1. Sum of Two Numbers

Objective: Variables, number conversion, output

Ask the user for two numbers and display their sum.

```
const num1 = Number(prompt("Enter first number:"));
const num2 = Number(prompt("Enter second number:"));
alert("The sum is: " + (num1 + num2));
```

Why it works:

`prompt()` returns a string, so `Number()` converts it. The numbers are added, and the result is shown with `alert()`.

---

2. Even or Odd?

Objective: Conditionals, modulo operator

```
const value = Number(prompt("Enter a number:"));
if (value % 2 === 0) console.log("Even");
else console.log("Odd");
```

Why it works: `%` 2 gives the remainder—0 means even.

---

3. Countdown

Objective: Loops, backwards iteration

```
const n = Number(prompt("Enter a number:"));
for (let i = n; i >= 1; i--) {
  console.log(i);
}
```

Why it works:

A `for` loop counts down until it reaches 1.

---

4. Maximum of an Array

Objective: Arrays, tracking max values

```
const nums = [3, 17, 9, 42, 5];
let max = nums[0];

for (let i = 1; i < nums.length; i++) {
  if (nums[i] > max) max = nums[i];
```

```
}

console.log(max);
```

---

5. Greeting Function

Objective: Defining and calling functions

```
function greet(name) {
  console.log("Hello, " + name);
}

greet("Lars");
```

---

6. Change Page Text (DOM)

Objective: DOM selection & modification

```
const heading = document.getElementById("title");
heading.textContent = "Updated via JavaScript!";
```

---

7. Button Click Counter

Objective: Events, state management

```
let count = 0;
button.addEventListener("click", () => {
  count++;
  countSpan.textContent = count;
});
```

---

8. Shopping Cart Total

Objective: Objects, arrays, totals

```
const cart = [
  { name: "Book", price: 12.99 },
  { name: "Headphones", price: 29.99 }
];
```

```
let total = 0;
for (const item of cart) total += item.price;
console.log(total);
```

---

9. Convert Celsius to Fahrenheit (map)

Objective: Higher-order functions

```
const c = [0, 10, 20];
const f = c.map(n => n * 9/5 + 32);
console.log(f);
```

---

10. Simple Timer

Objective: setInterval, clearInterval

```
let secs = 0;
const timer = setInterval(() => {
  console.log(secs++);
  if (secs > 10) clearInterval(timer);
}, 1000);
```

💡 Exercises 11–20 — Intermediate JavaScript & DOM Skills

---

11. Reverse a String

Objective: Strings, loops

```
const input = prompt("Enter a word:");
let reversed = "";

for (let i = input.length - 1; i >= 0; i--) {
  reversed += input[i];
}

console.log(reversed);
```

---

12. Word Counter

Objective: Split, trim, whitespace handling

```
const text = prompt("Enter text:").trim();
const words = text ? text.split(/\s+/) : [];
console.log(words.length);
```

---

13. Number Guessing Game

Objective: Random numbers, loops, conditionals

```
const secret = Math.floor(Math.random() * 20) + 1;
let guess;

while (guess !== secret) {
  guess = Number(prompt("Guess the number:"));
  if (guess < secret) alert("Too low");
  else if (guess > secret) alert("Too high");
  else alert("Correct!");
}
```

---

14. Filter Even Numbers (filter)

Objective: Array filtering

```
const nums = [1, 4, 7, 10];
const evens = nums.filter(n => n % 2 === 0);
console.log(evens);
```

---

15. Sum with reduce

Objective: Functional patterns

```
const nums = [5, 10, 15];
const total = nums.reduce((acc, n) => acc + n, 0);
console.log(total);
```

---

16. Unique Values with Set

Objective: Set, deduplicating arrays

```
const vals = [1, 2, 2, 4, 4];
const unique = [...new Set(vals)];
console.log(unique);
```

---

17. Generate List Items (DOM)

Objective: Create & insert elements

```
const tasks = ["Learn JS", "Practice DOM"];
const ul = document.getElementById("taskList");

tasks.forEach(t => {
  const li = document.createElement("li");
  li.textContent = t;
  ul.appendChild(li);
});
```

---

18. Show / Hide Toggle

Objective: classList.toggle

```
button.addEventListener("click", () => {
  details.classList.toggle("hidden");
});
```

---

19. Simple Form Validation

Objective: submit event, preventing submission

```
form.addEventListener("submit", e => {
  e.preventDefault();
  if (!email.value || !password.value) {
    msg.textContent = "Fill in all fields.";
  } else {
    msg.textContent = "Success!";
  }
});
```

