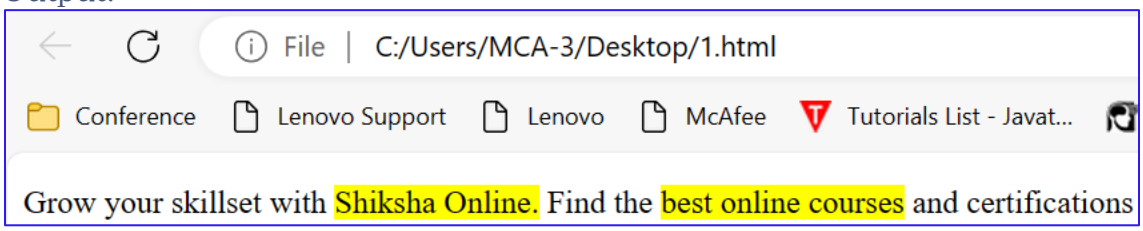
The <mark> element highlights a particular text of special interest to the user in an HTML document.

Example of the mark element:

```
<!DOCTYPE html>
<html>
      <body>
        <p> Grow your skillset with <mark>Shiksha Online.</mark> Find the <mark>best online courses</mark> and certifications in management, technology, programming, and data science.</p>
      </body>
</html>
```

Output:



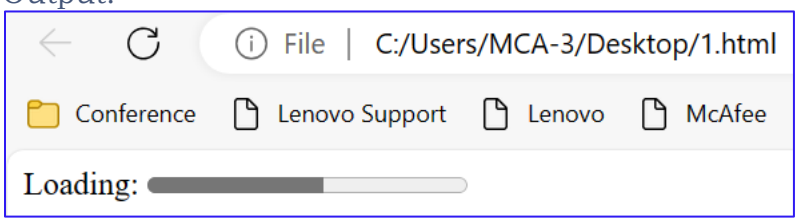
7. Progress Tag

The progress tag helps users check the progress of a task during the execution. Users will need to define the dynamically changing values of the progress bar with the scripts (JavaScript).

Example of progress tag in HTML5:

```
<!DOCTYPE html>
<html>
  <body>
    <span>Loading:</span>
    <progress value="55" max="100"></progress>
  </body>
</html>
```

Output:



8. Geolocation API

The Geolocation API is a standard web API that provides a way for web applications to request the user's location information.

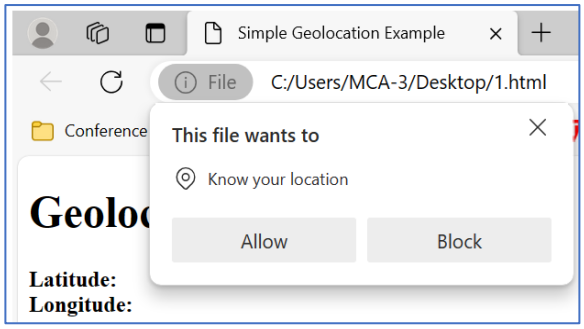
Modern web browsers support the Geolocation API, allowing websites to request and use the user's location data. JavaScript functions, such as navigator.geolocation.getCurrentPosition, facilitate this process.

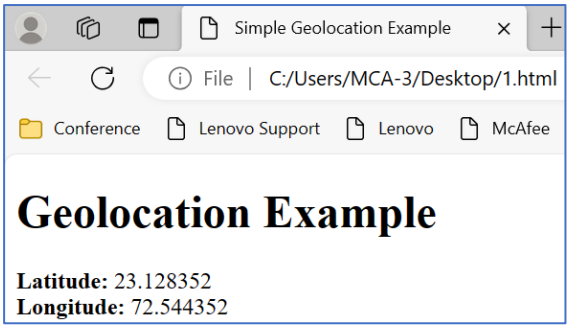
This example will display the latitude and longitude of the user's location on a web page.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Geolocation Example</title>
    <style>
        #location-info {
            font-size: 18px;
            margin-top: 20px;
        }
    </style>
</head>
<body>

