Advanced JavaScript Mastery (continuation)
Event Handling and User Interactions in JavaScript

Unlock the interactivity of your web applications by mastering JavaScript events. Understand event propagation, control event behavior, and implement efficient event handling for user interactions. Optimize performance with advanced techniques like event delegation.

1. Introduction to Events and Event Listeners

What Are Events in JavaScript? Events in JavaScript represent actions or occurrences that happen in the browser, which can be triggered by user interactions (such as clicking a button, submitting a form, or pressing a key) or system-generated events (like loading a page or completing a network request).

JavaScript provides mechanisms to handle these events and execute specific code in response, making your web application interactive and dynamic.

Common Event Types

Event Type	Description	Example
Click Events (click)	Triggered when a user clicks on an element, like a button or a link.	Clicking a button to submit a form or open a modal window.
Submit Events (submit)	Triggered when a user submits a form, allowing you to handle form validation or prevent default submission.	Submitting a sign-up or login form.
Keypress Events (keypress, keydown, keyup)	Triggered when a user presses or releases a key on the keyboard.	Detecting when a user types in an input field or presses a specific key combination.
Mouse Events (mouseover, mouseout, mousemove)	Triggered by mouse actions, such as hovering over an element or moving the mouse pointer.	Hovering over a dropdown to open it.

Focus Events (focus, blur)	Triggered when an element gains or loses focus.	Highlighting an input field when it is selected.
Load Events (load)	Triggered when the browser finishes loading a resource, such as an image or a webpage.	Running a function after the page has fully loaded.

Explanation of addEventListener() and How to Attach Event Listeners

To respond to events in JavaScript, you attach an **event listener** to an element. The most common method to do this is with the addEventListener() function, which listens for specific events and calls a function when the event occurs.

```
element.addEventListener('event', function, useCapture);
```

- event: The type of event you want to listen for (e.g., 'click', 'submit');
- function: The callback function to execute when the event is triggered;
- useCapture (optional): A boolean that indicates whether the event should be captured during the capturing or bubbling phase (default is false, meaning it listens during the bubbling phase).

Click Event Listener

Let's attach a click event listener to the button. When clicked, it triggers an alert.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <button id="myButton">Click Me</button>
    <script src="index.js"></script>
  </body>
</html>
```

index.js

```
// Access the button element
const button = document.getElementById('myButton');

// Add an event listener for the 'click' event
```

```
button.addEventListener('click', () => {
  alert('Button clicked!');
});
```

index.css

```
#myButton {
  padding: 12px 30px;
  background: linear-gradient(135deg, #6a11cb, #2575fc);
  color: #ffffff;
  border: none;
  border-radius: 50px;
  font-size: 16px;
  font-weight: bold;
  text-transform: uppercase;
  letter-spacing: 2px;
  cursor: pointer;
  transition: all 0.3s ease;
  box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);
}

#myButton:hover {
  background: linear-gradient(135deg, #2575fc, #6a11cb);
  transform: scale(1.05);
  box-shadow: 0 6px 20px rgba(0, 0, 0, 0.3);
}

#myButton:active {
  background: linear-gradient(135deg, #1c1c1c, #343434);
  transform: scale(0.98);
  box-shadow: 0 3px 10px rgba(0, 0, 0, 0.4);
}
```

The event listener is attached to the `#myButton` element, and when the user clicks the button, a message box appears with the text `"Button clicked!"`

Submit Event Listener

Let's add an event listener for the `submit` event, preventing the default form submission behavior to handle the data with JavaScript.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <form id="myForm">
      <input type="text" id="name" placeholder="Enter your name" />
      <button type="submit">Submit</button>
```

```
    </form>
    <p id="displayName"> </p>
    <script src="index.js"></script>
  </body>
</html>
```

index.js

```
const form = document.getElementById('myForm');
const displayName = document.getElementById('displayName');

// Add an event listener for form submission
form.addEventListener('submit', event => {
  event.preventDefault(); // Prevents the form from submitting

  const name = document.getElementById('name').value;

  // Display the name in the paragraph
  displayName.textContent = `Submitted Name: ${name}`;
});
```

index.css

```
body {
  font-family: 'Arial', sans-serif;
  background-color: #f4f4f4;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  padding: 20px;
}

#myForm {
  display: flex;
  flex-direction: column;
  gap: 10px;
  background-color: #ffffff;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  width: 100%;
  max-width: 400px;
}

#myForm input[type='text'] {
  padding: 12px;
  font-size: 16px;
  border: 1px solid #b5d7b7;
  border-radius: 6px;
  transition: border-color 0.3s ease;
```

```
}

#myForm input[type='text']:focus {
  border-color: #2e7d32;
  outline: none;
  box-shadow: 0 0 5px rgba(46, 125, 50, 0.2);
}

#myForm button {
  padding: 12px;
  background-color: #43a047;
  color: #ffffff;
  border: none;
  border-radius: 6px;
  font-size: 16px;
  cursor: pointer;
  transition: background-color 0.3s ease, transform 0.2s ease;
}

#myForm button:hover {
  background-color: #388e3c;
  transform: translateY(-2px);
}

#myForm button:active {
  background-color: #2e7d32;
  transform: translateY(0);
}

#displayName {
  margin-top: 20px;
  font-size: 18px;
  color: #2e7d32;
  text-align: center;
  font-weight: bold;
  transition: opacity 0.3s ease;
}
```

