Class 16 - Events and Events handling

1. Problem-1 Make the Filter Work

A practical exercise where the task is to make a filter function work. The scenario involves filtering movie tickets based on categories like action, romance, and comedy. The solution involves manipulating the DOM to only display movies of the selected category.

2. Problem-2 Key Tap Implementation for Section Scrolling and Page Navigation

This section focuses on enhancing navigation within a web page through keyboard inputs. It includes implementing functionality where pressing certain keys (1, 2, 3, b, t) navigates the user to different sections of the page or to the top/bottom of the page.

3. Problem-3 Remove sepecific and remove ALL

The task involves making a "Remove All" button that deletes all items in a list and individual delete buttons for each item. It's a common pattern seen in to-do lists or item management systems.

3. Probem-4 Destroy and create Buttons

This segment covers the Destroy and Create Buttons problem. The exercise involves a button that, upon being clicked, is replaced by two new buttons, demonstrating dynamic DOM manipulation.

## Make the filter work

### Problem statement

1. Develop a dynamic filter functionality for a webpage that lists movies by their categories, such as Action, Romance, and Comedy.
2. When a user selects a category from a dropdown menu, the page should update to display only the movies that belong to the selected category.
3. If "None" is selected, all movies should be shown. This feature should enhance the user experience by allowing them to easily find movies of their interest without scrolling through all available options.
4. Boiler plate code below

```
<!DOCTYPE html>
<html lang = "en">
 <head>
   <meta charset = "UTF - 8" />
   <meta http - equiv = "X - UA - Compatible" content = "IE = edge" />
   <meta name = "viewport" content = "width = device - width, initial - scale =
1.0" />
   <title>Document</title>
   <style>
     * {
```

```
    box-sizing: border-box;
  }

  body {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-around;
    padding-top: 5rem;
  }

  .movies {
    height: 10rem;
    width: 10rem;
    margin: 2rem;
  }

  .price {
    background-color: rgb(247 84 33);
    height: 8rem;
    padding: 1.5rem;
    font-size: 2rem;
  }

  .heading {
    background-color: rgba(0, 0, 0, 0.856);
    height: 2rem;
  }

  .heading,
  .price {
    color: white;
    text-align: center;
    margin: 0;
  }

  select,
  h1 {
    position: fixed;
  }

  h1 {
    top: 0px;
```

```
    }

    select {
      width: 8rem;
    }


  </style>
</head>

<body>
  <h1>BOOK MY SHOW</h1>
  <select>
    <option value = "none">None</option>
    <option value = "action">Action</option>
    <option value = "romance">Romance</option>
    <option value = "comedy">Comedy</option>
  </select>

  <div class = "movies">
    <h3 class = "heading">Action Movie - 1</h3>
    <p class = "price" data-category = "action">Rs. 100</p>
  </div>
  <div class = "movies">
    <h3 class = "heading">Action Movie - 2</h3>
    <p class = "price" data-category = "action">Rs 200</p>
  </div>
  <div class = "movies">
    <h3 class = "heading">Action Movie - 3</h3>
    <p class = "price" data-category = "action">Rs. 150</p>
  </div>
  <div class = "movies">
    <h3 class = "heading">Romance - 1</h3>
    <p class = "price" data-category = "romance">Rs 200</p>
  </div>
  <div class = "movies">
    <h3 class = "heading">Romance - 2</h3>
    <p class = "price" data-category = "romance">Rs. 150</p>
  </div>
  <div class = "movies">
    <h3 class = "heading">Romance - 3</h3>
    <p class = "price" data-category = "romance">Rs. 150</p>
```

```
  </div>
  <div class = "movies">
    <h3 class = "heading">Action Movie-4</h3>
    <p class = "price" data-category = "action">Rs 200</p>
  </div>
  <div class = "movies">
    <h3 class = "heading">Comedy - 1</h3>
    <p class = "price" data-category = "comedy">Rs. 100</p>
  </div>
  <div class = "movies">
    <h3 class = "heading">Romance - 4</h3>
    <p class = "price" data-category = "romance">Rs. 100</p>
  </div>
  <div class = "movies">
    <h3 class = "heading">Comedy - 2</h3>
    <p class = "price" data-category = "comedy">Rs 200</p>
  </div>
  <script>
    // Q Make the filter work


  </script>
 </body>
</html>
```

Approach

1. Access Dropdown Menu: Identify and select the dropdown menu element on the page.
2. Monitor Dropdown Changes: Set up a mechanism to detect when the user selects a different option in the dropdown menu.
3. Retrieve Selected Category: When a change is detected, determine which category the user has selected from the dropdown.