20. Persistent Click Counter (localStorage)

Objective: Saving data in the browser

```
let total = Number(localStorage.getItem("count")) || 0;
totalSpan.textContent = total;

button.addEventListener("click", () => {
  total++;
  totalSpan.textContent = total;
  localStorage.setItem("count", total);
});
```

21) FizzBuzz (Loops + Conditionals) — Console

Goal: Print numbers 1–100. For multiples of 3 print “Fizz”, multiples of 5 print “Buzz”, both → “FizzBuzz”.

```
for (let i = 1; i <= 100; i++) {
  if (i % 15 === 0) {
    console.log("FizzBuzz");
  } else if (i % 3 === 0) {
    console.log("Fizz");
  } else if (i % 5 === 0) {
    console.log("Buzz");
  } else {
    console.log(i);
  }
}
```

Explanation

- % checks divisibility via remainder.
- Check 15 first so numbers like 30 don’t get caught by the 3 check early.
- else prints the number when no rule applies.

22) Palindrome Checker (Strings) — Console

Goal: Check if a word reads the same forward and backward.

```
function isPalindrome(word) {
  const cleaned = word.toLowerCase().replace(/[^a-z0-9]/g, "");
  const reversed = cleaned.split("").reverse().join("");
  return cleaned === reversed;
}

console.log(isPalindrome("Racecar"));       // true
console.log(isPalindrome("hello"));         // false
console.log(isPalindrome("A man, a plan!"));// true-ish based on
cleaning
```

Explanation

- `toLowerCase()` makes comparison case-insensitive.
- `replace(/[^a-z0-9]/g, "")` removes non-alphanumeric characters.
- `split("")` → array of chars, `reverse()`, then `join("")` back to a string.

---

23) Factorial (Loops) — Console

Goal: Compute `n!` (e.g., 5! = 120).

```
function factorial(n) {
  if (!Number.isInteger(n) || n < 0) return null;

  let result = 1;
  for (let i = 2; i <= n; i++) {
    result *= i;
  }
  return result;
}

console.log(factorial(5));  // 120
console.log(factorial(0));  // 1
console.log(factorial(-2)); // null
```

Explanation

- Validates input (must be a non-negative integer).
- Starts at 1 and multiplies by every number up to n.
- `0!` is defined as 1, which this code naturally produces.

---

24) Fibonacci Sequence (Array) — Console

Goal: Generate the first n Fibonacci numbers.

```
function fibonacci(n) {
  if (!Number.isInteger(n) || n <= 0) return [];
  if (n === 1) return [0];

  const seq = [0, 1];
  while (seq.length < n) {
    const next = seq[seq.length - 1] + seq[seq.length - 2];
    seq.push(next);
  }
  return seq;
}

console.log(fibonacci(10));
```

Explanation

- Starts with `[0, 1]`.
- Each next value is the sum of the last two.
- Uses `while` until the array reaches length n.

---

25) Count Character Frequency (Objects) — Console

Goal: Count how many times each character appears in a string.

```
function charFrequency(text) {
  const freq = {};
```

```
  for (const ch of text) {
    freq[ch] = (freq[ch] || 0) + 1;
  }
  return freq;
}

console.log(charFrequency("banana"));
```

Explanation

- `freq` is an object where keys are characters.
- `freq[ch] || 0` defaults to 0 if the key doesn't exist.
- Adds 1 for each occurrence.

---

26) Sort Numbers Without `.sort()` (Algorithm) — Console

Goal: Sort an array ascending using Bubble Sort.

```
function bubbleSort(arr) {
  const a = [...arr]; // copy so we don't mutate input
  for (let i = 0; i < a.length - 1; i++) {
    for (let j = 0; j < a.length - 1 - i; j++) {
      if (a[j] > a[j + 1]) {
        [a[j], a[j + 1]] = [a[j + 1], a[j]]; // swap
      }
    }
  }
  return a;
}

console.log(bubbleSort([5, 3, 8, 1, 2]));
```

Explanation

- Outer loop controls passes.

- Inner loop compares adjacent pairs and swaps if out of order.
- `- i` optimization: after each pass, the largest value is already at the end.