Keypress Event

Let's attach a `keypress` event listener to the input field. When a key is pressed, it will be shown in the paragraph.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="index.css" />
  </head>
```

```
  <body>
    <input type="text" id="nameInput" placeholder="Type something..." />
    <p id="displayKey">You pressed:</p>
    <script src="index.js"></script>
  </body>
</html>
```

index.js

```
const input = document.getElementById('nameInput');
const displayKey = document.getElementById('displayKey');

input.addEventListener('keypress', event => {
  displayKey.textContent = `You pressed: ${event.key}`;
});
```

index.css

```
body {
  font-family: 'Arial', sans-serif;
  background-color: #f0f4f8;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  padding: 20px;
}

#nameInput {
  width: 300px;
  padding: 12px;
  font-size: 16px;
  border: 2px solid #ccc;
  border-radius: 8px;
  transition: border-color 0.3s, box-shadow 0.3s;
  outline: none;
}

#nameInput:focus {
  border-color: #4caf50;
  box-shadow: 0 0 8px rgba(76, 175, 80, 0.3);
}

#displayKey {
  margin-top: 20px;
  font-size: 20px;
  color: #333;
  padding: 10px;
  background-color: #ffffff;
  border-radius: 8px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
```

```
  transition: transform 0.3s ease, opacity 0.3s ease;
  transform: scale(0.9);
}

#displayKey.visible {
  opacity: 1;
  transform: scale(1);
}
```

Every time a key is pressed while typing into the input field, the pressed key is logged to the console.

Practical Example: Handling Form Validation on Submit

Let's create a logic for the form validation before the form submission.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <form id="signup-form">
      <input type="text" id="username" placeholder="Username" />
      <input type="password" id="password" placeholder="Password" />
      <button type="submit">Sign Up</button>
      <p id="error-message" style="display: none">Please fill in all fields</p>
      <div id="submitted-data" style="display: none">
        <p id="username-data"></p>
        <p id="password-data"></p>
      </div>
    </form>
    <script src="index.js"></script>
  </body>
</html>
```

index.js

```
const form = document.getElementById('signup-form');
const errorMessage = document.getElementById('error-message');
const submittedData = document.getElementById('submitted-data');
const usernameData = document.getElementById('username-data');
const passwordData = document.getElementById('password-data');

form.addEventListener('submit', event => {
  const username = document.getElementById('username').value.trim();
  const password = document.getElementById('password').value.trim();
```

```
  if (!username || !password) {
    event.preventDefault(); // Prevent form submission if fields are empty
    errorMessage.style.display = 'block'; // Show error message
    submittedData.style.display = 'none'; // Hide submitted data if there is an
error
  } else {
    event.preventDefault(); // Prevent default submission to show the data
    errorMessage.style.display = 'none'; // Hide error message
    submittedData.style.display = 'block'; // Show the submitted data
    usernameData.textContent = `Username: ${username}`;
    passwordData.textContent = `Password: ${password}`;

    // Reset the form after displaying the data
    form.reset();
  }
});
```

index.css

```
body {
  font-family: 'Arial', sans-serif;
  background-color: #f4f7f9;
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  padding: 20px;
}

#signup-form {
  background-color: #ffffff;
  padding: 30px;
  border-radius: 10px;
  box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
  display: flex;
  flex-direction: column;
  gap: 15px;
  width: 350px;
}

#signup-form input {
  padding: 12px;
  font-size: 16px;
  border: 2px solid #ddd;
  border-radius: 8px;
  transition: border-color 0.3s, box-shadow 0.3s;
}

#signup-form input:focus {
  border-color: #4caf50;
```

```
  box-shadow: 0 0 5px rgba(76, 175, 80, 0.2);
  outline: none;
}

#signup-form button {
  padding: 12px;
  background-color: #4caf50;
  color: #ffffff;
  border: none;
  border-radius: 8px;
  font-size: 16px;
  cursor: pointer;
  transition: background-color 0.3s, transform 0.2s;
}

#signup-form button:hover {
  background-color: #388e3c;
  transform: translateY(-3px);
}

#signup-form button:active {
  background-color: #2e7d32;
  transform: translateY(0);
}

#error-message {
  display: none;
  color: #ff4c4c;
  font-weight: bold;
  background-color: #ffe6e6;
  padding: 10px;
  border-radius: 6px;
  text-align: center;
}

#submitted-data {
  display: none;
  font-size: 16px;
  color: #4caf50;
  font-weight: bold;
  background-color: #e8f5e9;
  padding: 10px;
  border-radius: 6px;
  text-align: left;
}

#submitted-data p {
  margin: 5px 0;
}
```

The submit event listener ensures the form does not submit if the required fields are empty. If either the username or password field is empty, the event is prevented from proceeding, and an error message is displayed.

