

Web Programming 2x1

# JavaScript

## DOM Events

# Outcomes

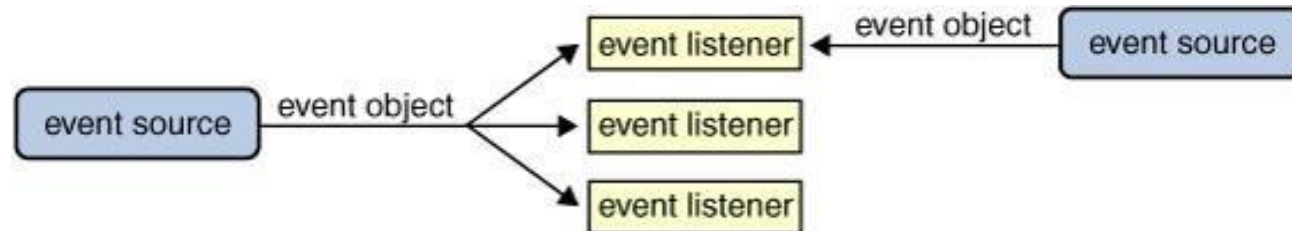
---

Students should understand the following outcomes, upon successful completion of this module:

- JavaScript Events
- Inline events
  - Mouse Events
  - Keyboard Events
  - Form Events
  - Window Events
  - Touch Events
- Event Listeners

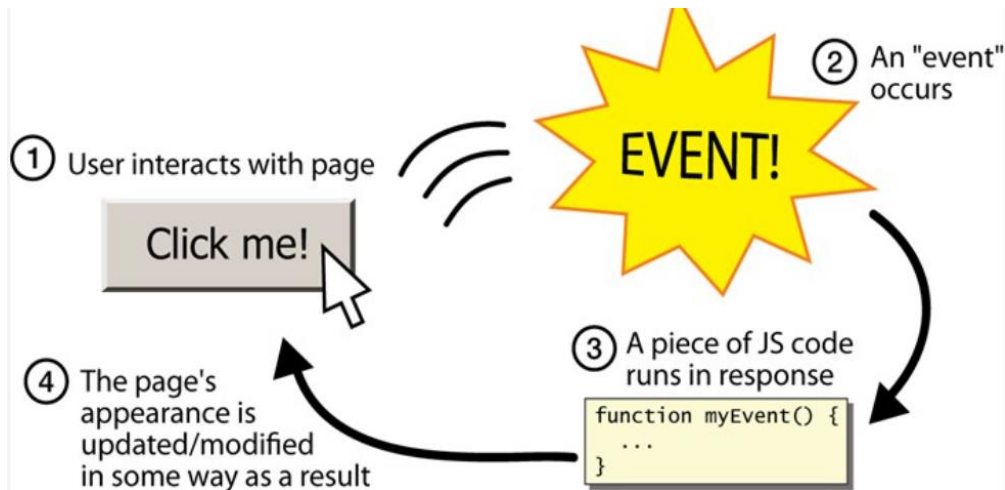
# Events in JavaScript

- In JavaScript, events are interactions or occurrences that happen on the web page, often initiated by the user or the browser itself.
- These events are triggered by actions like clicking a button, pressing a key, moving the mouse, submitting a form, resizing the window, and much more.
- Handling these events allows developers to create dynamic and interactive web applications.
- JavaScript provides the ability to detect and respond to these events, allowing you to create interactive and dynamic web applications.
- Events are a fundamental part of web development because they enable you to build responsive user interfaces and create interactions that go beyond static content.



# JavaScript Events

- DOM events are signals exposed by the browser that you can use to run a piece of JavaScript code.
- DOM Events allow JavaScript to add *event listeners* or *event handlers* to HTML elements.
- When a specific DOM event happens in the browser, a piece of code will be executed as a response to that action.



- These DOM events occur when the user interacts with the application we've created, such as clicking a button or typing letters into an input field.
- As a web developer, you can instruct JavaScript to listen for a specific event and do something in response to that event.