<h1>Geolocation Example</h1>

<div id="location-info">
    <strong>Latitude:</strong> <span id="latitude"></span><br>
    <strong>Longitude:</strong> <span id="longitude"></span>
</div>

<script>
    // Check if the browser supports the Geolocation API
    if (navigator.geolocation) {
        // Request the user's current location
        navigator.geolocation.getCurrentPosition(showPosition, showError);
    } else {
        // Display an error message if the Geolocation API is not supported
        showError("Geolocation is not supported by this browser.");
    }

    // Callback function to handle the successful retrieval of the user's location
    function showPosition(position) {
        // Extract latitude and longitude from the position object
        var latitude = position.coords.latitude;
        var longitude = position.coords.longitude;

        // Display the latitude and longitude on the web page
        document.getElementById("latitude").textContent = latitude.toFixed(6);
        document.getElementById("longitude").textContent = longitude.toFixed(6);
    }
```

```
    // Callback function to handle errors during the retrieval of the user's location
    function showError(error) {
        var errorMessage;

        switch (error.code) {
            case error.PERMISSION_DENIED:
                errorMessage = "User denied the request for Geolocation.";
                break;
            case error.POSITION_UNAVAILABLE:
                errorMessage = "Location information is unavailable.";
                break;
            case error.TIMEOUT:
                errorMessage = "The request to get user location timed out.";
                break;
            case error.UNKNOWN_ERROR:
                errorMessage = "An unknown error occurred.";
                break;
            default:
                errorMessage = "An unspecified error occurred.";
                break;
        }

        // Display the error message on the web page
        alert(errorMessage);
    }
</script>

</body>
</html>
```

- The script checks if the browser supports Geolocation using **navigator.geolocation**.
- If supported, it calls **navigator.geolocation.getCurrentPosition** to request the user's current location, passing two callback functions: **successCallback** for successful retrieval and errorCallback for handling errors.
- If Geolocation is not supported, an alert is displayed notifying the user.
- The **successCallback** function extracts the latitude and longitude from the position object and updates the corresponding spans in the HTML to display these values.
- The **errorCallback** function handles different types of errors that might occur during the Geolocation process and displays an alert with an appropriate message.





9. Local Storage

It is a modern feature of HTML and several browsers that typically store data in the user's browsers and can access them with the help of JavaScript APIs. This feature is useful for creating offline applications where data is need to be stored locally.

Moreover, using this feature you can reduce the transactions between the application and the backend server, creating a fast application. However, there are some limitations, such as there is limited storage, and you can not access more storage than that.

Local Storage is a simple key-value storage mechanism provided by web browsers to store data persistently on a user's device. Here's a basic example using JavaScript to interact with the Local Storage:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Local Storage Example</title>
</head>
<body>

<h1>Local Storage Example</h1>

<button onclick="saveData()">Save Data</button>
<button onclick="retrieveData()">Retrieve Data</button>

<script>
    // Function to save data to Local Storage
    function saveData() {
        // Get data from user (for simplicity, we're using a prompt here)
        var userInput = prompt("Enter some data:");

        // Check if the user entered something
        if (userInput) {
            // Save data to Local Storage with a specific key
```

```
            localStorage.setItem("userData", userInput);
            alert("Data saved to Local Storage!");
        } else {
            alert("Please enter some data.");
        }
    }

    // Function to retrieve data from Local Storage
    function retrieveData() {
        // Retrieve data from Local Storage using the key
        var storedData = localStorage.getItem("userData");

        // Check if there is any stored data
        if (storedData) {
            alert("Data retrieved from Local Storage: " + storedData);
        } else {
            alert("No data found in Local Storage.");
        }
    }
</script>

</body>
</html>
```

saveData Function:

This function is executed when the "Save Data" button is clicked.

It uses the prompt function to get input from the user (data to be saved).

If the user enters data, it uses localStorage.setItem to save the data with the key "userData" to Local Storage.