2. Understanding the Event Object in JavaScript

Whenever an event occurs in JavaScript, an event object is automatically passed to the event handler. This object contains important information about the event, such as the element that triggered it (target), the type of event (type), and methods to control the behavior of the event, like preventDefault() and stopPropagation().

Accessing Event-Related Information

The event object contains several useful properties and methods that allow you to access information about the event and interact with it.

- **target**

The target property refers to the element that triggered the event. It helps us determine which element was clicked, hovered, or interacted with.

event.target is used to identify the specific button that was clicked, and its id is displayed in the paragraph with the ID display.

- **type**

The type property provides the type of event that occurred (e.g., click, submit, keydown).

The event.type is displayed in the paragraph for feedback, showing that a click event has occurred. Also, when the button is clicked, the background color of the entire body is changed to lightblue.

- **preventDefault() Method in the Event Object**

The preventDefault() method stops the default behavior of an element, such as stopping a link from navigating or preventing a form from submitting.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <p id="display">Clicked button ID will appear here.</p>
    <button id="btn1">Button 1</button>
    <button id="btn2">Button 2</button>
```

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

index.js

```
const display = document.getElementById('display');

document.addEventListener('click', event => {
  if (event.target.tagName === 'BUTTON') {
    display.textContent = `Event triggered by: ${event.target.id}`;
  }
});
```

index.css

```
body {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  gap: 20px;
}

button {
  padding: 12px 20px;
  font-size: 16px;
  color: #ffffff;
  background-color: #4caf50;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  transition: background-color 0.3s, transform 0.2s ease;
}

button:hover {
  background-color: #388e3c;
}

button:active {
  background-color: #2e7d32;
}

#display {
  font-size: 18px;
  color: #333;
  font-weight: bold;
  background-color: #ffffff;
  border-radius: 8px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
  width: auto;
  text-align: center;
```

```
  transition: transform 0.3s ease, opacity 0.3s ease;
  opacity: 0.7;
}

#display.update {
  transform: scale(1.05);
  opacity: 1;
}
```

The type property.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <button id="eventButton">Click me!</button>
    <p id="eventDisplay">
      The event type will be displayed here and the background will change
      color.
    </p>
    <script src="index.js"></script>
  </body>
</html>
```

index.js

```
const eventDisplay = document.getElementById('eventDisplay');

document.getElementById('eventButton').addEventListener('click', event => {
  // Change the background color of the body
  document.body.style.backgroundColor = 'lightblue';

  // Display the event type in the paragraph
  eventDisplay.textContent = `Event type: ${event.type}`;
});
```

index.css

```
body {
  font-family: 'Arial', sans-serif;
  background-color: #f4f4f4;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  padding: 20px;
```

```
  transition: background-color 0.5s ease;
}

#eventButton {
  padding: 14px 30px;
  font-size: 16px;
  color: #ffffff;
  background-color: #ff9800;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  transition: background-color 0.3s ease, transform 0.2s ease;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
}

#eventButton:hover {
  background-color: #fb8c00;
}

#eventButton:active {
  background-color: #f57c00;
}

#eventDisplay {
  margin-top: 20px;
  font-size: 18px;
  color: #333;
  padding: 12px;
  background-color: #ffffff;
  border-radius: 8px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
  text-align: center;
  width: 350px;
  transition: transform 0.3s ease, opacity 0.3s ease, background-color 0.5s ease;
  opacity: 0.8;
  border-left: 5px solid #ff9800;
}

#eventDisplay.show {
  transform: scale(1.05);
  opacity: 1;
  background-color: #e0f7fa;
  border-left-color: #00796b;
}
```

Example. Method in the Event Object - preventDefault()

index.html

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <a href="https://example.com" id="link">Go to Example</a>
    <script src="index.js"></script>
  </body>
</html>
```

index.js

```
const link = document.getElementById('link');
link.addEventListener('click', event => {
  event.preventDefault(); // Prevent the link from navigating
  alert('Link click prevented!');
});
```

index.html

index.css

```
body {
  display: flex;
  justify-content: center;
  align-items: center;
  background-color: #f4f4f4;
  padding: 20px;
}