4. Access All Movie Elements: Identify all the movie elements listed on the page.
5. Determine Movies to Display: For each movie, check if its category matches the selected category from the dropdown.
6. Update Movie Visibility: Based on the matching result, adjust the visibility of each movie element. If a movie matches the selected category, it should be visible. If it does not match, it should be hidden.
7. Special Case for 'None' Selection: If the user selects "None" that signifies no filter should be applied, ensure all movies are shown regardless of category.
8. Apply the Changes: Implement the visibility adjustments so that only the relevant movies are displayed according to the user's selection.

Solution

1. Select the dropdown and add listener for option change

```
<script>
    // Q Make the filter work
    const select = document.querySelector("select");

    select.addEventListener("change", function () {
      const filterVal = select.value;

      console.log(filterVal);
    });
  </script>
```

2. Now we need to identify how to get all the tickets and then filter based on filterVal
    a. What we can see from structure is that paragraphs have data-category attribute which we can use to filter from values eventually
3. So we can select all para first

```
select.addEventListener("change", function () {
        const filterVal = select.value;
        console.log(filterVal);
      });
      const allTickets =
document.querySelectorAll('.price')
```

4. Now we need to select the tickets based on category

```
if(filter is none){
       // show all tickets
      } else {
        // show only the tickets that match the filter
    }
```

5. Let us breakdown the else part where we add filter based on filterval

```
else {
        // show only the tickets that match the filter
        // loop over all the tickets
        // get the attribute of the ticket
        // if the attribute matches the filterVal, show the
ticket
```

```
        // else hide the ticket
        // to hide the ticket, use the style property and
set the display to none
        // we need to hide the parent of the para that is
the div
    }
```

6. Let us learn how to implement these steps

```
for (let i = 0; i < allTickets.length; i++) {
          if (allTickets[i].getAttribute("data-category") !=
filterVal) {
            allTickets[i].parentElement.style.display =
"none";
          } else {
            allTickets[i].parentElement.style.display =
"block";
          }
        }
```

Explanation:

style is a property that allows us to get or set the style of an element. When we set this property, we can change how the element looks on the page. This includes its color, size, display property, and more.

What it means here: By accessing .style on the parent element of allTickets[i], we're preparing to change one or more of its CSS properties directly through JavaScript.

The complete line allTickets[i].parentElement.style.display = "none"; finds the i-th element in the allTickets array, goes up one level to access its parent, and then changes that parent's display property to "none".

