JavaScript Event Handling: Exercise-1 with Solution

Write a JavaScript program that creates a button and add a click event listener to log a message when it's clicked.

**Sample Solution:**

**JavaScript Code:**

// Create a button element

const button = document.createElement('button');

button.textContent = 'Click me';

// Add click event listener to the button

button.addEventListener('click', () => {

console.log('Button clicked!');

});

// Append the button to the document body

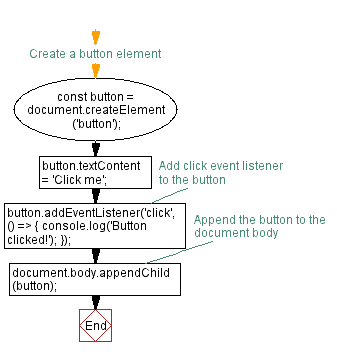
document.body.appendChild(button);

**Explanation:**

First, we create a new button element using the "createElement()" method and set its text content to "Click me". We then add a click event listener to the button using the "addEventListener()" method. When the button is clicked, the callback function inside the event listener is executed, which logs the message "Button clicked!" to the console.

Finally, we append the button to the document body using the "appendChild()" method so that it becomes visible on the page.

**Flowchart:**



JavaScript Event Handling: Exercise-2 with Solution

Write a JavaScript program to create a dropdown menu that shows and hides its options when clicked.

**Sample Solution:**

**HTML and JavaScript Code:**

<!DOCTYPE html>

<html>

<head>

<style>

.dropdown {

position: relative;

display: inline-block;

}

.dropdown-button {

background-color:#7FFFD4;

color: #333;

padding: 10px;

border: none;

cursor: pointer;

}

.dropdown-options {

display: none;

position: absolute;

background-color: #CC56FF;

min-width: 120px;

box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);

padding: 0;

margin: 0;

list-style: none;

z-index: 1;

}

.dropdown-option {

padding: 10px;

cursor: pointer;

border-bottom: 1px solid #ccc;

}

.dropdown-option:last-child {

border-bottom: none;

}

</style>

</head>

<body>

<div class="dropdown">

<button class="dropdown-button">Select a Subject</button>

<ul id="dropdown-options" class="dropdown-options">

<li class="dropdown-option">Mathematics</li>

<li class="dropdown-option">English</li>

<li class="dropdown-option">Physics</li>

</ul>

</div>

<script>

const dropdownButton = document.querySelector('.dropdown-button');

const dropdownOptions = document.querySelector('.dropdown-options');

dropdownButton.addEventListener('click', () => {

dropdownOptions.style.display = dropdownOptions.style.display === 'none' ? 'block' : 'none';

});