#link {
  display: inline-block;
  text-decoration: none;
  font-size: 20px;
  color: #ffffff;
  background-color: #007bff;
  padding: 12px 24px;
  border-radius: 8px;
  transition: background-color 0.3s ease, transform 0.2s ease,
    box-shadow 0.3s ease;
  box-shadow: 0 4px 10px rgba(0, 123, 255, 0.3);
}

#link:hover {
  background-color: #0056b3;
  transform: translateY(-3px);
  box-shadow: 0 6px 15px rgba(0, 86, 179, 0.4);
}

#link:active {
  background-color: #004494;
  transform: translateY(0);
}
```

Here, preventDefault() prevents the default action (navigating to a new page) when clicking the link. Instead, an alert is displayed.

Practical Example: Handling Form Submission and Event Object Usage

Let's validate a form, prevent submission if fields are empty, and display the form's data along with event-related information like the target, type, and how preventDefault() prevents default form submission.

When the "Sign Up" form is submitted, an event listener intercepts the submission with event.preventDefault(). If either the username or password field is empty, an error message is displayed; otherwise, a success message appears, showing the submitted username. Additionally, details about the event—such as type, form ID, and whether the default action was prevented—are displayed in a dedicated section. The form is then reset for further use.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <form id="signup-form">
      <h2>Create an Account</h2>
      <input type="text" id="username" placeholder="Username" />
      <input type="password" id="password" placeholder="Password" />
      <button type="submit">Sign Up</button>
      <p id="error-message" style="display: none">Please fill in all fields</p>
    </form>
    <div id="output-container">
      <p id="output"></p>
      <div id="event-info-container" style="display: none">
        <h3>Event Information</h3>
        <ul id="event-info"></ul>
      </div>
    </div>
    <script src="index.js"></script>
  </body>
</html>
```

index.js

```
const form = document.getElementById('signup-form');
const output = document.getElementById('output');
const errorMessage = document.getElementById('error-message');
const eventInfoContainer = document.getElementById('event-info-container');
const eventInfo = document.getElementById('event-info');

form.addEventListener('submit', event => {
  event.preventDefault(); // Prevent default form submission

  const username = document.getElementById('username').value.trim();
  const password = document.getElementById('password').value.trim();

  // Show error if fields are empty
```

```
  if (!username || !password) {
    errorMessage.style.display = 'block';
  } else {
    errorMessage.style.display = 'none'; // Hide error message
    output.textContent = `Form submitted successfully! Username: ${username}`;

    // Display event-related information
    eventInfoContainer.style.display = 'block';
    eventInfo.innerHTML = `
        <li><strong>Event Type:</strong> ${event.type}</li>
        <li><strong>Form ID:</strong> ${event.target.id}</li>
        <li><strong>Default prevented:</strong> ${event.defaultPrevented}</li>
      `;

    // Clear the form
    form.reset();
  }
});
```

index.css

```
body {
  font-family: 'Arial', sans-serif;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background-color: #f4f7f9;
  min-height: 100vh;
  padding: 20px;
}

form#signup-form {
  background-color: #ffffff;
  padding: 30px;
  border-radius: 12px;
  box-shadow: 0 6px 15px rgba(0, 0, 0, 0.1);
  display: flex;
  flex-direction: column;
  gap: 18px;
  width: 100%;
  max-width: 400px;
  text-align: center;
}

form#signup-form h2 {
  margin-bottom: 20px;
  color: #333;
  font-size: 24px;
  font-weight: bold;
}
```

```css
#signup-form input {
  padding: 14px;
  font-size: 16px;
  border: 2px solid #e2e8f0;
  border-radius: 8px;
  transition: border-color 0.3s ease, box-shadow 0.3s ease;
}

#signup-form input:focus {
  border-color: #4caf50;
  box-shadow: 0 0 10px rgba(76, 175, 80, 0.3);
  outline: none;
}

#signup-form button {
  padding: 14px;
  background-color: #4caf50;
  color: #ffffff;
  font-size: 16px;
  font-weight: bold;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  transition: background-color 0.3s ease, transform 0.2s ease;
}

#signup-form button:hover {
  background-color: #388e3c;
  transform: translateY(-2px);
}

#signup-form button:active {
  background-color: #2e7d32;
  transform: translateY(0);
}

#error-message {
  color: #ff4c4c;
  background-color: #ffe6e6;
  padding: 10px;
  border-radius: 6px;
  text-align: center;
  font-weight: bold;
}

#output-container {
  margin-top: 20px;
  text-align: center;
  max-width: 400px;
  width: 100%;
```

