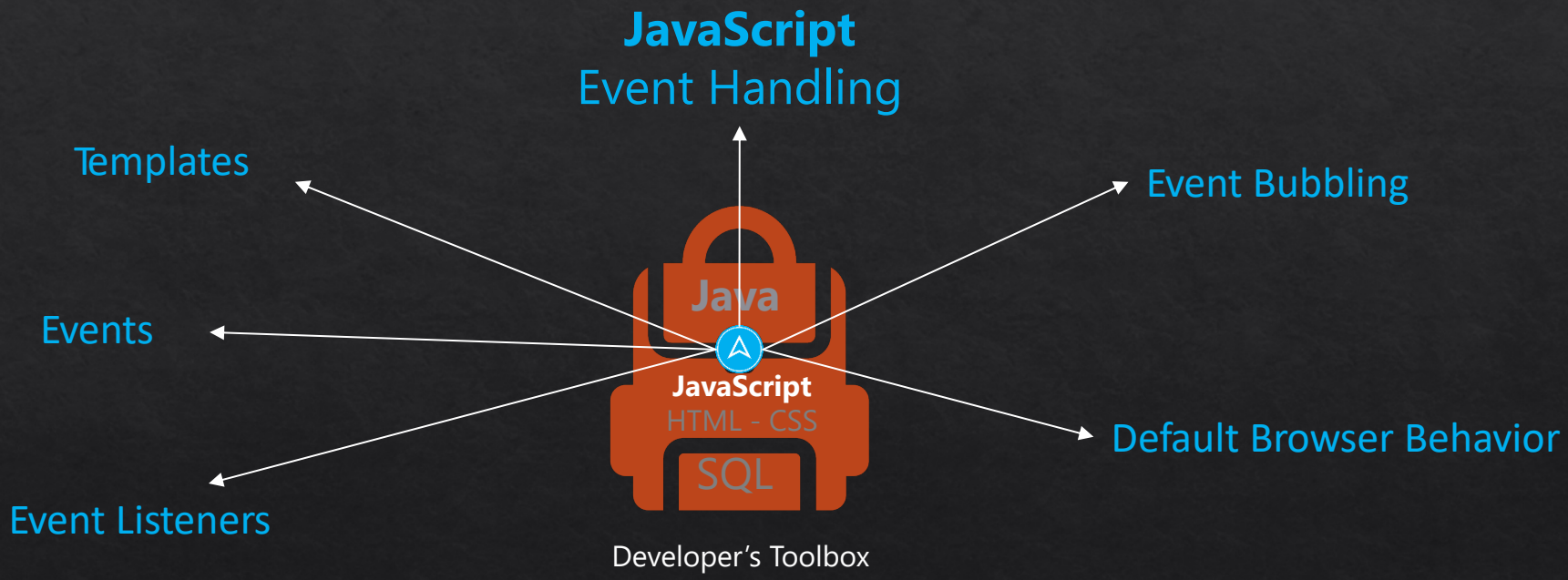
Lecture Code

TECH ELEVATOR

Module 3

JavaScript – Event Handling

New Tools! Let's Code!



Exercise HW