



*Welcome to
lecture 16!*

Agenda

Session Objectives

- Understand the concept of events in web development
- Learn how to "listen" for events using `addEventListener()`
- Write event handler functions to respond to events.
- Explore common event types (click, mouseover, keydown, etc.).
- Understand the basics of the event object.
- Learn how to stop default actions (e.g., form submission).
- Quiz



Introduction to Events

Introduction to Events

- What is Events?
 - Events are actions or occurrences that happen in the browser
- How do events get triggered?
 - User actions: Clicking a mouse button, pressing a key, moving the mouse
 - Browser actions: Page finishing loading, a video finishing playing, an error occurring
- When an event occurs on an HTML element, the browser can be told to run a specific piece of JavaScript code
- Think of events as signals: "Hey, something just happened here!"

Listening for Events

- To make an element react, we first need to tell JavaScript to "listen" for specific events on that element
- The primary method:
 - `element.addEventListener('eventType', handlerFunction);`
 - `element`: The DOM element you want to listen on (e.g., a button, a paragraph).
 - `'eventType'`: A string specifying the name of the event to listen for (e.g., 'click', 'mouseover').
 - `handlerFunction`: The function to be called when the event occurs on the element. This is also known as a "callback function."

```
const myButton = document.getElementById('myBtn');  
myButton.addEventListener('click', function() {  
  console.log('Button was clicked!');  
});
```

Event Handler

- The function you provide to `addEventListener` is the event handler or callback function
- This function contains the code that will run when the specified event occurs
- It can be defined in two main ways:
 - An anonymous function: Defined directly inside `addEventListener`. Good for simple, one-off actions
 - Eg: `element.addEventListener('click', function() { /* code for click */ });`
 - A named function: Defined separately and then passed by its name. Better for reusability or more complex logic
 - Eg:
 - `function handleClick() { console.log('Named handler executed!'); }`
 - `element.addEventListener('click', handleClick); // IMPORTANT: No parentheses () here!`
 - Why no parentheses when passing a named function?
 - We are passing a reference to the function, not calling it immediately
- Let's write a handler!



Common Event Types

Common Event Types - Beyond Clicks!

- Events can be of many different types. What are a few you can think of?
- Common event types
 - Mouse Events: interactions with mouse events
 - click and dblclick (double click)
 - mousedown (button pressed) and mouseup (button released)
 - mouseover (pointer enters element), mouseout (pointer leaves element)
 - mousemove (pointer moves over element)
 - Keyboard Events: Interactions with the keyboard
 - keydown (key pressed), keyup (key released)
 - keypress (key that produces a character is pressed - less common now, keydown often preferred)
 - Form Events: Interactions with form elements.
 - submit (form submitted)
 - focus (element gains focus), blur (element loses focus)

Common Event Types (Contd.)

- Form Events (Contd.)
 - change (value of input/select/textarea changes and loses focus)
 - input (value of input/textarea changes - fires immediately)
- Window/Document Events: load, DOMContentLoaded, resize, scroll
 - load (entire page & resources loaded)
 - DOMContentLoaded (HTML loaded and parsed, DOM ready)
 - resize (browser window resized), scroll (document scrolled)
- Don't worry about memorizing them. Use the [documentation](#) and practice!

Exercise: The "Magic Light Switch"!

- Goal: Create a div that acts like a light switch
- Requirements:
 - When you click the div:
 - If it's "OFF", change its background color to yellow and its text to "ON".
 - If it's "ON", change its background color to grey and its text to "OFF".
- Hint: Use `classList.toggle()` for a style class, or change `style.backgroundColor` directly. Check `textContent` to know the current state.