---

27) Remove Duplicates Manually (No Set) — Console

Goal: Return unique values in order.

```
function uniqueValues(arr) {
  const result = [];
  for (const item of arr) {
    if (!result.includes(item)) {
      result.push(item);
    }
  }
  return result;
}

console.log(uniqueValues([1, 2, 2, 3, 1, 4]));
```

Explanation

- Uses `result.includes(item)` to check if already added.
- Preserves first-seen order.
- Not the fastest for huge arrays, but perfect for learning.

---

28) Flatten One Level of Nested Arrays — Console

Goal: Turn `[1,[2,3],[4],5]` into `[1,2,3,4,5]`.

```
function flattenOneLevel(arr) {
  const out = [];
  for (const item of arr) {
    if (Array.isArray(item)) out.push(...item);
    else out.push(item);
  }
  return out;
```

```
}

console.log(flattenOneLevel([1, [2, 3], [4], 5]));
```

Explanation

- `Array.isArray` detects nested arrays.
- `...item` spreads array elements into `out`.
- Only flattens one level (by design).

---

29) Group Items by Property — Console

Goal: Group objects by `category`.

```
function groupBy(arr, key) {
  const grouped = {};
  for (const obj of arr) {
    const k = obj[key];
    if (!grouped[k]) grouped[k] = [];
    grouped[k].push(obj);
  }
  return grouped;
}

const products = [
  { name: "Apple", category: "Fruit" },
  { name: "Carrot", category: "Veg" },
  { name: "Banana", category: "Fruit" },
];

console.log(groupBy(products, "category"));
```

Explanation

- Uses an object of arrays.
- Each unique key (like `"Fruit"`) becomes a bucket.

- Push each item into the correct bucket.

---

30) Safe JSON Parse (Try/Catch) — Console

Goal: Parse JSON without crashing your program.

```
function safeJsonParse(str) {
  try {
    return { ok: true, data: JSON.parse(str) };
  } catch (err) {
    return { ok: false, error: err.message };
  }
}

console.log(safeJsonParse('{"a":1}'));
console.log(safeJsonParse("{bad json}"));
```

Explanation

- `JSON.parse` throws if the string isn't valid JSON.
- `try/catch` prevents the error from stopping execution.
- Returns a consistent result object.

---

31) Debounce (Functions + Timers) — Console/Browser

Goal: Only run a function after the user "stops" triggering it for X ms.

```
function debounce(fn, delayMs) {
  let timerId;

  return function (...args) {
    clearTimeout(timerId);
    timerId = setTimeout(() => fn(...args), delayMs);
  };
}

// Example:
```

```
const debouncedLog = debounce((msg) => console.log(msg), 500);
debouncedLog("A");
debouncedLog("B");
debouncedLog("C"); // only "C" logs after 500ms
```

Explanation

- A closure keeps `timerId` between calls.
- Each call clears the previous timeout and sets a new one.
- Only the final call survives long enough to run.

---

32) Throttle (Functions + Timers) — Console/Browser

Goal: Run a function at most once every X ms.

```
function throttle(fn, intervalMs) {
  let lastTime = 0;

  return function (...args) {
    const now = Date.now();
    if (now - lastTime >= intervalMs) {
      lastTime = now;
      fn(...args);
    }
  };
}

// Example:
const throttledLog = throttle(() => console.log("tick"), 1000);
setInterval(() => throttledLog(), 100); // logs about once per
second
```

Explanation

- `lastTime` tracks the last allowed execution time.
- If not enough time passed, calls are ignored.

- Useful for scroll/resize events.

---

33) Build a URL Query String — Console

Goal: Convert an object to ?key=value&....

```
function toQueryString(params) {
  const parts = [];
  for (const key in params) {
    const value = encodeURIComponent(params[key]);
    parts.push(`${encodeURIComponent(key)}=${value}`);
  }
  return parts.length ? "?" + parts.join("&") : "";
}

console.log(toQueryString({ q: "js exercises", page: 2 }));
```

Explanation

- `encodeURIComponent` makes values safe for URLs.
- `parts` becomes an array of `key=value` pairs.
- Joins them with & and adds a leading ?.

---

34) Deep Clone Simple JSON Objects — Console

Goal: Copy nested JSON-safe objects without shared references.

```
function deepCloneJson(obj) {
  return JSON.parse(JSON.stringify(obj));
}

const original = { a: 1, b: { c: 2 } };
const clone = deepCloneJson(original);
clone.b.c = 999;

console.log(original.b.c); // 2
console.log(clone.b.c);    // 999
```

Explanation

- `JSON.stringify` converts to a string.
- `JSON.parse` creates a brand new structure.
- Limitation: won't preserve functions, Dates, Maps, etc. (fine for learning + JSON data).