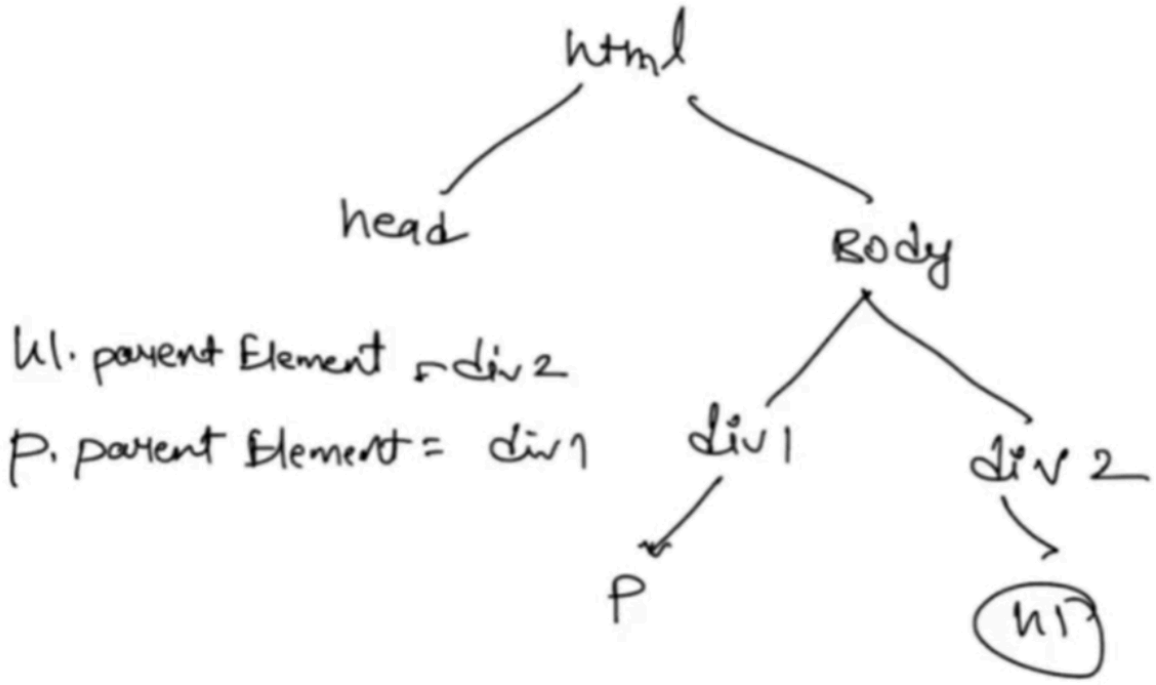
As a result, the parent element gets hidden from view on the webpage. This is a common technique used in web development to dynamically show or hide elements based on user actions or other conditions without removing them from the document structure entirely.

7. Lets do the if part where if the user selects filter as none, we show all the tickets
    a. Basically we set the display property for all elements as block

```
if (filterVal === "none") {
          // show all tickets
          for (let i = 0; i < allTickets.length; i++) {
            allTickets[i].parentElement.style.display =
"block";
          }
        }
```

8. ParentElement

a.




## Problem - 2

Key Tap Implementation for Section Scrolling and Page Navigation

Desc- Implement key taps such that on pressing 1 you scroll to section 1, pressing 2 you scroll to section 2 and pressing 3 you scroll to section 3. Also implement key tap b to go to bottom of the page and key tap t to go to top of the page

Task - The problem statement requires you to implement a feature in a webpage where specific key presses trigger scrolling actions. Pressing the keys '1', '2', or '3' should scroll the page to predefined sections labeled as Section 1, Section 2, and Section 3, respectively.

Additionally, pressing 'b' should scroll to the bottom of the page, and pressing 't' should scroll to the top.

Boiler plate code:

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta http - equiv="X-UA-Compatible" content="IE = edge" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Document</title>
   <style>
     p {
       font-size: 1.5rem;
     }
   </style>
 </head>
 <body>
   <h1>
     Press 1 to go to Section-1 <br />
     Press 2 to go to Section-2 <br />
     Press 3 to go to Section-3 <br />
     Press t to go to top of the page <br />
     Press b to go to bottom of the page
   </h1>

   <h2 id="s1">Section 1</h2>
<p>
     lorem5000
   </p>

   <h2 id="s2">Section 2</h2>
<p>
     lorem5000
   </p>

   <h2 id="s3">Section 2</h2>
<p>
     lorem1000
   </p>
```

```
<script>
    // Q Implement key taps such that on pressing 1 you scroll to section 1,
pressing 2 you scroll to section 2 and  pressing 3 you scroll to section 3. Also
implement key tap b to go to bottom of the page and key tap t to go to top of the page
    const s1 = document.querySelector("#s1");
    const s2 = document.querySelector("#s2");
    const s3 = document.querySelector("#s3");