# Mouse Events

---

- Mouse events allow you to handle user interactions involving the mouse, such as clicking, moving, or hovering over elements.
  - **onclick**: Fires when the element is clicked.
  - **dblclick**: Fires when the element is double-clicked.
  - **mouseover**: Fires when the mouse pointer is moved over the element.
  - **mouseout**: Fires when the mouse pointer leaves the element.
  - **mousemove**: Fires when the mouse pointer moves within the element.
  - **mousedown**: Fires when a mouse button is pressed down.
  - **mouseup**: Fires when a mouse button is released.

# Mouse Events

- Examples:

```
<body>

<button onclick="alert('Button clicked!')">Click Me</button>

<div ondblclick="alert('Div double-clicked!')" style="width: 60px; height: 60px; border: 1px solid black;">
| Double-click me
</div> <br>

<div onmouseover="alert('Mouse is over the div!')" style="width: 60px; height: 60px; border: 1px solid black;">
| Hover over me
</div> <br>

<div onmouseout="alert('Mouse left the div!')" style="width: 60px; height: 60px; border: 1px solid black;">
| Hover over me
</div> <br>

<div onmousemove="alert('Mouse is moving over the div!')" style="width: 60px; height: 60px; border: 1px solid black;">
| Move mouse over me
</div> <br>

</body>
```

# Keyboard Events

---

- Keyboard events in web development are essential for creating interactive web applications.
- They allow you to capture and respond to user interactions with the keyboard, such as key presses and releases.
- In JavaScript, there are three primary keyboard events:
  - **keydown**: Triggered when a key is pressed down.
  - **keyup**: Triggered when a key is released.
  - **keypress**: Triggered when a key is pressed and released

# Keyboard Events

- Example:

```
<body>

  <input type="text" placeholder="Type something..." onkeydown="alert('Key Down: ' + event.key)">

  <input type="text" placeholder="Type something..." onkeypress="alert('Key Press: ' + event.key)" >

  <input type="text" placeholder="Type something..." onkeyup="alert('Key Up: ' + event.key)">

</body>
```

- The input element has inline event handlers for keydown, keypress, and keyup events.
- Each event handler uses an inline alert to display the event type and the key that was pressed or released.



# Form Events

---

- Form events in web development are events that occur when a user interacts with form elements like input fields, select boxes, and text areas.
- Examples of form events include:
  - **submit**: Triggered when a form is submitted.
  - **change**: Triggered when the value of an input, select, or textarea element is changed.
  - **focus**: Triggered when an element gains focus.
  - **blur**: Triggered when an element loses focus.

# Form Events

- Examples:

```
<form onsubmit="alert('Form Submitted'); return false;">

  <input type="text" placeholder="Type something..." onfocus="alert('Input Focused'); this.blur();" >
  <input type="text" placeholder="Type something..." onblur="alert('Input Blurred')">

  <br><br>
  <select onchange="alert('Selection Changed: ' + this.value)">
    <option value="Option 1">Option 1</option>
    <option value="Option 2">Option 2</option>
    <option value="Option 3">Option 3</option>
  </select>

  <br><br>
  <button type="submit">Submit</button>

</form>
```

# JavaScript Inline Events

- Inline events are JavaScript code directly embedded within HTML tags (So far we have utilized inline events).
- These are the simplest form of handling events, often used for quick and small scripts

```
<button onclick="alert('Button clicked!')">Click Me</button>
```

## ADVANTAGES

- **Simplicity:** Easy to implement for small, simple tasks.
- **Quick Setup:** Can be added directly within the HTML, making it quick for prototyping.

## DISADVANTAGES

- **Maintenance:** Harder to maintain and debug, especially as the complexity grows.
- **Separation of Concerns:** Violates the principle of separating HTML, CSS, and JavaScript.
- **Scalability:** Not suitable for larger applications where more complex event handling is required.