</script>

</body>

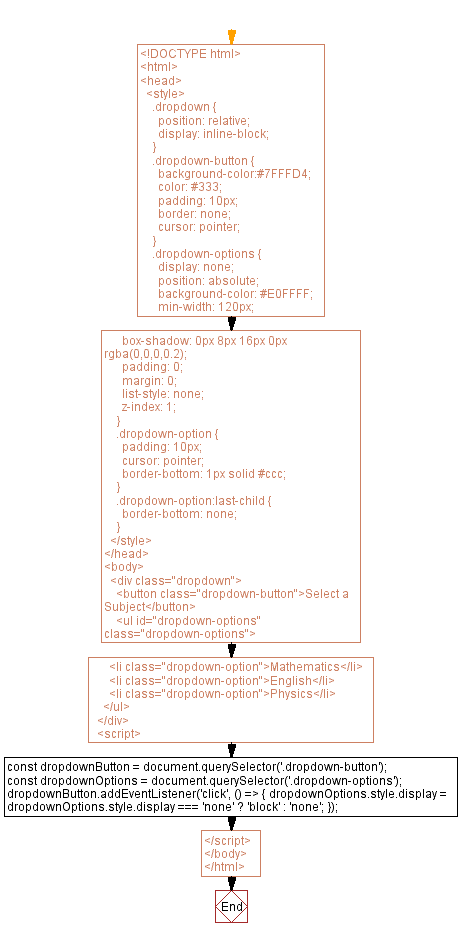
</html>

**Explanation:**

In the above exercise -

* First the HTML document creates a dropdown menu with three options: Mathematics, English, and Physics. Here's an explanation of the code:
* The <style> section contains CSS styles for the dropdown and its options.
* Inside the <body> section, there is a <div> element with the class "dropdown" that acts as a container for the dropdown menu.
* Inside the <div>, there is a <button> element with the class "dropdown-button". This button is the visible part of the dropdown menu that users can click to show or hide the options.
* Below the button, there is a <ul> element with the id "dropdown-options" and the class "dropdown-options". This is an unordered list of dropdown options.
* Inside the <ul>, there are three <li> elements with the class "dropdown-option". These list items represent the available options in the dropdown menu.
* In the <script> section, JavaScript code is used to add functionality to the dropdown menu. It selects the dropdown button and dropdown options using document.querySelector() and assigns them to the dropdownButton and dropdownOptions variables.
* An event listener is added to the dropdown button using addEventListener(). When the button is clicked, the event listener function is executed.

**Flowchart:**



JavaScript Event Handling: Exercise-3 with Solution

Write a JavaScript function that changes the background color of an element when a mouse enters it.

**Sample Solution:**

**HTML and JavaScript Code:**

<!DOCTYPE html>

<html>

<head>

<style>

.my-element {

width: 200px;

height: 200px;

background-color: lightgray;

}

</style>

</head>

<body>

<div id="myDiv" class="my-element"></div>

<script>

function changeBackgroundColor(elementId, color) {

const element = document.getElementById(elementId);

if (element) {

element.addEventListener('mouseenter', () => {

element.style.backgroundColor = color;

});

}

}

// Call the function with element ID and desired color

changeBackgroundColor('myDiv', 'green');

</script>

</body>

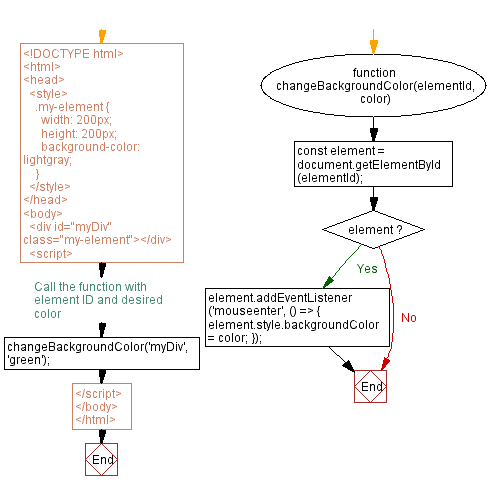
</html>

**Explanation:**

In the above exercise -

* The "changeBackgroundColor()" function takes two parameters: elementId and color. The elementId is the ID of the element you want to target, and color is the desired background color.
* Inside the function, it first retrieves the element using document.getElementById() and assigns it to the element variable.
* Event listeners are added to the element using addEventListener(). The event being listened to is 'mouseenter', which triggers when the mouse enters the element.
* When the 'mouseenter' event is triggered, the callback function sets the backgroundColor CSS property of the element to the specified color value.

**Flowchart:**



JavaScript Event Handling: Exercise-4 with Solution

Write a JavaScript program that implements a "form" validation that displays an error message if a required field is left empty when submitting the form.

