Here's the full algorithm explanation for each function:

1. loadTodos()

- **Purpose**: Load todos from the browser's local storage.
- **Steps**:
    1. Attempt to retrieve the stored todos from localStorage.
    2. If the todos exist, parse them from a JSON string to a JavaScript object.
    3. If no todos are found, return a default object { "todoList": [] }.
    4. Log the todos for debugging purposes.
    5. Return the parsed or default todos object.

---

2. addTodoToLocalStorage(todo)

- **Purpose**: Add a new todo to local storage.
- **Steps**:
    1. Call loadTodos() to get the current list of todos.
    2. Add the new todo to the todoList array.
    3. Convert the updated todos object into a JSON string.
    4. Save the JSON string back to localStorage under the key "todos".

---

3. executeFilterAction(event)

- **Purpose**: Apply a filter to display todos based on the selected type (all, pending, or completed).
- **Steps**:
    1. Get the button that triggered the event.
    2. Retrieve the data-filter attribute value from the clicked button, which determines the type of filter.
    3. Call refreshTodos(), passing the filter type (e.g., all, pending, completed) to update the displayed todos.

---

4. executeEditAction(event)

- **Purpose**: Edit the text of a specific todo item.
- **Steps**:

1. Get the list of todos from the DOM.
2. Identify the specific todo item that contains the clicked "Edit" button.
3. Prompt the user for new text for the todo.
4. If the new text is provided, update the text content of the corresponding HTML element.
5. Call updateTodoInLocalStorage() to update the todo text in localStorage.
6. Call refreshTodos() to reload the list and reflect the change.

---

5. executeDeleteAction(event)

- **Purpose**: Delete a specific todo item.
- **Steps**:
    1. Get the clicked "Delete" button from the event.
    2. Retrieve the id of the todo to be deleted.
    3. Filter the todo list to remove the todo with the matching id.
    4. Save the updated todoList to localStorage.
    5. Call refreshTodos() to refresh the todo display and remove the deleted item.

---

6. executeCompleteAction(event)

- **Purpose**: Toggle the completion status of a todo item (mark as complete/incomplete).
- **Steps**:
    1. Get the clicked "Complete/Incomplete" button from the event.
    2. Retrieve the id of the todo that should be updated.
    3. Find the todo with the matching id and toggle its isCompleted property.
    4. Save the updated todoList back to localStorage.
    5. Call refreshTodos() to refresh the list and update the UI accordingly.

---

7. updateTodoInLocalStorage(id, newText)

- **Purpose**: Update the text of a specific todo in localStorage.

- **Steps**:
    1. Call loadTodos() to retrieve the current list of todos.
    2. Find the todo with the matching id and update its text property with the newText.
    3. Save the updated todos back to localStorage.

---

8. appendTodoInHtml(todoItem)

- **Purpose**: Append a single todo item to the HTML (DOM).
- **Steps**:
    1. Create a new li element representing a todo item and set its id to the todo's ID.
    2. Create a div to display the todo text.
    3. Add the todoItem.text as the content of the div.
    4. If the todo is completed (isCompleted is true), apply a CSS class to visually mark it as complete.
    5. Create a wrapper div to hold buttons for editing, deleting, and toggling completion status.
    6. Create the buttons and add event listeners for the respective actions (edit, delete, complete).
    7. Append the buttons to the wrapper and the wrapper to the li element.
    8. Finally, append the li element to the todoList in the DOM.

---

9. refreshTodos(filter = "all")

- **Purpose**: Refresh the displayed todos based on a filter (all, pending, completed).
- **Steps**:
    1. Clear the current todoList from the DOM.
    2. Call loadTodos() to retrieve the current todos from localStorage.
    3. Based on the provided filter, loop through the todos:
        - If filter is "all", display all todos.
        - If filter is "pending", display only those where isCompleted is false.

        - If filter is "completed", display only those where isCompleted is true.

    4. For each todo to be displayed, call appendTodoInHtml() to render it in the DOM.

---

10. addNewTodo()

- **Purpose**: Add a new todo and refresh the displayed list.
- **Steps**:
    1. Retrieve the text from the todo input field.
    2. If the input is empty, show an alert asking the user to write something.
    3. If valid text is provided:
        - Create a new todo object with a unique id, text, and isCompleted set to false.
        - Call addTodoToLocalStorage() to store the new todo.
        - Clear the input field.
        - Call refreshTodos() to refresh the list and display the newly added todo.

---

11. resetTodosHtml()

- **Purpose**: Clear the todo list from the HTML (reset the displayed list).
- **Steps**:
    1. Retrieve the todoList element from the DOM.
    2. Set its inner HTML to an empty string, effectively clearing all rendered todos.

---

12. DOMContentLoaded Event Listener

- **Purpose**: Initialize the app after the page has loaded.
- **Steps**:
    1. Add event listeners to filter buttons for filtering the todo list.
    2. Add an event listener to the "Add Todo" button to handle adding new todos.
    3. Add a keypress event listener for the "Enter" key to submit a todo when the user presses Enter.

4. Call refreshTodos() to load and display existing todos from localStorage on page load.