Lab — JavaScript Review: Task 1  
ToDo List App (Plain JavaScript)

# What you’ll build

A tiny ToDo app with an input and a button. Three tasks are already visible when the page loads (hard-coded in HTML). Clicking 'Add task' appends another list item. No arrays or frameworks needed.

# Languages & tools

* HTML — structure: headings, input, button, list.
* CSS — presentation: spacing, fonts, layout (optional but included).
* JavaScript — behaviour: read the input, react to clicks/keys, create <li> elements and append them to the list.
* Browser DevTools — open Console to see errors and test small snippets.

# Step-by-step

1. Create an HTML page skeleton with <!doctype html>, <html>, <head> and <body>.
2. Add a heading and a sentence that states what the app does.
3. Add a header bar with <input id="new-task"> and <button id="add-btn">.
4. Add <ul id="todo-list"> and put three <li> items inside it.
5. Write a <script> that selects the input, button and list using document.getElementById.
6. Attach a click handler to the button that reads input.value.trim(), guards against blank, creates a new <li>, assigns .textContent, appends it to the list, clears the input and focuses it again.
7. Optional: add Enter-key support by listening to keyup on the input and, when e.key === 'Enter', call button.click().

# Complete example (copy & run)

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>ToDo List — Plain JS (Task 1)</title>

<style>

body { font-family: system-ui, -apple-system, Segoe UI, Roboto, sans-serif; max-width: 720px; margin: 40px auto; padding: 0 16px; }

header { display: flex; gap: 8px; }

input[type=text] { flex: 1; padding: 8px; }

button { padding: 8px 12px; cursor: pointer; }

ul { padding-left: 20px; }

li { padding: 6px 0; }

</style>

</head>

<body>

<h1>ToDo List — Plain JS (Task 1)</h1>

<p>Initial items are directly in the HTML. Clicking "Add task" appends a new <code>&lt;li&gt;</code> without using an array.</p>

<header>

<input id="new-task" type="text" placeholder="New task..." />

<button id="add-btn">Add task</button>

</header>

<ul id="todo-list">

<li>Task 1</li>

<li>Task 2</li>

<li>Task 3</li>

</ul>

<script>

const input = document.getElementById('new-task');

const button = document.getElementById('add-btn');

const list = document.getElementById('todo-list');

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

const text = input.value.trim();

if (!text) return;

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

li.textContent = text;

list.appendChild(li);

input.value = "";

input.focus();

});

input.addEventListener('keyup', (e) => {

if (e.key === 'Enter') button.click();

});

</script>

</body>

</html>

# Why this design?

* DOM is the source of truth in this task: we are not storing data anywhere; we manipulate the live document tree.
* We guard against blank input with trim() + early return, so you learn basic input validation.

# Assessment checklist

* Page shows 3 tasks on load.
* Button click appends a new <li>.
* No blank tasks are added.