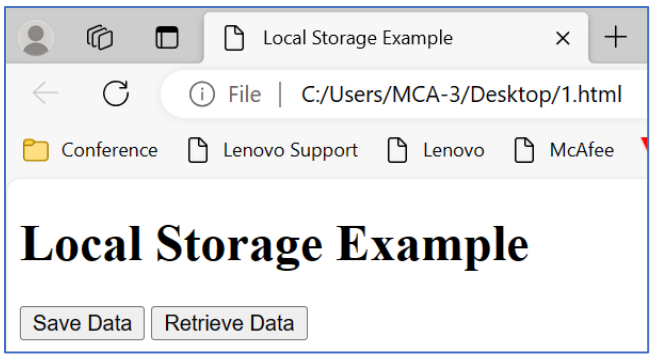
An alert is displayed indicating that the data has been saved, or an alert prompts the user to enter data if nothing is entered.

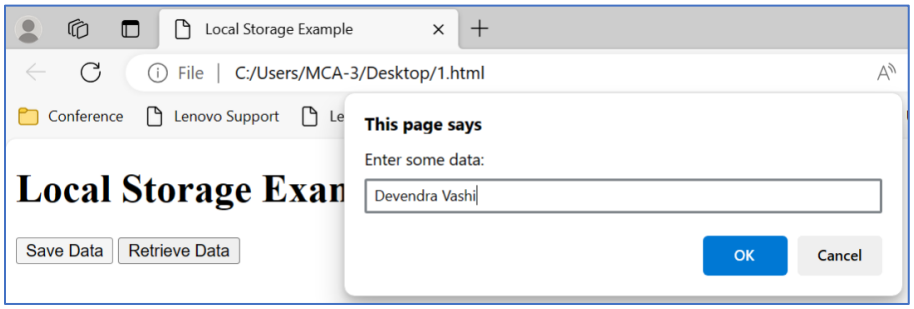
retrieveData Function:

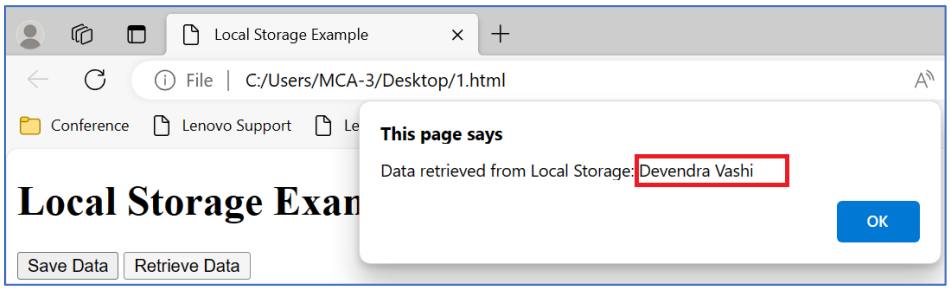
This function is executed when the "Retrieve Data" button is clicked.

It uses localStorage.getItem to retrieve the previously saved data from Local Storage using the key "userData."

If data is found, an alert is displayed with the retrieved data. If no data is found, an alert indicates that no data was found.