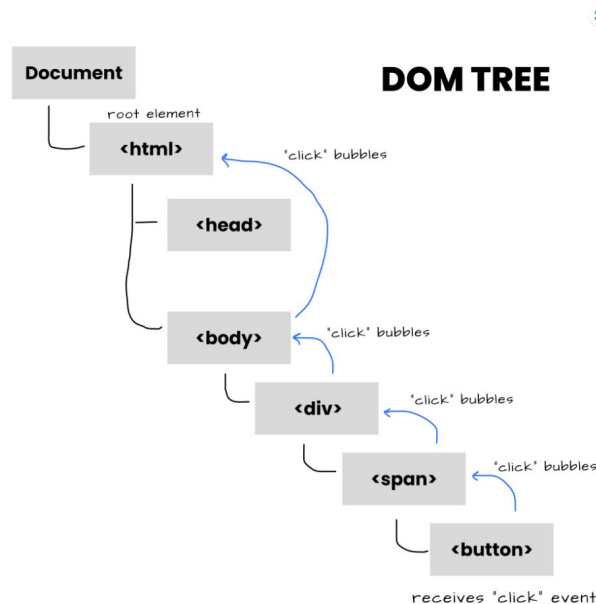
What's the Event Object?

The Event Object: Information Carrier

- When an event occurs and your handler function is called, it automatically receives an event object as its first argument.
- A handler function may receive the event as follows:
 - `function myHandler(event) { /* ... */ }`
- This object contains valuable information about the event:
 - `event.target`: The DOM element that triggered the event (where the event originated).
 - `event.type`: The type of event (e.g., 'click', 'keydown').
 - `event.key` / `event.code` (for keyboard events).
 - `event.clientX` / `event.clientY` (for mouse events - cursor position).
 - And much more!
- Always `console.log(event)` in your handlers when learning – it's your best friend!

Event Propagation: Bubbling

- What is event bubbling?
 - Concept in DOM where an element receives an event and that event bubbles up until it reaches the root element
 - This default behavior can be prevented using `event.stopPropagation()`
 - Benefit: Instead of adding event listeners to multiple child elements, you can attach a single listener to their parent. This helps reduce unnecessary event listeners and improves efficiency



Event Propagation: Example

Example 1 (bubbling up)

```
<div id="container">
  <p>Click the button below:</p>
  <button id="myButton">Click Me!</button>
</div>
```

```
// Handler for the container <div>
document.getElementById('container')
  .addEventListener('click', function (event) {
    console.log("Container handler: clicked on " + event.target.tagName);
  });

// Handler for the <button>
document.getElementById('myButton')
  .addEventListener('click', function (event) {
    console.log("Button handler: clicked on " + event.target.tagName);
  });
```

Example 2 (stopPropagation)

```
<div id="outer">
  Outer button
  <button id="inner"> Inner button </button>
</div>
```

```
const outer = document.getElementById('outer');
const inner = document.getElementById('inner');

outer.addEventListener('click', () => {
  console.log('Outer div handler fired');
});

inner.addEventListener('click', e => {
  console.log('Inner button handler fired');
  // Prevent this click from bubbling up to the outer div
  e.stopPropagation();
});
```

event.target - Who Triggered Me?

- event.target refers to the actual DOM element on which the event originally occurred
- Example: If you have multiple buttons using the same click handler, event.target will tell you which specific button was clicked.

```
<button class="action-btn">Action 1</button>  
<button class="action-btn">Action 2</button>
```

```
// javascript  
const buttons = document.querySelectorAll('.action-btn');  
buttons.forEach(button => {  
  button.addEventListener('click', function(event) {  
    console.log(event.target.textContent + ' was clicked!');  
  });  
});
```

Stopping Default Actions: event.preventDefault()

- Some HTML elements have default behaviors for certain events:
 - Clicking a link (<a> tag with href): Navigates to the URL.
 - Submitting a form (<form> by clicking <button type="submit"> or pressing Enter): Reloads the page and attempts to send data to a server.
 - Pressing spacebar when a button is focused: "Clicks" the button.
- To handle the event without the default browser action occurring, you can use event.preventDefault() inside the event handler

```
<form id="myForm">
  <label for="nameInput">Name:</label>
  <input type="text" id="nameInput" required>
  <button type="submit">Submit with JS</button>
</form>
<div id="formMessage"></div>
```

```
const myForm = document.getElementById('myForm');
const nameInput = document.getElementById('nameInput');
const formMessage = document.getElementById('formMessage');

myForm.addEventListener('submit', function(event) {
  event.preventDefault(); // Stop page reload!
  const userName = nameInput.value;
  if (userName) {
    formMessage.textContent = 'Form submitted via JS! Hello, ' + userName + '!';
    console.log('Form submission prevented. Name:', userName);
    // Here you would typically send data via AJAX/fetch in a real app
  } else {
    formMessage.textContent = 'Please enter your name.';
  }
  myForm.reset(); // Optionally reset the form fields
});
```