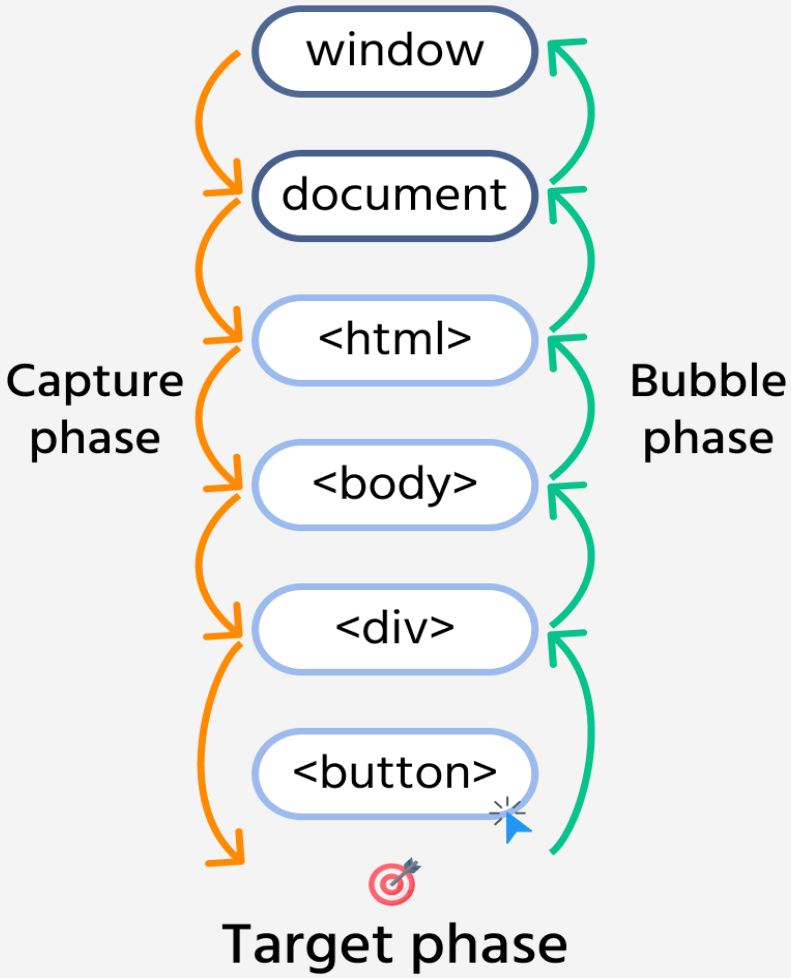
```
}

#output {
  background-color: #e8f5e9;
  color: #2e7d32;
  font-weight: bold;
  padding: 10px;
  border-radius: 8px;
  margin-top: 10px;
}

#event-info-container {
  background-color: #f1f1f1;
  padding: 15px;
  margin-top: 15px;
  border-radius: 8px;
}

#event-info-container h3 {
  color: #333;
  font-size: 18px;
  margin-bottom: 10px;
}

#event-info {
  list-style-type: none;
  padding-left: 0;
  font-size: 16px;
}

#event-info li {
  margin-bottom: 5px;
  color: #555;
}
```

3. Event Propagation and Delegation Explained

https://codefinity.com/courses/v2/3ad37fbc-0d15-4d27-bee7-b107747da548/ef089ecc-cea7-4b55-95ab-780c02d6b6ac/9f742ed8-d969-46f7-87c8-343afb99e8b3

Event Propagation

Event propagation describes how an event moves through the DOM after being triggered, and it has three distinct phases: Capturing, Target, and Bubbling Phases.



Capturing Phase (Capture)

The event starts at the root of the DOM tree (window) and moves down toward the target element. Event listeners in this phase catch the event as it travels downward.

Target Phase

The event reaches the target element (the element that triggered the event). This is where the event listeners attached to the target element itself are executed.

Bubbling Phase (Bubble)

After reaching the target element, the event starts moving back up the DOM tree to the root (window), bubbling through parent elements. This is the most

commonly used phase, allowing parent elements to react to events triggered by child elements.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <div id="outerDiv">
      Outer DIV
      <div id="innerDiv">
        Inner DIV
        <button id="myButton">Click Me</button>
      </div>
    </div>
    <p id="output">Click an element to see how the event bubbles.</p>
    <script src="index.js"></script>
  </body>
</html>
```

index.js

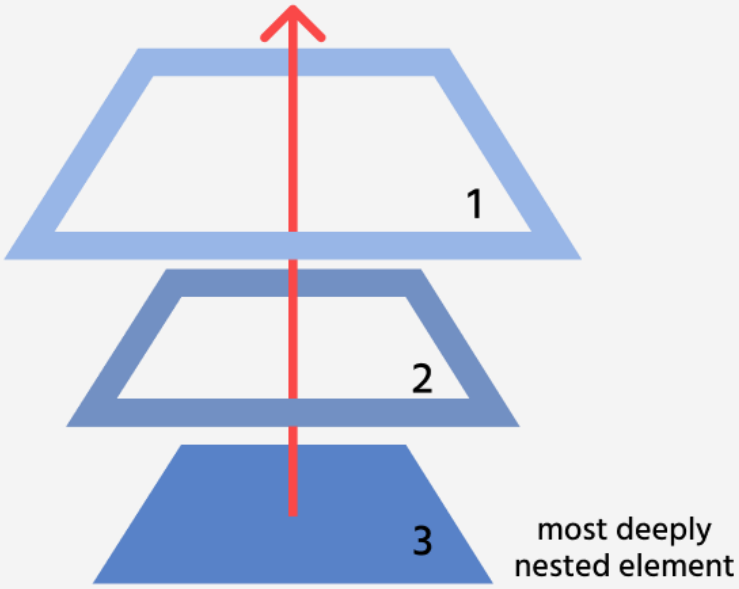
```
const output = document.getElementById('output');
const outerDiv = document.getElementById('outerDiv');
const innerDiv = document.getElementById('innerDiv');
const button = document.getElementById('myButton');