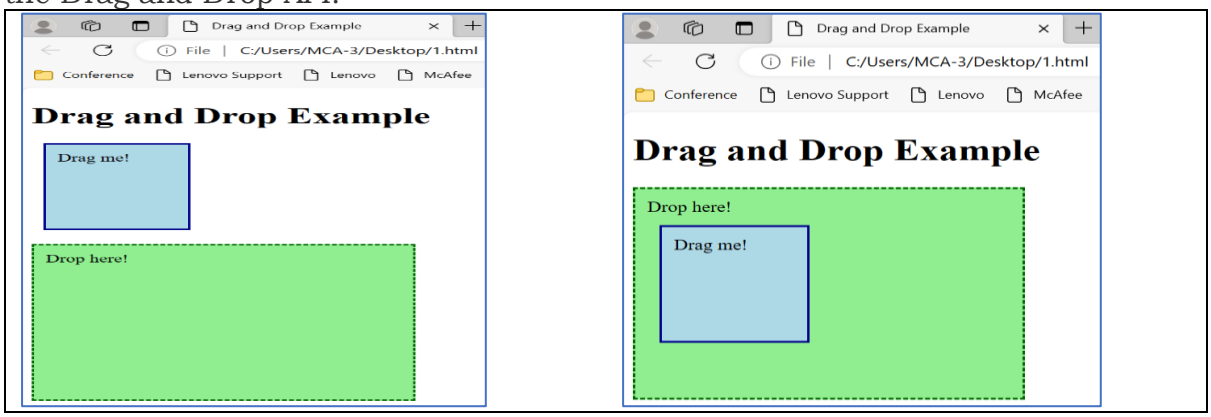




10. Drag and Drop API

Drag and drop is among the most unique features of HTML5 that allow you to grab any element in the DOM and drop it to a different location. To create an element able to drag and drop, set the attribute "**draggable**" on the tag and put its value to true. Let's understand this process using a practical coding example.

The Drag and Drop API in HTML allows you to make elements on a web page draggable and define drop zones for them. Here's a simple example demonstrating the Drag and Drop API:




```
<!DOCTYPE html>
```

```html
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Drag and Drop Example</title>
    <style>
        #dragElement {
            width: 100px;
            height: 100px;
            background-color: lightblue;
            border: 2px solid darkblue;
            margin: 10px;
            padding: 10px;
            cursor: move;
        }

        #dropZone {
            width: 300px;
            height: 200px;
            background-color: lightgreen;
            border: 2px dashed darkgreen;
            margin-top: 20px;
            padding: 10px;
        }
    </style>
</head>
<body>

<h1>Drag and Drop Example</h1>

<div id="dragElement" draggable="true" ondragstart="dragStart(event)">
    Drag me!
</div>

<div id="dropZone" ondragover="allowDrop(event)" ondrop="drop(event)">
    Drop here!
</div>

<script>
    // Function to handle the drag start event
    function dragStart(event) {
        // Set data to be transferred during drag
```

```
        event.dataTransfer.setData("text/plain", event.target.id);
    }

    // Function to allow dropping on the drop zone
    function allowDrop(event) {
        event.preventDefault();
    }

    // Function to handle the drop event
    function drop(event) {
        event.preventDefault();

        // Get the dragged element's id from the data transfer
        var draggedElementId = event.dataTransfer.getData("text/plain");

        // Get the dragged element by id
        var draggedElement = document.getElementById(draggedElementId);

        // Append the dragged element to the drop zone
        document.getElementById("dropZone").appendChild(draggedElement);
    }
</script>

</body>
</html>
```

When you run this code, you'll see a draggable element ("Drag me!") that can be dragged and dropped into the specified drop zone ("Drop here!"). The JavaScript functions handle the drag-and-drop events, allowing for interactive behavior on the webpage.

- The HTML document includes the necessary meta tags and a title.
- Inside the **<style>** tag, there are CSS styles for the draggable element (**#dragElement**) and the drop zone (**#dropZone**).

- The "dragElement" div is created with an id, and the **draggable** attribute is set to "true" to make it draggable.
- The **ondragstart** attribute is set to the function **dragStart(event)**, which will be called when the drag starts.

- The "dropZone" div is created with an id.
- The **ondragover** attribute is set to the function **allowDrop(event)**, which allows dropping by preventing the default behavior.
- The **ondrop** attribute is set to the function **drop(event)**, which handles the drop event.

- **dragStart Function:**

    - This function is called when the drag starts (**ondragstart**).
    - It sets data to be transferred during the drag using **event.dataTransfer.setData**. In this case, it sets the id of the dragged element.
- **allowDrop Function:**
    - This function is called during the **ondragover** event.
    - It prevents the default behavior (allowing dropping) using **event.preventDefault()**.
- **drop Function:**
    - This function is called during the **ondrop** event.
    - It prevents the default behavior using **event.preventDefault()**.
    - It retrieves the dragged element's id from the data transfer using **event.dataTransfer.getData**.
    - It gets the dragged element by id using **document.getElementById**.
    - It appends the dragged element to the drop zone (**#dropZone**) using **appendChild**.

Bootstrap

Tailwind and CSS based web development

Vue JS: Directives and Binding

React JS: Elements, UI Components, JSX, Redux, Fetch Data and Asynchronous Programming, GraphQL APIs