**Sample Solution:**

**HTML and JavaScript Code:**

<!DOCTYPE html>

<html>

<head>

<style>

.error-message {

color: red;

margin-top: 5px;

}

</style>

</head>

<body>

<form id="myForm">

<label for="name">Name:</label>

<input type="text" id="name" required>

<br>

<label for="email">Email:</label>

<input type="email" id="email" required>

<br>

<label for="message">Message:</label>

<textarea id="message" required></textarea>

<br>

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

</form>

<div id="errorMessages"></div>

<script>

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

const errorMessagesDiv = document.getElementById('errorMessages');

form.addEventListener('submit', (event) => {

event.preventDefault();

errorMessagesDiv.innerHTML = '';

const requiredFields = form.querySelectorAll('[required]');

requiredFields.forEach((field) => {

if (field.value.trim() === '') {

const fieldName = field.getAttribute('name');

const errorMessage = document.createElement('p');

errorMessage.textContent = `${fieldName} is required.`;

errorMessagesDiv.appendChild(errorMessage);

}

});

});

</script>

</body>

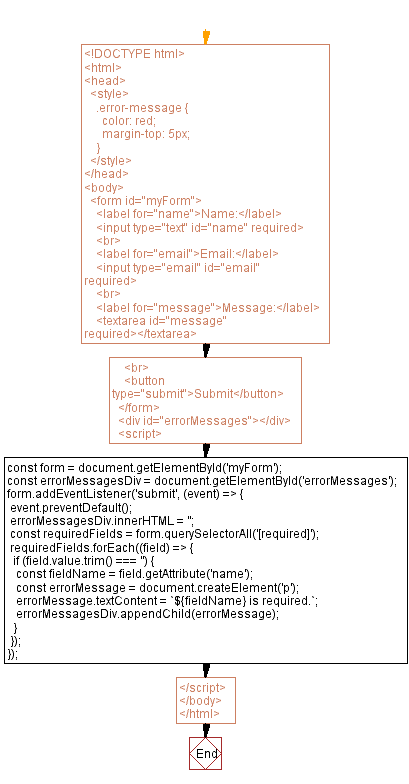
</html>

**Explanation:**

In the above exercise,

* The HTML structure consists of a <form> element with an id of "myForm" that contains several input fields and a submit button. Below the form, there is a <div> element with an id of "errorMessages" which serves as a container for displaying error messages.
* The JavaScript code attaches an event listener to the form's "submit" event using addEventListener. When the form is submitted, the event listener function is triggered.
* Inside the event listener function, the default form submission behavior is prevented using event.preventDefault() to stop the form from being submitted and the page from being refreshed.
* The code then retrieves all the input fields with the required attribute using form.querySelectorAll('[required]'). It loops through each of these required fields using forEach and checks if the field's value, after trimming any leading or trailing spaces, is empty. If a required field is empty, an error message is created as a <p> element. Its content is set to indicate the field name followed by the text "is required."
* The error message element is then appended to the "errorMessagesDiv" container using appendChild.

**Flowchart:**



JavaScript Event Handling: Exercise-5 with Solution

Write a JavaScript program to create a slideshow that changes the displayed image when a next or previous button is clicked.

**Sample Solution:**

**HTML and JavaScript Code:**

<!DOCTYPE html>

<html>

<head>

<style>

.slideshow {

display: flex;

justify-content: center;

align-items: center;

height: 300px;

}

.slideshow img {

width: 300px;

height: 300px;

}

</style>

</head>

<body>

<div class="slideshow">

<button id="previousBtn">Previous</button>

<img id="image" src="image1.jpg">

<button id="nextBtn">Next</button>

</div>

<script>

const images = ['image1.jpg', 'image2.jpg', 'image3.jpg'];

let currentIndex = 0;

const previousBtn = document.getElementById('previousBtn');

const nextBtn = document.getElementById('nextBtn');

const image = document.getElementById('image');

previousBtn.addEventListener('click', () => {

currentIndex = (currentIndex - 1 + images.length) % images.length;

image.src = images[currentIndex];

});

nextBtn.addEventListener('click', () => {

currentIndex = (currentIndex + 1) % images.length;

image.src = images[currentIndex];