function clearOutput() {
  output.textContent = '';
}

outerDiv.addEventListener('click', () => {
  output.textContent += 'Event caught at Outer DIV\n';
});

innerDiv.addEventListener('click', event => {
  output.textContent += 'Event caught at Inner DIV\n';
});

button.addEventListener('click', event => {
  clearOutput();
  output.textContent += 'Event caught at Button\n';
});
```

index.css

```
body {
  display: flex;
  flex-direction: column;
  justify-content: center;
```

```
  align-items: center;
  background-color: #f4f7f9;
  padding: 20px;
}

#outerDiv {
  padding: 20px;
  border: 2px solid #e74c3c;
  background-color: #f8d7da;
  margin-bottom: 20px;
  border-radius: 10px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
  text-align: center;
  width: 300px;
}

#innerDiv {
  padding: 20px;
  border: 2px solid #28a745;
  background-color: #d4edda;
  border-radius: 10px;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
  text-align: center;
}
#myButton {
  padding: 12px 24px;
  background-color: #007bff;
  color: #ffffff;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  transition: background-color 0.3s ease, transform 0.2s ease;
}
#myButton:hover {
  background-color: #0056b3;
  transform: translateY(-2px);
}

#output {
  margin-top: 20px;
  padding: 15px;
  border: 1px solid #000;
  background-color: #f8f9fa;
  white-space: pre-line;
  border-radius: 6px;
  width: auto;
  min-width: 100px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
  line-height: 1.5;
}
```

The event propagates through the DOM when the button is clicked. First, the event triggers on the button (target phase), then bubbles up to the inner div and finally the outer div (bubbling phase).



Event Delegation

Event Delegation is a technique that leverages event propagation to handle events from child elements using a single event listener on a parent element. Instead of attaching individual listeners to each child, a parent element listens for events that bubble up from its children. This approach has two main advantages:

1. **Performance**: Reducing the number of event listeners improves performance, especially in situations where elements are created dynamically (e.g., a list that grows as new items are added);
2. **Maintainability**: Event delegation simplifies code, especially when working with large DOM structures or dynamic content.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <div id="buttons-container">
      <button>Button 1</button>
      <button>Button 2</button>
      <button>Button 3</button>
    </div>
    <p id="clicked-item-display">Click an item to see the result here.</p>
```

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

Instead of adding click listeners to each `button` element individually, a single listener is added to the `div` parent element. The event bubbles up from the `button` elements to the `div`, where it's handled.