---

35) Validate Email (Basic Regex) — Console

Goal: Basic email format check.

```
function isEmail(str) {
  return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(str);
}

console.log(isEmail("test@example.com")); // true
console.log(isEmail("bad@email"));        // false
```

Explanation

- This regex checks:
    - something before @
    - something after @
    - at least one dot section
- Not "perfect" for every valid email ever, but solid for beginner validation.

Exercises 36–50 (DOM + Practical Web Skills)

For these, use this HTML template once, then swap scripts per exercise:

```
<!doctype html>
<html>
```

```
  <head>
    <meta charset="utf-8" />
    <title>JS Exercises</title>
    <style>
      body { font-family: Arial, sans-serif; padding: 16px; }
      .hidden { display: none; }
      .error { color: #b00020; }
      .ok { color: #0a7a0a; }
      .box { padding: 12px; border: 1px solid #ddd;
border-radius: 8px; margin: 12px 0; }
    </style>
  </head>
  <body>
    <div id="app"></div>
    <script src="script.js"></script>
  </body>
</html>
```

---

36) Live Character Counter — DOM

Goal: Count characters as the user types.

script.js

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Live Character Counter</h2>
    <textarea id="txt" rows="4" cols="40" placeholder="Type
here..."></textarea>
    <p>Characters: <span id="count">0</span></p>
  </div>
`;

const txt = document.getElementById("txt");
const count = document.getElementById("count");

txt.addEventListener("input", () => {
  count.textContent = txt.value.length;
```

```
});
```

Explanation

- `input` event fires on every change (typing, paste, delete).
- `txt.value.length` gives current character count.
- Updates the DOM span instantly.

---

37) To-Do List Add Items — DOM

Goal: Add items to a list from an input field.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>To-Do List</h2>
    <input id="taskInput" placeholder="New task" />
    <button id="addBtn">Add</button>
    <ul id="list"></ul>
  </div>
`;

const input = document.getElementById("taskInput");
const addBtn = document.getElementById("addBtn");
const list = document.getElementById("list");

addBtn.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();
});
```

Explanation

- `trim()` prevents empty/space-only tasks.
- `createElement("li")` creates new list items.
- Resetting and focusing improves user experience.

---

38) To-Do Remove Items (Event Delegation) — DOM

Goal: Click a list item to remove it.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Click-to-Remove List</h2>
    <input id="taskInput" placeholder="New item" />
    <button id="addBtn">Add</button>
    <ul id="list"></ul>
    <p><em>Tip: click an item to remove it.</em></p>
  </div>
`;

const input = document.getElementById("taskInput");
const addBtn = document.getElementById("addBtn");
const list = document.getElementById("list");

addBtn.addEventListener("click", () => {
  const text = input.value.trim();
  if (!text) return;

  const li = document.createElement("li");
  li.textContent = text;
  list.appendChild(li);

  input.value = "";
});
```

```
// Event delegation: one listener on the parent <ul>
list.addEventListener("click", (e) => {
  if (e.target.tagName === "LI") {
    e.target.remove();
  }
});
```

Explanation

- Instead of adding a click handler to every `<li>`, we add one to the `<ul>`.
- `e.target` is the clicked element.
- `.remove()` deletes it from the DOM.

---

39) Tabs UI (Switch Sections) — DOM

Goal: Click buttons to show different content panels.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Tabs</h2>
    <button data-tab="one">Tab 1</button>
    <button data-tab="two">Tab 2</button>
    <button data-tab="three">Tab 3</button>

    <div id="one" class="panel">Content for Tab 1</div>
    <div id="two" class="panel hidden">Content for Tab 2</div>
    <div id="three" class="panel hidden">Content for Tab 3</div>
  </div>
`;

function showTab(id) {
  document.querySelectorAll(".panel").forEach(p =>
p.classList.add("hidden"));
  document.getElementById(id).classList.remove("hidden");
}
```

```
document.querySelectorAll("button[data-tab]").forEach(btn => {
  btn.addEventListener("click", () => showTab(btn.dataset.tab));
});
```

Explanation

- Panels share class `panel`; we hide all, then reveal the chosen one.
- `dataset.tab` reads `data-tab="..."`.
- This pattern scales to many tabs.