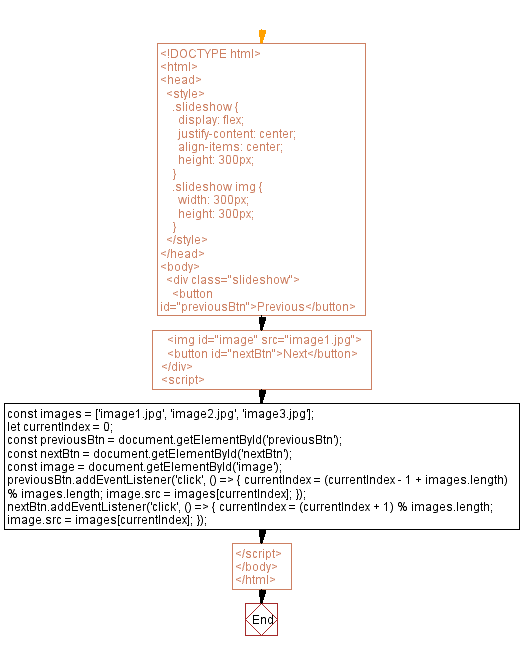
});

</script>

</body>

</html>

**Flowchart:**



JavaScript Event Handling: Exercise-6 with Solution

Write a JavaScript program to implement drag-and-drop functionality to allow users to reorder items in a list.

**Sample Solution:**

**HTML and JavaScript Code:**

<!DOCTYPE html>

<html>

<head>

<style>

.drag-list {

list-style: none;

padding: 0;

}

.drag-item {

background-color: #CC56FF;

padding: 10px;

margin-bottom: 5px;

cursor: move;

}

</style>

</head>

<body>

<ul id="dragList" class="drag-list">

<li class="drag-item" draggable="true">Mobile</li>

<li class="drag-item" draggable="true">Laptop</li>

<li class="drag-item" draggable="true">Desktop</li>

<li class="drag-item" draggable="true">Television</li>

<li class="drag-item" draggable="true">Radio</li>

</ul>

<script>

const dragList = document.getElementById('dragList');

let draggedItem = null;

// Add event listeners for drag and drop events

dragList.addEventListener('dragstart', handleDragStart);

dragList.addEventListener('dragover', handleDragOver);

dragList.addEventListener('drop', handleDrop);

// Drag start event handler

function handleDragStart(event) {

draggedItem = event.target;

event.dataTransfer.effectAllowed = 'move';

event.dataTransfer.setData('text/html', draggedItem.innerHTML);

event.target.style.opacity = '0.5';

}

// Drag over event handler

function handleDragOver(event) {

event.preventDefault();

event.dataTransfer.dropEffect = 'move';

const targetItem = event.target;

if (targetItem !== draggedItem && targetItem.classList.contains('drag-item')) {

const boundingRect = targetItem.getBoundingClientRect();

const offset = boundingRect.y + (boundingRect.height / 2);

if (event.clientY - offset > 0) {

targetItem.style.borderBottom = 'solid 2px #000';

targetItem.style.borderTop = '';

} else {

targetItem.style.borderTop = 'solid 2px #000';

targetItem.style.borderBottom = '';

}

}

}

// Drop event handler

function handleDrop(event) {

event.preventDefault();

const targetItem = event.target;

if (targetItem !== draggedItem && targetItem.classList.contains('drag-item')) {

if (event.clientY > targetItem.getBoundingClientRect().top + (targetItem.offsetHeight / 2)) {

targetItem.parentNode.insertBefore(draggedItem, targetItem.nextSibling);

} else {

targetItem.parentNode.insertBefore(draggedItem, targetItem);

}

}

targetItem.style.borderTop = '';

targetItem.style.borderBottom = '';

draggedItem.style.opacity = '';

draggedItem = null;