# Window Events

---

- Window events in web development are events that are associated with the browser window or the document.
- These events help you handle actions related to the loading and closing of the window, resizing the window, scrolling, and other interactions.
  - **load**: Triggered when the entire page, including all dependent resources, has loaded.
  - **resize**: Triggered when the window is resized.
  - **scroll**: Triggered when the window is scrolled.

# Window Events

- Examples:

```
<html lang="en">
<head>
  <title>Window Event Example</title>
  <style>
    |   body {
    |   |   height: 2000px; /* Make the page scrollable */
    |   }
  </style>
  <script>
    |   function showAlertOnResize() {
    |   |   alert('Window has been resized!');
    |   }
    |
    |   function showAlertOnScroll() {
    |   |   alert('Window has been scrolled!');
    |   }
    |
    |   window.onload = function() {
    |   |   alert('Page has fully loaded!');
    |   };
    |
    |   window.onresize = showAlertOnResize;
    |   window.onscroll = showAlertOnScroll;
  </script>
</head>
<body>
  <h1>Window Event Example</h1>
  <p>Resize the browser window or scroll down the page to see alerts.</p>
</body>
</html>
```

# Event Handlers

- Event handlers are JavaScript functions assigned to HTML elements using properties.
- These handlers are defined within the JavaScript code and assigned to the elements, providing a clearer separation between HTML and JavaScript.
- *Example 1*: use *Embedded* JavaScript to define *event handler* function:

```
<body>
  <h1>Embedded JavaScript Event Handler</h1>
  <button onclick="showAlert()">Click Me!</button>

  <script>
    function showAlert() {
      alert('Button clicked!');
    }
  </script>
</body>
```

# Event Handlers

- *Example 2*: use *External* JavaScript to define *event handler* function:

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>External JS Event Handler</title>
  <script src="script.js" defer></script>
</head>
<body>
  <h1>External JavaScript Event Handler</h1>
  <button onclick="showAlert()">Click Me!</button>
</body>
```

```
function showAlert() {
    alert('Button clicked!');
}
```

# Event Handlers

---

## ADVANTAGES

**Clearer Separation:** Keeps HTML and JavaScript code separate, improving readability.

**Reusability:** Functions can be reused and reassigned

## DISADVANTAGES

- **Overwriting:** Assigning a new handler will overwrite the existing one, which can be limiting.



# The `defer` Attribute

---

- The defer attribute is used in HTML `<script>` tags to control the loading and execution of external JavaScript files.

## Purpose of defer:

- *Defers Execution*: When you include the defer attribute in a `<script>` tag, it tells the browser to delay the execution of the script until after the HTML document has been completely parsed. This ensures that the script runs only after the DOM is fully constructed.
- *Maintains Script Order*: If you have multiple `<script>` tags with the defer attribute, they will be executed in the order in which they appear in the document.
- *Non-blocking*: The defer attribute allows the browser to continue parsing the HTML document while the script is being downloaded. This avoids blocking the HTML rendering process, which can improve page load performance.

# Event Listeners

---

- Event listeners offer a more flexible way to handle events by using JavaScript to add event listeners to elements.
- This method keeps the HTML clean and separates the JavaScript code.
- We use an *addEventListener()* method, which can also allow multiple event handlers for the same event, without overwriting each other.

Syntax:

```
element.addEventListener(eventType, callback);
```

*element*: The HTML element to which you want to attach the event listener.

*eventType*: A string that specifies the type of event you want to listen for, such as "click", "keydown", "mouseover", etc.

*callback*: The function that will be executed when the specified event occurs.

# Event Listeners

---

## ADVANTAGES

**Multiple Handlers:** Allows multiple event handlers for the same event.

**Advanced Features:** Supports capturing, bubbling, and options for fine-tuning the event flow.

**Cleaner Code:** Encourages better code organization and separation of concerns.

## DISADVANTAGES

**Complexity:** Slightly more complex syntax compared to inline events and event handlers.

# Event Listeners

---

You have a form with multiple buttons performing different actions. Using event listeners, you can easily manage these actions without cluttering your HTML or risking overwriting handlers.

HTML CODE:

```
<form id="myForm">  
  <input type="text" id="textInput">  
  <button type="button" id="submitButton">Submit</button>  
  <button type="button" id="resetButton">Reset</button>  
</form>
```

# Example Continued

## JavaScript CODE

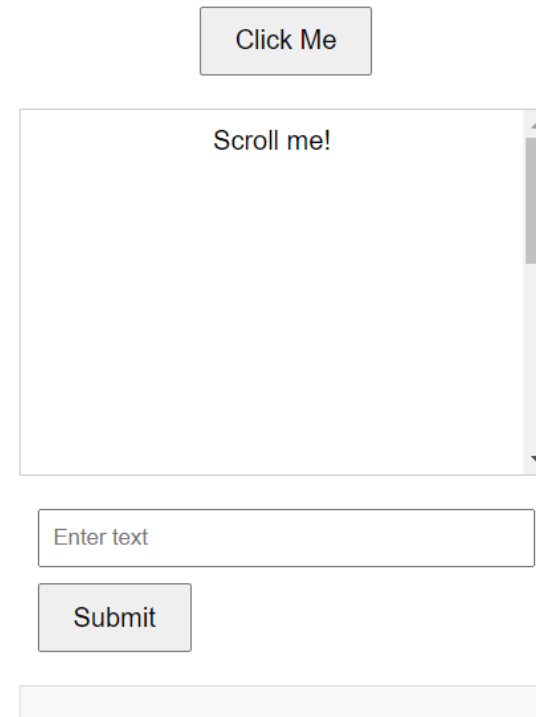
```
document.getElementById("submitButton").addEventListener("click", function() {  
    alert('Form submitted!');  
});  
  
document.getElementById("resetButton").addEventListener("click", function() {  
    document.getElementById("myForm").reset();  
    alert('Form reset!');  
});  
  
document.getElementById("textInput").addEventListener("input", function() {  
    console.log('Input changed to: ' + this.value);  
});
```

# addEventListener

## Example 2:

- Lets create the following in HTML page:
- Use the CSS file in addendum.

## Event Listeners Demo



The demo UI consists of several elements: a 'Click Me' button at the top; a large text area with the text 'Scroll me!' and a vertical scrollbar; a text input field with the placeholder 'Enter text'; a 'Submit' button below the input field; and a horizontal scrollbar at the bottom.

- Create and utilise event listeners for *click*, *mouseover*, *submit* and *resize* events.

# addEventListener() method

Solution:

```
<body>

  <h1>Event Listeners Demo</h1>

  <button id="clickButton">Click Me</button>

  <div id="scrollDiv">
    Scroll me!
    <div style="height: 500px;"></div>
  </div>

  <form id="myForm">
    <input type="text" placeholder="Enter text" id="textInput">
    <button type="submit">Submit</button>
  </form>

  <div id="output"></div>

  <script src="app.js"></script>

</body>
```

Index.html

# addEventListener() method

Solution:

```
// Function to handle mouseover event
function handleMouseOver() {
  // Change background color on mouse over
  document.body.style.backgroundColor = 'blue';
}

// Function to handle button click
function handleClick() {
  alert('Button clicked!');
}

// Function to handle form submission
function handleFormSubmit(event) {
  event.preventDefault(); // Prevent default form submission
  const inputText = document.getElementById('textInput').value;
  const outputDiv = document.getElementById('output');
  outputDiv.textContent = `Form submitted with input: ${inputText}`;
  console.log('Form submitted!');
}

// Function to handle window resize
function handleWindowResize() {
  alert(`Window resized to ${window.innerWidth}x${window.innerHeight}`);
}
```

app.js

```
// Adding event listeners
document.getElementById('clickButton').addEventListener('click', handleClick);
document.getElementById('scrollDiv').addEventListener('mouseover', handleMouseOver);
document.getElementById('myForm').addEventListener('submit', handleFormSubmit);
window.addEventListener('resize', handleWindowResize);
```



# Removing event listeners

- If we need to remove the event listeners, the most common way is using the `removeEventListener`.
- It requires to follow strict parameters:
  - ✓ the type of listener to remove,
  - ✓ the same *callback* function for that listener

