Event Handling & Form Submission

Software Development Bootcamp

Topic

Handling Events

What Are Event Listeners?

Event listeners are a way for HTML elements to respond to certain events that are relevant to the element.

- Buttons listen for onclick
- Forms listen for onsubmit
- DOM Nodes have an addEventListener method for custom circumstances

Why Use Event Listeners?

- **Interactivity:** React to user interactions such as clicks, key presses, mouse movements, and form submissions
- **Form Validation:** Validate form inputs in real-time, providing immediate feedback to users
- **Asynchronous Operations:** Trigger asynchronous operations like API calls based on user actions
- **User Experience:** By responding to user actions immediately, event listeners help create a more responsive and engaging user interface.

What Is `addEventListener`?

- Method available on DOM elements that allows you to attach an event handler to a specific event for that element.
- Provides a flexible way to handle user interactions and other events in your application.

`addEventListener` Syntax

```
element.addEventListener(eventType, eventHandler, options);
```

- **eventType:** A string that specifies the type of event (e.g., `'click'`, `'keydown'`, `'submit'`)
- **eventHandler:** A function to be called when the event occurs
- **options (optional):** An object that specifies characteristics about the event listener

Common Event Types

- `'click'`: Fires when an element is clicked
- `'submit'`: Fires when a form is submitted
- `'keydown'`, `'keyup'`: Fire when a key is pressed down or released
- `'mouseover'`, `'mouseout'`: Fire when the mouse enters or leaves an element
- `'focus'`, `'blur'`: Fire when an element gains or loses focus
- `'load'`: Fires when a page or an element has finished loading

Handling Events

- Here our event type is `click` and our event handler function sends and alert with the string “Abracadabra”

```
// Grab the relevant element from the DOM
let button =
document.getElementById("magic");
// Use addEventListener to tell button how
to handle being clicked
button.addEventListener("click", () => {
 alert("Abracadabra!");
});
```

Handling Events

- If you have already defined an event handler function, you can attach it by reference

```
// Grab the relevant element from the DOM
let button = document.getElementById("magic");
// Use addEventListener to tell button how to
handle being clicked
function sayMagicWord() {
   alert("Shazam!")
}
// Referencing the sayMagicWord function
button.addEventListener("click", sayMagicWord)
```

Event Handler Parameters

- The browser passes an Event object to your handler function as its first argument
- Often see as parameter `e`
- `e.target` represents the node where the Event occured

Upright Education

html

```html
<body>
        <button type="button"
id="presto">Presto...</button>
        <button type="button"
id="abra">Abra...</button>
        </script><script
src="index.js"></script>
    </body>
```

js

```js
let prestoBtn = document.getElementById('presto')
let abraBtn = document.getElementById('abra')
// event handler function
function sayMagicWord(e) {
   if (e.target === prestoBtn) {
       alert("Change-o!")
   } else if (e.target === abraBtn) {
       alert('Cadabra!')
   } else {
       alert("Shazam!")
   }
   //console.log for debugging
   console.log({ e })
}
// Adding event handler function to event
prestoBtn.addEventListener('click', sayMagicWord)
abraBtn.addEventListener('click', sayMagicWord)
```

Event Handler Parameters

`sayMagicWord()` Explained

- Takes an event object `(e)` as its parameter
- Uses conditional statements to check which button was clicked
  - If the target of the event `(e.target)` is the `prestonBtn` it alerts “Change-O!”
  - If the `abraBtn` is clicked it alerts “Cadabra!”
- Finally it logs the entire event object to the console for debugging

```
js
let prestoBtn = document.getElementById('presto')
let abraBtn = document.getElementById('abra')
// event handler function
function sayMagicWord(e) {
    if (e.target === prestoBtn) {
        alert("Change-o!")
    } else if (e.target === abraBtn) {
        alert('Cadabra!')
    } else {
        alert("Shazam!")
    }
    //console.log for debugging
    console.log({ e })
}
// Adding event handler function to event
prestoBtn.addEventListener('click', sayMagicWord)
abraBtn.addEventListener('click', sayMagicWord)
```

Event Bubbling

- Describes the order in which events are received on the page: from the target element where the event occurs, up through its parent elements in the DOM hierarchy

```
html
  <div id="outer">
          OUTER
          <div id="middle">
            MIDDLE
            <button id="inner">INNER</button>
          </div>
       </div>
```

```
js
document.getElementById('outer').addEventListener('
click', () => {
   console.log('Outer div clicked');
 });
document.getElementById('middle').addEventListener(
'click', () => {
   console.log('Middle div clicked');
 });
document.getElementById('inner').addEventListener('
click', () => {
   console.log('Button clicked');
 });
```

Topic

Built-In Events And Forms

Built-In Events

Some HTML elements have default interactive behaviors

- Buttons: `onClick`
- Anchors: `onClick`
- Forms: `onsubmit`

Anchors

- `<a>` elements listen for onclick events so it knows when to send you to the linked location.
- We can link to other pages or other spots within the same page.

Internal Anchors

- We can use the id attribute of elements to use **<a>** to send us to a different part of the same page.
- We use the id selector # in the href attribute to do this.

```
<body>
   <a href="#pizza">🍕</a>
   <a href="#burger">🍔</a>
   <h2 id="pizza">Clicking 🍕 sends you here.</h2>
   <p>Content in between gets skipped, since that's the point.</p>
   <h2 id="burger">Clicking 🍔sends you here.</h2>
 </body>
```

What Are Forms?

Forms are a crucial component of web development, allowing users to input data that can be sent to a server for processing. They provide a structured way to collect information from users and facilitate interaction between users and websites or web applications.

Why Use Forms?

- User registration and login
- Collecting user feedback
- Making purchases (e-commerce)
- Searching for information
- Submitting data for processing

Form Structure

- The `<form>` tag encapsulates all form elements
- Input fields (text, passwords, checkbox etc.)
- Select Dropdowns
- Textareas for longer text input
- Submit buttons

Basic HTML Form

- `action:` Specifies where to send the form data when submitted
- `method:` Defines the HTTP method to use when submitting the form
- `name:` Identifies form controls for reference when the form is submitted
- `required:` Specifies that an input field must be filled out before submitting the form

```
form action="/submit-url" method="post">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username"
required>
    <label for="password">Password:</label>
    <input type="password" id="password"
name="password" required>
    <input type="submit" value="Submit">
</form>
```

Submit Handler

- `e.preventDefault()`: Prevents the default form submission behavior, which would normally refresh the page
- `form.username`: Accesses the input element directly by its `name` attribute
- `.value`: Gets the current value entered in that input field

```
js
let form = document.querySelector("form")
// adding event listener to form with event type of 'submit'
form.addEventListener('submit', (e) => {
    // prevent the browser from refreshing the page
    e.preventDefault()
    // user variable to hold the username input value
    const user = form.username.value
    console.log(user)
    // reset the form inputs so they are blank
    form.reset()
})
```

Exercise

Mission Countdown