Given is the following HTML code:

Events Exercise

In this exercise: - The user should be able to set a color for boxes (this affects both current boxes and new boxes). - The user should be able to add boxes with the set color to the div with the ID `box-container`: - When the button with the ID `new-box-button` is clicked. - When the `N` key is pressed. - The user should be able to remove a box when the box is double-clicked on. - Each box should display its ID. - Each box should display its page coordinates when hovered (when the mouse leaves, it displays its ID back). - Each box should have a class `box` for styling and selecting. We coded the `index.html` and `style.css` files in the starter code for you. Before you start coding the `script.js` file, please review the `index.html` file. After you try on your own, please see the steps if you need guidance on the steps you should follow.

Box Color: [ ] Set as New Color New Box

1. The `script.js` is included in the head of the `index.html`. To run your code after the DOM is loaded, wrap it with a callback function of an event listener (`DOMContentLoaded`).
2. Get the elements from the DOM that you will work on (box container, new box button, color form, color input).
3. Create variables to store the box color and counter for the box ID.
4. When the form is submitted, get the value from the color input element from Step 2 and set this color to all boxes (to get all boxes, use the class name `box`). Remember to reset the color input element's value and store the color in the box color variable we created in Step 3.
5. Create a function that adds a new box. In this function, set the box ID as content, the class name, and the background color from the box color variable we created in Step 3. Besides, set the box ID to a data attribute. You'll need this attribute to set the display text back to ID when the mouse leaves. Since we used this counter ID, increment it using the counter variable we created in Step 3 to keep its uniqueness.
6. When the new box button is clicked, call the function that we created above that adds a new box.
7. To remove a box, listen to the double-click events in the document. If the event's target's class list contains the value `box`, remove the element.
8. To display a box's page coordinates, listen to the mouse over events in the document. If the event's target's class list contains the value `box`, display the coordinates.
9. To display a box's ID back when the mouse leaves after displaying the box's page coordinates, listen to the mouse out events in the document. If the event's target's class list contains the value `box`, get the ID from the box's data attributes and display it.

10. To create a new box when the `N` key is pressed, listen to the key-down events in the document. If the key is `N` (check both upper and lower cases), call the function that adds a new box. Remember to ignore the event if it is triggered from the color input element

Here’s the implementation of the `script.js` file based on the requirements described in your exercise. Each step is carefully followed to ensure all functionalities are implemented.

```
document.addEventListener("DOMContentLoaded", () => {
  // Step 2: Get elements from the DOM
  const boxContainer = document.getElementById("box-container");
  const newBoxButton = document.getElementById("new-box-button");
  const colorForm = document.getElementById("color-form");
  const colorInput = document.getElementById("color-input");

  // Step 3: Create variables for box color and counter for box ID
  let boxColor = "black"; // Default color
  let boxCounter = 1;

  // Step 4: Handle form submission to set a new color
  colorForm.addEventListener("submit", (event) => {
    event.preventDefault(); // Prevent form from refreshing the page
    const newColor = colorInput.value.trim();
    if (newColor) {
      boxColor = newColor; // Update the global box color
      const boxes = document.querySelectorAll(".box");
      boxes.forEach((box) => {
        box.style.backgroundColor = boxColor; // Update existing boxes' color
      });
      colorInput.value = ""; // Clear the input field
    }
  });

  // Step 5: Function to add a new box
  function addNewBox() {
    const box = document.createElement("div");
    box.className = "box";
    box.style.backgroundColor = boxColor;
    box.textContent = boxCounter; // Display the ID as content
    box.dataset.id = boxCounter; // Store the ID in a data attribute
    boxCounter++; // Increment the counter for unique IDs
    boxContainer.appendChild(box);
  }

  // Step 6: Add a new box when the "New Box" button is clicked
  newBoxButton.addEventListener("click", addNewBox);

  // Step 7: Remove a box on double-click
  document.addEventListener("dblclick", (event) => {
    if (event.target.classList.contains("box")) {
      event.target.remove();
    }
  });

  // Step 8: Display a box's page coordinates on hover
  document.addEventListener("mouseover", (event) => {
    if (event.target.classList.contains("box")) {
      const rect = event.target.getBoundingClientRect();
      const coordinates = `(${Math.round(rect.left)}, ${Math.round(rect.top)})`;
      event.target.textContent = coordinates;
    }
  });

  // Step 9: Display a box's ID back when the mouse leaves
  document.addEventListener("mouseout", (event) => {
    if (event.target.classList.contains("box")) {
      event.target.textContent = event.target.dataset.id;
    }
  });

  // Step 10: Create a new box when the "N" key is pressed
  document.addEventListener("keydown", (event) => {
    if (
      (event.key === "N" || event.key === "n") &&
      document.activeElement !== colorInput
    ) {
      addNewBox();
    }
  });
});
```

Explanation of Key Steps:

1. `DOMContentLoaded` **Listener**:
    - Ensures that the script runs after the DOM is fully loaded.
2. **Global Variables**:
    - `boxColor`: Stores the current color for boxes.
    - `boxCounter`: Keeps track of unique IDs for each new box.
3. **Form Submission**:
    - Updates `boxColor` and applies it to all existing boxes.
4. **Adding New Boxes**:
    - Creates a `<div>` element with appropriate styles, content, and attributes.
5. **Removing Boxes**:
    - Listens for double-click events and removes the target element if it's a `.box`.
6. **Displaying Coordinates**:
    - Uses `getBoundingClientRect()` to get page coordinates of a hovered element.
7. **Restoring Box ID**:
    - Retrieves and displays the stored ID from `data-id` when the mouse leaves.
8. **Keyboard Shortcut**:
    - Listens for `N` key presses and ensures it doesn't interfere with typing in the input field.

Notes:

- The script ensures that all interactions are intuitive and responsive.
- The use of `data-id` allows easy retrieval of IDs without relying on inner text, which can change dynamically.
- The `keydown` logic prevents unintended behavior when typing in the input field.

This implementation meets all requirements and provides a robust solution!