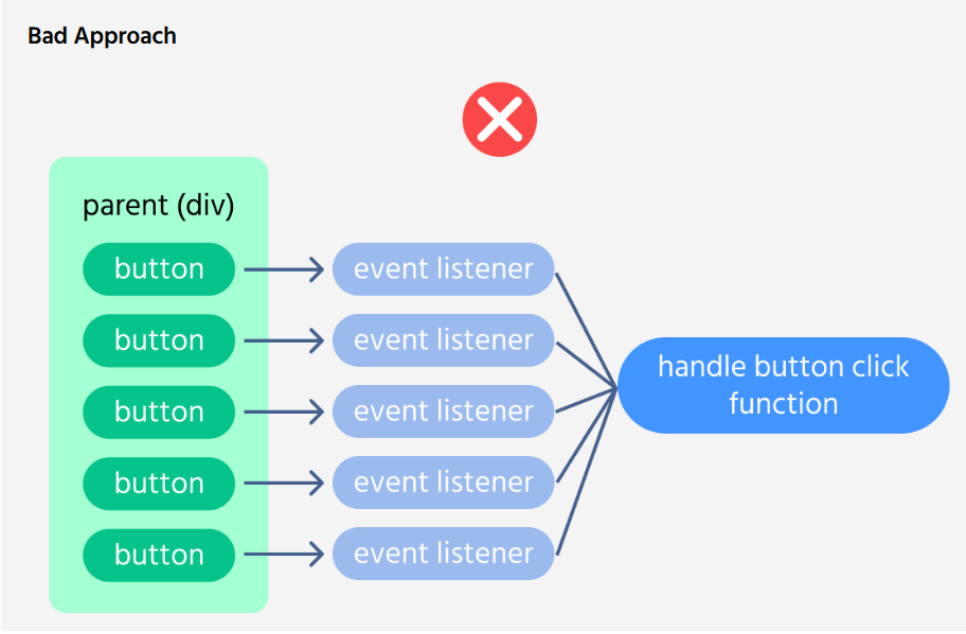
index.js

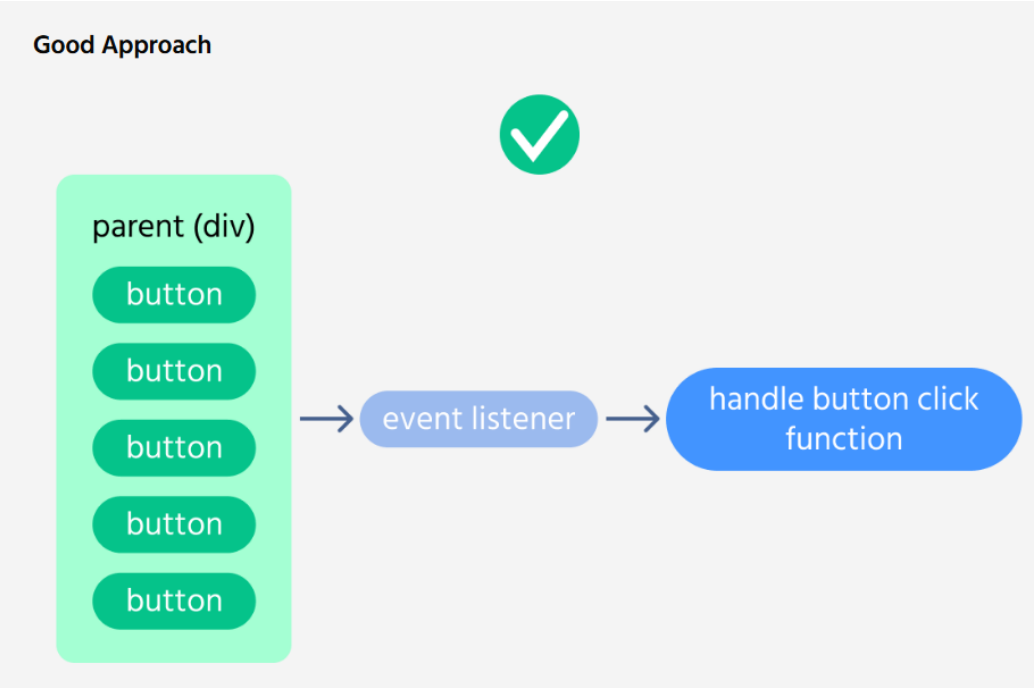
```
const itemList = document.getElementById('buttons-container');
const clickedItemDisplay = document.getElementById('clicked-item-display');

// Event delegation: Listen for clicks on the parent element (DIV)
itemList.addEventListener('click', (event) => {
  if (event.target.tagName === 'BUTTON') {
    clickedItemDisplay.textContent = `You clicked on:
${event.target.textContent}`;
  }
});
```

index.css

```
body {
  margin: 0;
  font-family: Arial, sans-serif;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 100vh;
  background: linear-gradient(120deg, #74ebd5, #acb6e5);
  color: #333;
}

#buttons-container {
  display: flex;
  gap: 20px;
  margin-bottom: 20px;
}

button {
  background: linear-gradient(135deg, #ff7eb3, #ff758c);
  border: none;
  border-radius: 12px;
  padding: 15px 30px;
  font-size: 16px;
  font-weight: bold;
  color: white;
  cursor: pointer;
  transition: transform 0.2s, box-shadow 0.2s, background 0.3s;
}
```

```css
button:hover {
  background: linear-gradient(135deg, #ff6a9f, #ff4d7a);
  box-shadow: 0px 8px 15px rgba(0, 0, 0, 0.2);
  transform: translateY(-3px);
}

button:active {
  transform: translateY(0);
  box-shadow: 0px 5px 10px rgba(0, 0, 0, 0.1);
}

#clicked-item-display {
  font-size: 18px;
  font-weight: bold;
  padding: 10px 20px;
  background: rgba(255, 255, 255, 0.8);
  border-radius: 10px;
  box-shadow: 0px 4px 10px rgba(0, 0, 0, 0.2);
  transition: background 0.3s;
}

#clicked-item-display:hover {
  background: rgba(255, 255, 255, 0.9);
}
```





Practical Example: Handling Dynamic List

Event delegation is perfect for managing interactions in lists or tables that may grow over time (e.g., adding tasks to a to-do list, or products to a shopping cart). As items are added or removed, the parent container (like `ul` or `table`) handles all interactions, saving the need to attach or remove listeners to each child element.