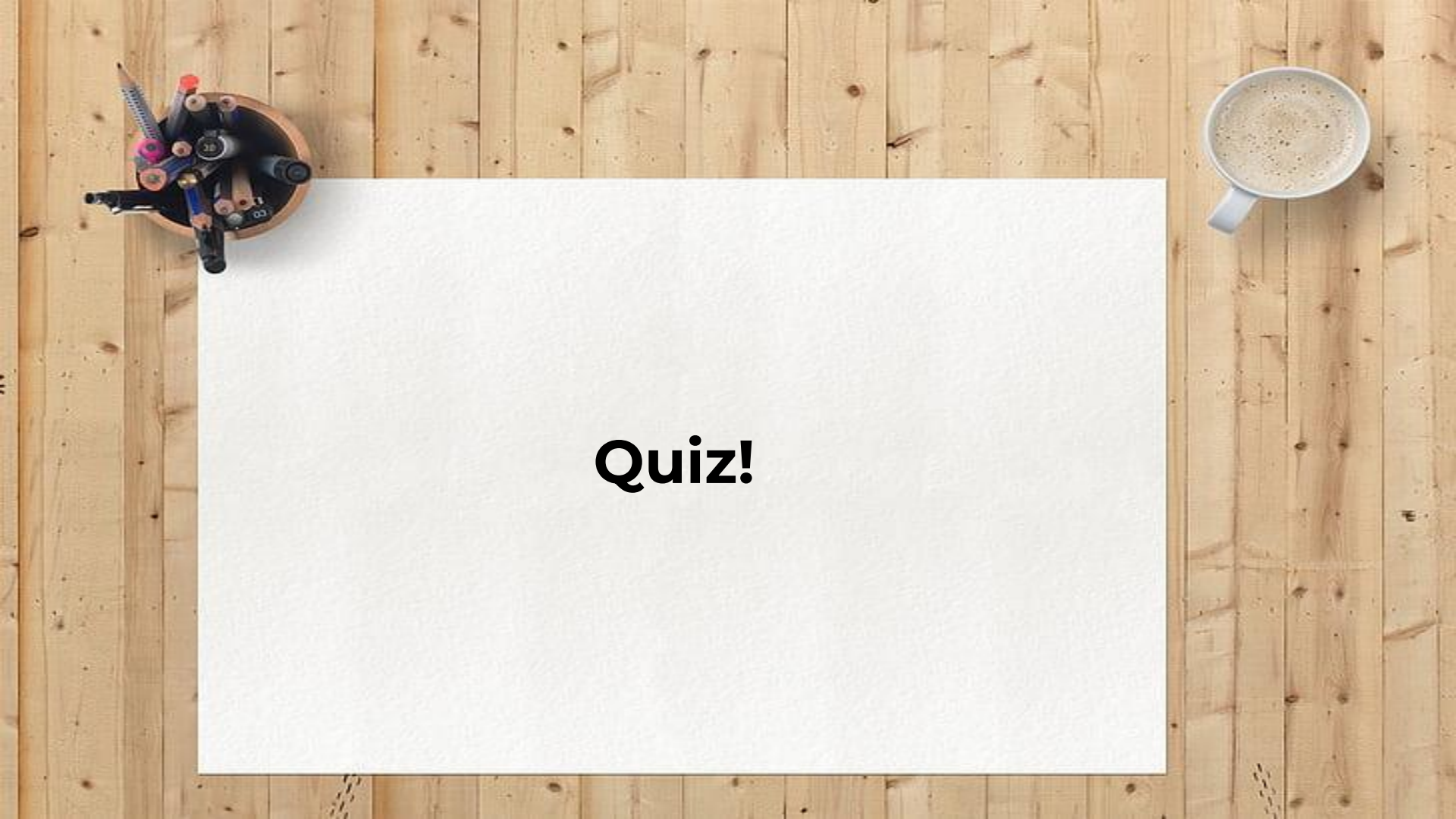

A Quick Look: Removing Event Listeners

- Sometimes you might want to stop listening for an event
 - `element.removeEventListener('eventType', handlerFunction);`
- Important: To remove a listener, you must pass the exact same function reference that was used for `addEventListener`
 - This means anonymous functions cannot be easily removed this way
- Usually needed in more complex scenarios (e.g., applications where components are destroyed)

```
function handleClickOnce() {  
  console.log('Clicked once!');  
  myButton.removeEventListener('click', handleClickOnce);  
}  
myButton.addEventListener('click', handleClickOnce);
```



That's it!
Let's answer your
doubts.

A top-down view of a wooden desk. In the center is a large white rectangular paper. In the top-left corner of the paper is a small wooden bowl filled with various colored pencils and pens. In the top-right corner of the paper is a white mug filled with a frothy beverage. The word "Quiz!" is written in the center of the white paper.

Quiz!

Question 1

- What is the primary method in JavaScript to make an HTML element listen for an event?
 - A) `element.addEventListener('click', myFunction)`
 - B) `element.listenFor('click', myFunction)`
 - C) `element.attachEvent('onclick', myFunction)`
 - D) `element.on('click', myFunction)`

Correct Answer: A

Question 2

- When an event handler function is called, what is automatically passed as its first argument?
 - A) The HTML element that was clicked
 - B) The name of the event type as a string
 - C) An event object containing details about the event
 - D) Nothing is passed automatically

Correct Answer: C

Question 3

- Which of these is a common mouse event type?
 - A) submit
 - B) mouseover
 - C) keydown
 - D) load

Correct Answer: B

submit is a form event

keydown is a keyboard event

load is a window/document event.

Question 4

- If you have a form and you want to handle its submission with JavaScript without the page reloading, what should you call inside your 'submit' event handler?
 - A) `event.stopReload()`
 - B) `event.preventDefault()`