  </script>
```

Approach

1. High-Level Approach:
2. Listen for Key Presses: You'll need to set up an event listener for keyboard events. This listener should detect when a user presses a key.
3. Identify the Key Pressed: Within the event listener, you'll need logic to identify which key was pressed. Specifically, you're interested in the keys '1', '2', '3', 'b', and 't'.
4. Scroll to the Specific Section: Once you've identified which key was pressed, you'll need to implement the logic to scroll the page accordingly.
5. For '1', '2', and '3', you'll locate the corresponding section on the page (using an identifier like an id) and then scroll to that section.

6. For 'b', you'll scroll to the bottom of the page. This might involve calculating the total height of the page's content and scrolling to that position.
7. For 't', you'll scroll back to the top of the page, which can be done by scrolling to top of the page.
8. So Now again as we Know the High level approach to tackle this problem , let's try solving this on the lower level , lets write some code and make this work
9. In this process of implementing, we will learn few new methods to aid in our work
10. **Before moving to the Solution let us discuss a specific method that we will be helpful here **getBoundingClientRect**()

getBoundingClientRect() Method

1. getBoundingClientRect(): The getBoundingClientRect() method is a JavaScript method that is used to retrieve the position and dimensions of an element relative to the viewport (the visible area of a web page).
2. In simple words, The getBoundingClientRect() is a tool you can use in JavaScript, kind of like a measuring tape for elements on a webpage. Imagine you want to find out exactly where an image or a box is on a webpage and how big it is. This method helps you do just that.
3. When you use getBoundingClientRect() on an element, JavaScript returns a **'DOMRect'** object. The full form of

'DOMRect' is '**Document Object Model Rectangle**.' Think of it as a virtual rectangle that wraps around an element, giving us detailed information about its position and size on the web page.

4. Create another html and check the console

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta http - equiv="X - UA - Compatible" content="IE =
edge" />
   <meta
     name="viewport"
     content="width = device - width, initial - scale =
1.0"
   />
   <title>Document</title>

   <style>
       *{
           box-sizing: border-box;
           margin:0;
           padding:0;
       }
     #hello {
       height: 100px;
       width: 150px;
       position: absolute;
       top: 20px;
       left: 100px;
     }
   </style>
```

```
</head>
<body>
  <div id="hello">Hello</div>

  <script>
    let helloDiv = document.querySelector("#hello");

    const results = helloDiv.getBoundingClientRect();
    console.log(results);
  </script>
</body>
</html>
```

So now you know that you can get the position and dimensions of an element relative to the viewport (the visible area of a web page) , so based on this we can now navigate to the position based on measurements

Approach

1. Select the elements where we need to navigate to

```
<script>
    // Q Implement key taps such that on pressing 1 you scroll to section 1,  pressing 2 you scroll to section 2 and  pressing 3 you scroll to section 3. Also implement key tap b to go to bottom of the page and key tap t to go to top of the page
    const s1 = document.querySelector("#s1");
```

```
    const s2 = document.querySelector("#s2");
    const s3 = document.querySelector("#s3");
  </script>
```

2. Add an event listener to the keydown event handler
3. We add the event listener for keydown on the entire document

```
const s1 = document.querySelector("#s1");
    const s2 = document.querySelector("#s2");
    const s3 = document.querySelector("#s3");

    document.addEventListener("keydown", function(e){
      if(e.key == 1){
          // scroll to section 1
      } else if(e.key == 2){
          // scroll to section 2
      } else if(e.key == 3){
          // scroll to section 3
      } else if(e.key == "b"){
          // scroll to bottom
      } else if(e.key == "t"){
          // scroll to top
      }
    })
```

4. There is a scrollIntoView method
    a. Pressing "1", "2", or "3": We write script that checks if the key pressed is "1", "2", or "3". Depending on the key, it scrolls to the corresponding section (s1, s2, or s3) using the

scrollIntoView() method. This method scrolls the specified element into the visible area of the browser window.

b. code

```
if (e.key == 1) {
          s1.scrollIntoView();
        } else if (e.key == 2) {
          s2.scrollIntoView();
        } else if (e.key == 3) {
          s3.scrollIntoView();
```

