# Task Js(2)


**Name:** Matthew Kamal Saad

**Submitted to:** Eng. Ahmed Saber

**Date:** 22/11/2025

## INLINE JAVASCRIPT

**Inline JavaScript** involves embedding JavaScript code directly within HTML element attributes. This method is ideal for simple, small scripts or event handlers.

```
<!DOCTYPE html>
<html>
<head>
  <title>Inline JavaScript Example</title>
</head>
<body>
  <!-- The onclick attribute contains inline JavaScript -->
  <button onclick="alert('Hello from inline JavaScript!');">Click Me</button>
</body>
</html>
```

### When to Use Inline JavaScript

- Perfect for small scripts or single event handlers.
- Useful during the initial stages of development when testing functionality.

### Drawbacks

- It can become messy and hard to manage as your project grows.
- Mixes HTML structure with JavaScript behavior, making the codebase less organized.

## INTERNAL JAVASCRIPT

**Internal JavaScript** is placed within the HTML document inside <script> tags. This method keeps JavaScript code separate from HTML element attributes while still residing in the same file.

```
<!DOCTYPE html>
<html>
<head>
  <title>Internal JavaScript Example</title>
  <script>
    // This script is placed in the head of the document
    function greet() {
      alert('Hello from internal JavaScript!');
    }
  </script>
</head>
<body>
  <button onclick="greet();">Click Me</button>
</body>
</html>
```

### Key Characteristics

- Keeps JavaScript separate from HTML attributes, promoting cleaner code.
- All code resides in one HTML file, simplifying deployment for smaller projects.

### Considerations

- As the project grows, having all scripts in one file can become unwieldy.
- Mixing content and behavior in a single file might reduce organization for larger applications.

## EXTERNAL JAVASCRIPT

**External JavaScript** involves placing JavaScript code in separate .js files, which are then linked to the HTML document using the <script> tag with a src attribute. This method is widely preferred for its maintainability, reusability, and clear separation of concerns.

### HTML File (index.html):

```
<!DOCTYPE html>
<html>
<head>
  <title>External JavaScript Example</title>
  <!-- Link to the external JS file -->
  <script src="script.js"></script>
</head>
<body>
  <button onclick="sayHello();">Click Me</button>
</body>
</html>
```

### External JavaScript File (script.js):

```
function sayHello() {
  alert('Hello from external JavaScript!');
}
```

### Advantages of External JavaScript

- Clearly separates HTML structure from JavaScript behavior, making the codebase more organized.
- Allows the same JavaScript file to be used across multiple HTML pages, reducing redundancy.

## Difference between addEventListener and onClick

- **onClick**

  The. onclick property directly assigns a single function to handle the click event of an element.

  **How It Works:**

1. The DOMContentLoaded event ensures the DOM is fully loaded before executing the script.
2. The .onclick property assigns a single function to handle the click event on the element with the ID getMessage.
3. When the button is clicked, the function sends an HTTP request, parses the response, and updates the element with the class message.

- **addEventListener**

  The addEventListener method allows you to attach multiple event listeners to the same element for the same event.

  **How It Works:**

1. Similar to the .onclick method, DOMContentLoaded ensures the DOM is loaded before the script runs.
2. addEventListener attaches a click event listener to the button.
3. The same actions (sending an HTTP request, parsing the response, and updating the message element) occur when the button is clicked.