}

</script>

</body>

</html>

**Explanation:**

In the above exercise,

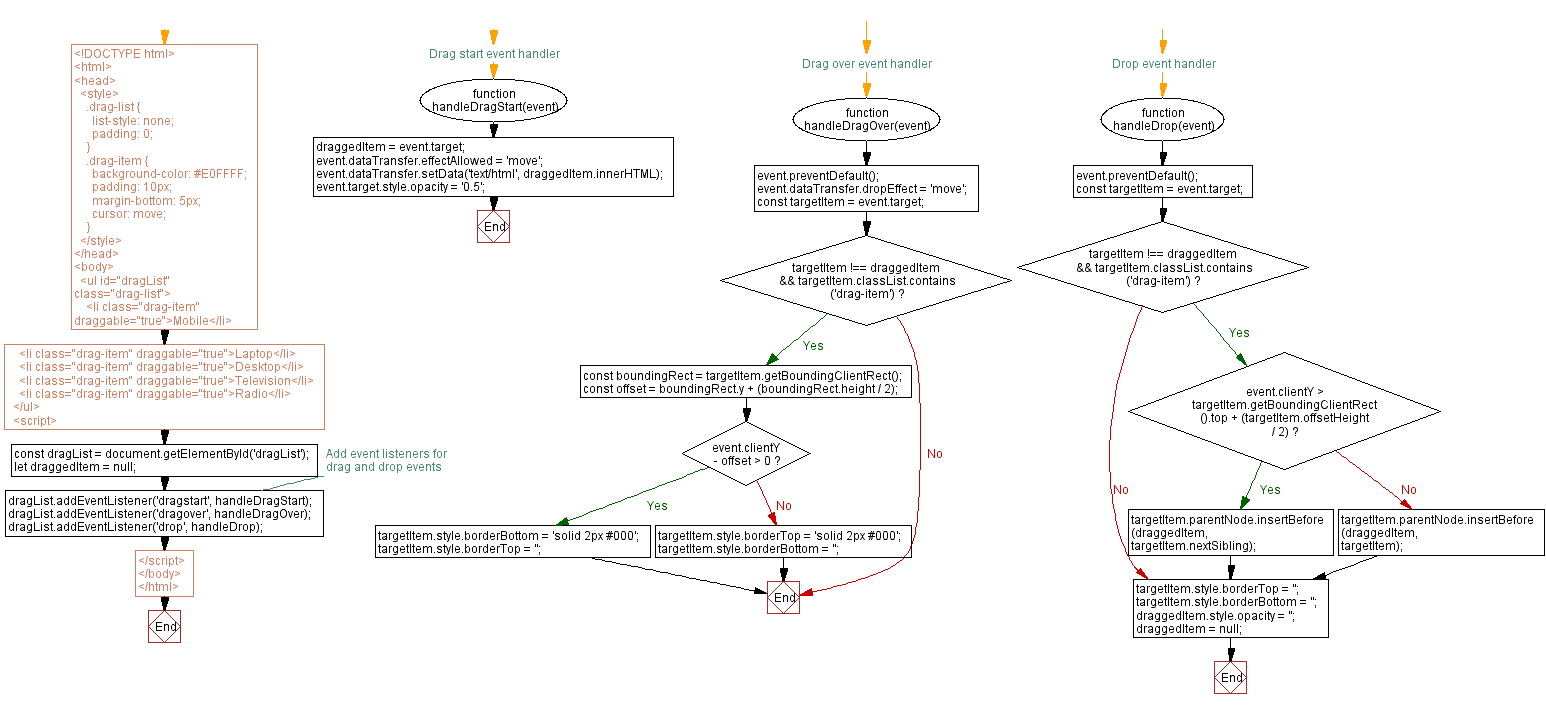
We have an unordered list (<ul>) with the ID dragList. Each list item (<i>) inside the list has the class drag-item and the draggable attribute set to true.

Next code adds event listeners for dragstart, dragover, and drop events to the dragList element.

When a dragstart event occurs on a draggable item, the handleDragStart function is called. It sets the dragged item (draggedItem) and the data to be transferred during the drag operation.