5. Let us see one more variation where we need to scroll to the top and bottom of the page
    a. There is a method scrollBy which takes two params
        i. scroll in x direction and in the y direction
        ii. We can use the getBoundingClientRect to get the total height of the html document
        iii. For bottom, we can ask browser to go the height or the bottom
            1. Pressing "b" (bottom): If the key pressed is "b", the window is scrolled to the bottom of the page. This is done by using window.scrollBy() with the *document.querySelector("html").getBoundingClientRect().height* as the vertical distance. This effectively scrolls down by the height of the document, moving to the bottom

iv. For top, we can use the same height in negative direction ( although not an accurate way but it will work in our case )

1. Pressing "t" (top): Similarly, if the key pressed is "t", the window is scrolled to the top of the page. This is achieved by scrolling up by the negative height of the document.

```
else if (e.key == "b") {
          // scroll to bottom
          window.scrollBy(
            0,

document.querySelector("html").getBoundingClientRect().height
          );
        } else if (e.key == "t") {
          // scroll to top
          window.scrollBy(
            0,

-document.querySelector("html").getBoundingClientRect().height
          );
        }
```

## Full code

```
<script>
    // Q Implement key taps such that on pressing 1 you scroll to section 1,
pressing 2 you scroll to section 2 and  pressing 3 you scroll to section 3.
Also implement key tap b to go to bottom of the page and key tap t to go to top
of the page
    const s1 = document.querySelector("#s1");
    const s2 = document.querySelector("#s2");
```

```
    const s3 = document.querySelector("#s3");

    document.addEventListener("keydown", function (e) {
      if (e.key == 1) {
        s1.scrollIntoView();
      } else if (e.key == 2) {
        s2.scrollIntoView();
      } else if (e.key == 3) {
        s3.scrollIntoView();
      } else if (e.key == "b") {
        // scroll to bottom
        window.scrollBy(
          0,
          document.querySelector("html").getBoundingClientRect().height
        );
      } else if (e.key == "t") {
        // scroll to top
        window.scrollBy(
          0,
          -document.querySelector("html").getBoundingClientRect().height
        );
      }
    });
  </script>
```

## Problem: Remove and Remove All Button

Problem statement

1. Implement JavaScript functionality for a task management web application that allows users to individually remove tasks from a list or clear the entire list at once.
2. The HTML template includes a list of tasks, each with a dedicated "delete" button, and a separate "Remove All" button for bulk deletion.
3. Your solution should enable the "delete" buttons to remove their respective tasks from the list and the "Remove All" button to clear all tasks from the list simultaneously.
4. Boiler plate below

```
<!DOCTYPE html>
<html lang = "en">
 <head>
   <meta charset = "UTF - 8" />
   <meta http-equiv = "X - UA - Compatible" content = "IE = edge" />
   <meta name = "viewport" content = "width = device - width, initial-scale =
1.0" />
   <title>Document</title>
 </head>
 <body>
   <button id = "removeAll">Remove All</button>
   <ul>
     <li>
       <p>Task - 1</p>
       <button class="delete_button">delete</button>
     </li>
     <li>
       <p>Task - 2</p>
       <button class="delete_button">delete</button>
     </li>
     <li>
       <p>Task - 3</p>
       <button class="delete_button">delete</button>
     </li>
```

```
    <li>
      <p>Task - 4</p>
      <button class="delete_button">delete</button>
    </li>
  </ul>
  <script>
    // Q- Make remove all button work  and make individual delete btn work

  </script>
</body>
</html>
```

Approach:

1. Select the remove all button
    a. Add listener
    b. When clicked, remove everything inside ul
2. Code

```
<script>
    // Q- Make remove all button work  and make
individual delete btn work
    // select remove All button
    document
      .getElementById("removeAll")
      .addEventListener("click", function () {
        // Select the ul element containing the list
items
        const ul = document.querySelector("ul");
        // Remove all child elements of the ul,
effectively clearing the list
        ul.innerText = "";
      });
  </script>
```