```
document.querySelector('#btn').addEventListener('click', () => {  
  console.log('clicked');  
});  
  
document.querySelector('#btn').removeEventListener('click', () => {  
  console.log('clicked');  
});
```

# Key Differences:

---

Aspect	Inline HTML Event Handling	JavaScript Event Handling
Simplicity	Simple for small tasks	Slightly more complex initially
Separation of Concerns	Poor (mixes HTML and JavaScript)	Good (separates HTML and JavaScript)
Reusability	Limited (harder to reuse and maintain)	High (easier to reuse and maintain)
Scalability	Poor (becomes hard to manage as project grows)	Good (scales well with project size)
Flexibility	Limited (hard to add/remove dynamically)	High (easy to add/remove listeners dynamically)
Advanced Features	Limited (no support for capturing/bubbling phases)	Full support (can handle advanced event features)

# Exercise

- Create a To-Do List web application that functions as shown below. A user should add tasks, mark them as complete or delete a task once complete. Use event listeners to achieve this. Addendum provided for HTML and CSS.

## To-Do List

Add Task

## To-Do List

Add Task

~~Clean room~~

Complete

Delete

~~Study for test~~

Complete

Delete

Cook pap

Complete

Delete

## To-Do List

Add Task

~~Clean room~~

Complete

Delete

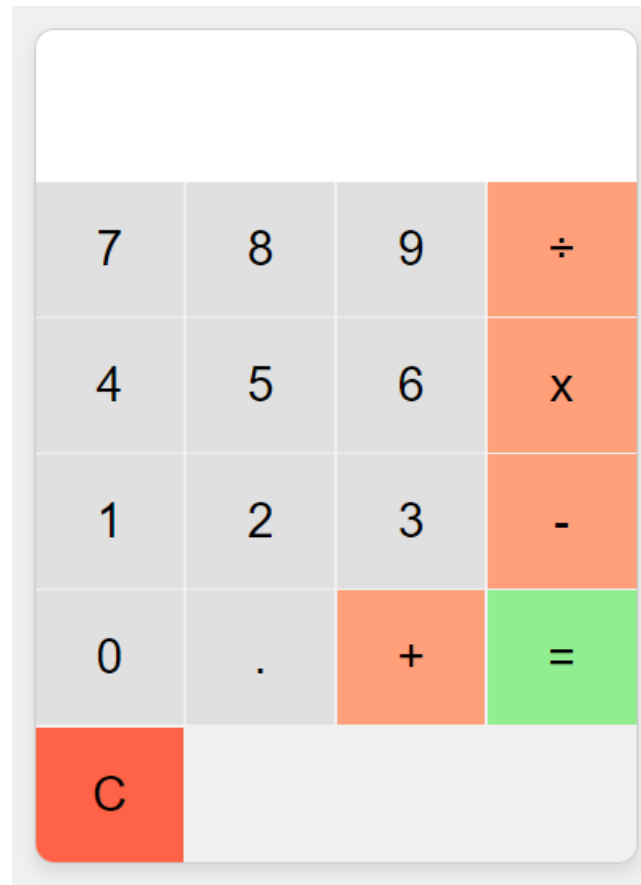
Cook pap

Complete

Delete

# Exercise

- Create a Calculator web application that functions as shown below. A user should be able to add, divide, multiply and subtract values. C should clear the display. Use event listeners to achieve this. Addendum provided for HTML and CSS.



Thank You!

*THE END*

[info@belgiumcampus.ac.za](mailto:info@belgiumcampus.ac.za)

+27 10 593 53 68



/belgiumcampusSA



#Belgium Campus



/belgiumcampus

# References

---

<https://www.freecodecamp.org/news/here-are-the-new-built-in-methods-and-functions-in-javascript-8f4d2fd794fa/>

<https://javascript.info/array-methods>

<https://www.tutorialrepublic.com/javascript-examples.php>

<https://ultimatecourses.com/blog/array-reduce-javascript>