---

1. DOM Element Selection
```javascript
let text = document.querySelector(".row input");
let btn = document.querySelector(".row button");
let list = document.querySelector(".list-container ul");
let img = document.querySelector(".row img");
```
- **`text`**: Selects the input field where the user enters a new task.
- **`btn`**: Selects the button that adds the task to the list.
- **`list`**: Selects the `<ul>` element where the tasks will be displayed.
- **`img`**: Selects an image element (not used in the code, possibly for future functionality).

---

2. Adding a Task
```javascript
btn.addEventListener("click", () => {
    try {
        let li = document.createElement("li");
        li.textContent = text.value;
        list.appendChild(li);
        let span = document.createElement("span");
        span.textContent = "x";
        li.appendChild(span);
        text.value = "";
        saveData();
    } catch (error) {
        console.error("Error adding task:", error);
    }
});
```
- When the user clicks the button:
  1. A new `<li>` element is created to represent the task.
  2. The text entered in the input field (`text.value`) is set as the content of the `<li>`.
  3. The `<li>` is appended to the `<ul>` (task list).
  4. A `<span>` element with the text "x" is created and appended to the `<li>`. This will act as a delete button.
  5. The input field is cleared (`text.value = ""`).
  6. The `saveData` function is called to save the updated task list to local storage.
- If an error occurs (e.g., invalid input), it is caught and logged to the console.

---

3. Handling Task Interactions
```javascript
list.addEventListener("click", (e) => {
    try {
        if (e.target.tagName === "LI") {
            e.target.classList.toggle("checked");
            saveData();
        } else if (e.target.tagName === "SPAN") {
            e.target.parentElement.remove();
            saveData();
        }
    } catch (error) {
        console.error("Error updating task:", error);
```

```
    }
});
```

- When the user clicks on the task list:
  1. If the clicked element is an `<li>` (task):
     - The `checked` class is toggled on the `<li>`. This can be used to visually mark the task as completed (e.g., by applying a strikethrough style via CSS).
     - The `saveData` function is called to save the updated task list.
  2. If the clicked element is a `<span>` (delete button):
     - The parent `<li>` (task) is removed from the list.
     - The `saveData` function is called to save the updated task list.
- If an error occurs (e.g., DOM manipulation issue), it is caught and logged to the console.

---

4. Saving Data to Local Storage
```javascript
function saveData(){
    try {
        localStorage.setItem("data", list.innerHTML);
    } catch (error) {
        console.error("Error saving data:", error);
    }
}
```
- This function saves the current state of the task list (`list.innerHTML`) to the browser's **local storage** under the key `"data"`.
- If an error occurs (e.g., local storage is full), it is caught and logged to the console.

---

5. Loading Tasks from Local Storage
```javascript
function showTask(){
    try {
        list.innerHTML = localStorage.getItem("data");
    } catch (error) {
        console.error("Error loading tasks:", error);
    }
}
```
- This function retrieves the saved task list from local storage (`localStorage.getItem("data")`) and sets it as the inner HTML of the `<ul>` element (`list.innerHTML`).
- If an error occurs (e.g., no data in local storage), it is caught and logged to the console.

---

6. Initializing the Application
```javascript
showTask();
```
- When the page loads, the `showTask` function is called to load any saved tasks from local storage and display them in the task list.

How It Works
1. The user enters a task in the input field and clicks the button to add it to the list.
2. The task is displayed as an `<li>` element with a delete button (`<span>`).
3. The user can:

   - Mark a task as completed by clicking on it (toggles the `checked` class).
   - Delete a task by clicking the "x" button.
4. All changes to the task list are saved to local storage.
5. When the page is refreshed, the saved tasks are loaded from local storage and displayed.

Example Usage
Adding a Task:
- User types "Buy groceries" in the input field and clicks the button.
- The task "Buy groceries" is added to the list with a delete button ("x").

Marking a Task as Completed:
- User clicks on the task "Buy groceries".
- The task is visually marked as completed (e.g., strikethrough text).

Deleting a Task:
- User clicks the "x" button next to "Buy groceries".
- The task is removed from the list.

Refreshing the Page:
- All tasks are still displayed because they are saved in local storage.

Improvements
1. **Input Validation**: Ensure the user cannot add empty tasks.
2. **Edit Functionality**: Allow users to edit existing tasks.
3. **Styling**: Add CSS to make the UI more visually appealing (e.g., strikethrough for completed tasks).
4. **Drag-and-Drop**: Allow users to reorder tasks by dragging and dropping.
5. **Error Handling**: Provide user-friendly error messages (e.g., "Task cannot be empty").