Approach for individual delete buttons

1. Select each delete buttons and add event listeners to each
2. For each event , remove the parent i.e the containing li item
3. Code

```
     const allDeleteButton =
document.querySelectorAll(".delete_button");
```

```
for (let i = 0; i < allDeleteButton.length; i ++ ) {
          allDeleteButton[i].addEventListener("click",
function (e) {
        e.target.parentElement.remove();
      });
```

4. Whole code

```
<script>
     // Q- Make remove all button work  and make individual
delete btn work
     // select remove All button
     const allDeleteButton =
document.querySelectorAll(".delete_button");

     document
       .getElementById("removeAll")
       .addEventListener("click", function () {
         // Select the ul element containing the list items
         const ul = document.querySelector("ul");
         // Remove all child elements of the ul,
effectively clearing the list
```

```
        ul.innerText    = "";
      });
      for (let i = 0; i < allDeleteButton.length; i ++ ) {
          allDeleteButton[i].addEventListener("click",
function (e) {
        e.target.parentElement.remove();
      });
    }
  </script>
```

## Problem: Destroy and Create Buttons

1. Problem statement
2. Create a web application feature where clicking a button causes it to be removed from the page, and simultaneously, two new buttons are created and added to the page in its place.
3. Boiler plate code

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF - 8" />
   <meta http-equiv="X - UA - Compatible" content="IE = edge" />
   <meta name="viewport" content="width = device-width, initial-scale = 1.0" />
   <title>Document</title>
 </head>
 <body>
   <div id="doubleHolder">
     <button class="double">double</button>
   </div>

   <script>
```

```
    //Q Create a button that is destroyed by clicking on it but two new
buttons are created in it's place.
  </script>
 </body>
</html>
```

4. Task
    a. This is a very simple boilerplate which will just show a button double as soon as you click on it , the double button will be destroyed and two new buttons should be created on click of it.
5. code

```
script>
    //    Q Create a button that is destroyed by clicking
on it but two new buttons are created in it's place.

    //   Solution:

    // get the doubleHolder section
    const doubleBtn = document.querySelector(".double");
    // add an event
    doubleBtn.addEventListener("click", function (e) {
        // create a new button
        let btn = document.createElement("button");
        // set the class of the new button
        btn.setAttribute("class", "first-button");
        // set the innerHTML of the new button
        btn.innerHTML = "first button";
        // create a new button
        let btn2 = document.createElement("button");
        // set the class of the new button
```

```
        btn2.setAttribute("class", "second-button");
        // set the innerHTML of the new button
        btn2.innerHTML = "second button";
        // append the new button to the doubleHolder
        e.target.parentElement.appendChild(btn);
        // append the new button to the doubleHolder
        e.target.parentElement.appendChild(btn2);
        // remove the clicked button
        e.target.remove();
    });
  </script>
```

Extra

1. Lets say the task is that when you click on the double button, it adds two new button with text button 1 and button 2
2. It deletes the original clicked button
3. It adds the same functionality as in the previous double button to the first button
4. code

```
<script>
    //    Q Create a button that is destroyed by clicking
on it but two new buttons are created in it's place.

    //   Solution:

    // get the doubleHolder section
    const doubleBtn = document.querySelector(".double");
    // add an event
    function cb(e) {
```

```
      // create a new button
      let btn = document.createElement("button");
      // set the class of the new button
      btn.setAttribute("class", "double");
      btn.addEventListener("click", cb);
      // set the innerHTML of the new button
      btn.innerHTML = "first button";
      // create a new button
      let btn2 = document.createElement("button");
      // set the class of the new button
      btn2.setAttribute("class", "second-button");
      // set the innerHTML of the new button
      btn2.innerHTML = "second button";
      // append the new button to the doubleHolder
      e.target.parentElement.appendChild(btn);
      // append the new button to the doubleHolder
      e.target.parentElement.appendChild(btn2);
      // remove the clicked button
      e.target.remove();
    }
    doubleBtn.addEventListener("click", cb);
  </script>
```