 - C) `event.cancelSubmit()`
 - D) `form.pause()`

Correct Answer: B

Question 5

- What does `event.target` refer to inside an event handler function?
 - A) The type of event that occurred
 - B) The JavaScript function that handled the event
 - C) The HTML element that the event listener was attached to
 - D) The HTML element that originally triggered the event

Correct Answer: D

Question 7

- Why is it generally necessary to use a named function if you plan to use `removeEventListener()`?
 - A) Anonymous functions don't have access to the event object.
 - B) `removeEventListener` requires the exact same function reference that was passed to `addEventListener`.
 - C) Named functions are more performant for event handling.
 - D) Anonymous functions cannot be used as event handlers.

Correct Answer: B

Question 8

- If you attach an event listener to an `<input type="text">` for the 'keydown' event, and inside the handler you check `event.target.value`, what will you typically get?
 - A) The value of the input field before the current key press is processed
 - B) The value of the input field after the current key press is processed
 - C) An empty string, as value is only updated on keyup
 - D) The event.key that was just pressed

Correct Answer: A

In the bubbling phase (default), the event fires on the most deeply nested target first (button), then "bubbles up" to its ancestors (div)

Homework: The Mood Changer

- Create a page that changes its appearance and displayed "mood" based on button clicks
- [Base code in Google Doc](#)
- Features:
 - Have at least three buttons, each representing a mood (eg: Happy, Sad, Excited)
 - A central div that displays the current mood as text (e.g., "Current Mood: Happy")
 - When a mood button is clicked:
 - The text in the central div updates to the clicked mood
 - The document.body background color changes to reflect the mood (e.g., yellow for happy, blue for sad, orange for excited)
 - (Optional) An emoji or an image related to the mood changes in another div
 - Ensure a default mood is displayed when the page loads
- [Submission Link](#)