During the dragover event, the handleDragOver function.

**Flowchart:**



JavaScript Event Handling: Exercise-7 with Solution

Write a JavaScript program to implement a toggle switch that changes its state when clicked.

**Sample Solution:**

**HTML and JavaScript Code:**

<!DOCTYPE html>

<html>

<head>

<style>

.toggle {

display: inline-block;

width: 60px;

height: 34px;

position: relative;

border-radius: 34px;

background-color: #ccd;

cursor: pointer;

transition: background-color 0.3s;

}

.toggle::before {

content: "";

position: absolute;

width: 24px;

height: 24px;

border-radius: 50%;

background-color: #fff;

top: 4px;

left: 4px;

transition: transform 0.3s;

}

.toggle.on {

background-color: #66bb6a;

}

.toggle.on::before {

transform: translateX(26px);

}

</style>

</head>

<body>

<div class="toggle"></div>

<script>

const toggle = document.querySelector('.toggle');

toggle.addEventListener('click', () => {

toggle.classList.toggle('on');

});

</script>

</body>

</html>

**Explanation:**

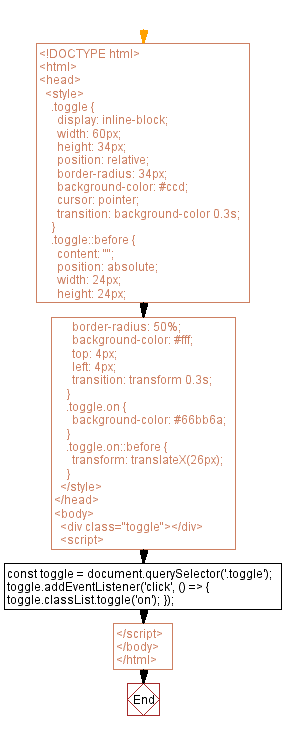
In the above exercise,

We have a <div> element with the class toggle that represents a toggle switch. The switch has an initial state of "off" (default) defined by the .toggle class. When the switch is toggled on, the .on class is added to the toggle element, changing its appearance.

CSS styles define the toggle switch's appearance and behavior. The .toggle class sets the dimensions, position, border radius, and transition properties. The .toggle::before pseudo-element creates the circular handle of the switch. The .toggle.on class changes the background color of the switch when it is toggled on. The .toggle.on::before class transforms the handle to the "on" position.

The JavaScript code attaches a click event listener to the toggle element. When the toggle switch is clicked, the event listener toggles the .on class on the toggle element using the classList.toggle() method. This changes the state of the switch and triggers the corresponding CSS styles for the on or off state.

**Flowchart:**



JavaScript Event Handling: Exercise-8 with Solution

Write a JavaScript program to create a progress bar that updates its width based on task completion.

**Sample Solution:**

**HTML and JavaScript Code:**

<!DOCTYPE html>

<html>

<head>

<style>

.progress-bar {

width: 300px;

height: 20px;

background-color: #f0f0f0;

border-radius: 10px;

overflow: hidden;

}

.progress-bar-fill {

height: 100%;

background-color: #4caf50;

transition: width 0.3s;

}

</style>

</head>

<body>

<div class="progress-bar">

<div class="progress-bar-fill" id="progress"></div>

</div>

<script>

function updateProgress(progressPercentage) {

const progressBarFill = document.getElementById('progress');

progressBarFill.style.width = `${progressPercentage}%`;

}

// Example usage: updating progress to 75%

updateProgress(75);