index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <ul id="todo-list">
      <li>Task 1</li>
      <li>Task 2</li>
    </ul>
    <button id="add-task">Add Task</button>
    <script src="index.js"></script>
  </body>
</html>
```

index.js

```js
const todoList = document.getElementById('todo-list');
const addTaskButton = document.getElementById('add-task');
let taskCount = 2;
```

```
// Event delegation to handle clicks on any task
todoList.addEventListener('click', event => {
  if (event.target.tagName === 'LI') {
    // Displaying an alert when a task is clicked
    alert(`You clicked on: ${event.target.textContent}`);

    // Stopping the event from bubbling further up the DOM tree
    event.stopPropagation(); // Prevent event propagation beyond the clicked task
  }
});

// Dynamically adding tasks
addTaskButton.addEventListener('click', event => {
  taskCount++;
  const newTask = document.createElement('li');
  newTask.textContent = `Task ${taskCount}`;
  todoList.appendChild(newTask); // Event delegation still applies to new tasks

  // Prevent further propagation of the event (optional)
  event.stopPropagation();
});

// You can remove the body event listener to avoid console usage entirely
document.body.addEventListener('click', () => {
  alert('Clicked outside the todo list.');
});
```

index.css

```
body {
  background-color: #f4f4f9;
  color: #333;
  padding: 20px;
  display: flex;
  flex-direction: column;
  align-items: center;
}

#todo-list {
  list-style-type: none;
  padding: 0;
  width: 100%;
  max-width: 400px;
}

#todo-list li {
  background-color: #fff;
  border: 1px solid #ccc;
  padding: 12px 20px;
  margin: 8px 0;
```

```
  border-radius: 8px;
  cursor: pointer;
  transition: background-color 0.3s, transform 0.2s, box-shadow 0.3s;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
}

#todo-list li:hover {
  background-color: #e0f7fa;
  transform: translateY(-2px);
  box-shadow: 0 6px 12px rgba(0, 123, 255, 0.1);
}

button {
  background-color: #007bff;
  color: white;
  border: none;
  padding: 12px 24px;
  border-radius: 8px;
  cursor: pointer;
  font-size: 16px;
  margin-top: 20px;
  transition: background-color 0.3s, transform 0.2s, box-shadow 0.3s;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
}

button:hover {
  background-color: #0056b3;
  transform: translateY(-2px);
  box-shadow: 0 6px 15px rgba(0, 86, 179, 0.3);
}

button:active {
  background-color: #004080;
  transform: translateY(0);
}
```

4 Working with Mouse Events

Handling Mouse Events

Mouse events are triggered when users interact with their mouse or a touchpad. The most commonly used mouse events are:

Let's create an example where a tooltip is displayed when the mouse hovers over a button.

index.html

```
<!DOCTYPE html>
```

```html
<html>
  <head>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <button id="infoButton">Hover over me</button>
    <div id="tooltip">This is a helpful tooltip!</div>
    <script src="index.js"></script>
  </body>
</html>
```

index.js

```javascript
const button = document.getElementById('infoButton');
const tooltip = document.getElementById('tooltip');

button.addEventListener('mouseover', event => {
  tooltip.style.display = 'block'; // Show the tooltip
  tooltip.style.left = `${event.pageX + 10}px`;
  tooltip.style.top = `${event.pageY + 10}px`;
  tooltip.classList.add('show'); // Add class to enable the fade-in effect
});

button.addEventListener('mousemove', event => {
  tooltip.style.left = `${event.pageX + 10}px`;
  tooltip.style.top = `${event.pageY + 10}px`;
});

button.addEventListener('mouseout', () => {
  tooltip.style.display = 'none'; // Hide the tooltip
  tooltip.classList.remove('show'); // Remove class to reset
});
```

index.css

```css
body {
  font-family: 'Arial', sans-serif;
  background-color: #f4f4f9;
  padding: 40px;
  display: flex;
  justify-content: center;
  align-items: center;
}

button#infoButton {
  background-color: #007bff;
  color: white;
  padding: 12px 24px;
  border: none;
  border-radius: 8px;
```

```
  cursor: pointer;
  font-size: 16px;
  transition: background-color 0.3s, transform 0.2s ease;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
}

button#infoButton:hover {
  background-color: #0056b3;
  transform: translateY(-2px);
}

button#infoButton:active {
  background-color: #003f8c;
}

#tooltip {
  display: none;
  position: absolute;
  background-color: rgba(0, 0, 0, 0.8);
  color: white;
  padding: 8px 12px;
  border-radius: 6px;
  font-size: 14px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
  opacity: 0;
  transition: opacity 0.3s ease;
  pointer-events: none;
}

#tooltip.show {
  opacity: 1;
}
```

Offering quick, contextual information without cluttering the interface is a common use case, improving usability and guiding users through complex interfaces.