</script>

</body>

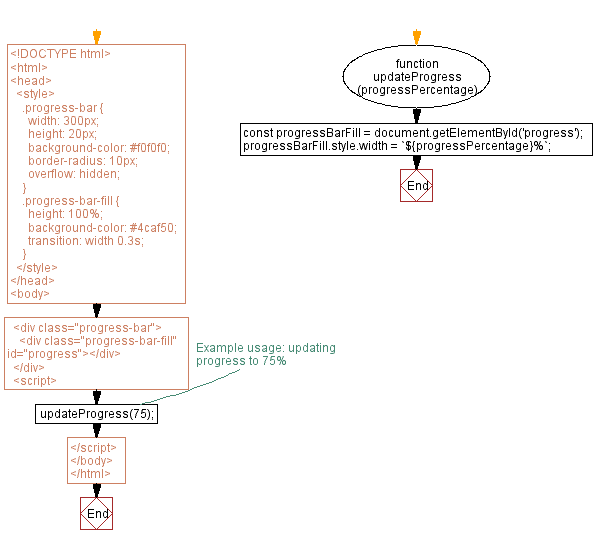
</html>

**Explanation:**

In the above exercise,

* The HTML structure consists of a <div> element with a class of "progress-bar". This serves as the progress bar container. Inside this container, there is another <div> element with a class of "progress-bar-fill" and an id of "progress". This inner div represents the progress bar itself and will be dynamically updated to reflect progress.
* The CSS styles defined in the <style> block specify the progress bar appearance. The "progress-bar" class sets the width, height, background color, border radius, and overflow properties. The "progress-bar-fill" class sets the height and background color, and includes a transition effect to smoothly update the width property.
* The JavaScript code defines a function named "updateProgress" that takes a parameter called "progressPercentage". Inside the function, it retrieves the element with the id "progress" using document.getElementById('progress'). This element represents the progress bar fill.
* At the end of the code, there is an example usage of the "updateProgress" function, setting the progress to 75% by calling updateProgress(75).

**Flowchart:**



JavaScript Event Handling: Exercise-9 with Solution

Write a JavaScript program that adds a keydown event listener to a text input to detect when the "Enter key" is pressed.

**Sample Solution:**

**HTML and JavaScript Code:**

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<input type="text" id="textInput" placeholder="Press Enter...">

<script>

const textInput = document.getElementById('textInput');

textInput.addEventListener('keydown', (event) => {

if (event.key === 'Enter') {

console.log('Enter key pressed!');

// Perform desired actions here

}

});

</script>

</body>

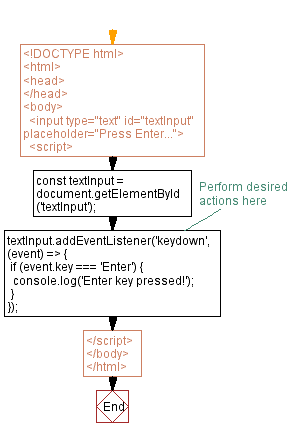
</html>

**Explanation:**

In the above exercise -

* First we define an input element with the id "textInput". The JavaScript code retrieves this input element using document.getElementById('textInput') and assigns it to the textInput variable.
* Next we attach an event listener to the text input using the addEventListener method. We specify the event type as "keydown" to detect when any key is pressed while the text input has focus.
* Inside the event listener function, we check if the event.key property is equal to "Enter". If it is, we log a message to the console indicating that the Enter key was pressed.

**Flowchart:**



JavaScript Event Handling: Exercise-10 with Solution

Write a JavaScript function that listens for a double click on an element and performs a specific action.

**Sample Solution:**

**HTML and JavaScript Code:**

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<button id="myButton">Double click me!</button>

<script>

const myButton = document.getElementById('myButton');

myButton.addEventListener('dblclick', () => {

console.log('A double click has been performed!');

// Perform desired actions here

});

</script>

</body>

</html>

**Explanation:**

In the above exercise,

* First, we define a button element with the id "myButton". The JavaScript code retrieves this button element using document.getElementById('myButton') and assigns it to the myButton variable.
* Next we attach an event listener to the button using the addEventListener method. We specify the event type as "dblclick" to detect double clicks on the button.
* Inside the event listener function, we log a message to the console indicating that the double click action was performed. You can replace the console.log statement with your desired action.

**Flowchart:**