---

40) Modal Popup (Open/Close) — DOM

Goal: Open a modal overlay and close it.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Modal</h2>
    <button id="open">Open Modal</button>
  </div>

  <div id="overlay" class="hidden" style="
    position:fixed; inset:0; background:rgba(0,0,0,.5);
    display:flex; align-items:center; justify-content:center;">
    <div style="background:#fff; padding:16px;
border-radius:10px; width:300px;">
      <h3>Modal Title</h3>
      <p>This is a simple modal.</p>
      <button id="close">Close</button>
    </div>
  </div>
`;

const overlay = document.getElementById("overlay");

document.getElementById("open").addEventListener("click", () =>
{
```

```
  overlay.classList.remove("hidden");
});

document.getElementById("close").addEventListener("click", () =>
{
  overlay.classList.add("hidden");
});

// Close if user clicks outside the modal box
overlay.addEventListener("click", (e) => {
  if (e.target === overlay) overlay.classList.add("hidden");
});
```

Explanation

- Overlay covers the screen and centers the modal content.
- Clicking the overlay (not the inner box) closes it.
- Uses `.hidden` to control visibility.

---

41) Form Validation with Inline Errors — DOM

Goal: Validate name + email and show errors.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Signup</h2>
    <form id="form">
      <div>
        <label>Name</label><br/>
        <input id="name" />
        <div id="nameErr" class="error"></div>
      </div>
      <div style="margin-top:8px;">
        <label>Email</label><br/>
        <input id="email" />
        <div id="emailErr" class="error"></div>
```

```
      </div>
      <button style="margin-top:10px;">Submit</button>
      <p id="status"></p>
    </form>
  </div>
`;

const form = document.getElementById("form");
const nameEl = document.getElementById("name");
const emailEl = document.getElementById("email");
const nameErr = document.getElementById("nameErr");
const emailErr = document.getElementById("emailErr");
const status = document.getElementById("status");

function isEmail(str) {
  return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(str);
}

form.addEventListener("submit", (e) => {
  e.preventDefault();

  nameErr.textContent = "";
  emailErr.textContent = "";
  status.textContent = "";

  const name = nameEl.value.trim();
  const email = emailEl.value.trim();

  let ok = true;

  if (name.length < 2) {
    nameErr.textContent = "Name must be at least 2 characters.";
    ok = false;
  }
  if (!isEmail(email)) {
```

```
    emailErr.textContent = "Please enter a valid email
address.";
    ok = false;
  }

  status.textContent = ok ? "✅ Submitted (demo)!" : "❌ Fix
errors above.";
  status.className = ok ? "ok" : "error";
});
```

Explanation

- Prevents page reload via `preventDefault()`.
- Clears old errors each submit.
- Uses ok flag to track validation results.
- Provides immediate, targeted feedback.

---

42) Theme Toggle (Dark/Light) — DOM

Goal: Switch page theme with a button.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Theme Toggle</h2>
    <button id="toggle">Toggle Theme</button>
    <p>Click the button to switch styles.</p>
  </div>
`;

let dark = false;

document.getElementById("toggle").addEventListener("click", ()
=> {
  dark = !dark;
```

```
  document.body.style.background = dark ? "#111" : "#fff";
  document.body.style.color = dark ? "#fff" : "#111";
});
```

Explanation

- Tracks state with `dark`.
- Updates inline styles on the body.
- Simple pattern that can later be upgraded to CSS classes.

---

43) Random Quote Generator (Array + DOM) — DOM

Goal: Show a random quote each click.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Random Quote</h2>
    <button id="new">New Quote</button>
    <p id="quote" style="margin-top:10px;"></p>
  </div>
`;

const quotes = [
  "Small steps, daily.",
  "Make it work, then make it better.",
  "Practice beats theory.",
  "Debugging is learning in disguise."
];

const quoteEl = document.getElementById("quote");

function showRandomQuote() {
  const idx = Math.floor(Math.random() * quotes.length);
  quoteEl.textContent = quotes[idx];
}
```

```
document.getElementById("new").addEventListener("click",
showRandomQuote);
showRandomQuote();
```

Explanation

- Random index: `Math.floor(Math.random() * length)`.
- Updates quote text.
- Calls once at start so the page isn't empty.

---

44) Simple Stopwatch (Start/Stop/Reset) — DOM

Goal: Build a stopwatch with interval control.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Stopwatch</h2>
    <h3 id="time">0.0</h3>
    <button id="start">Start</button>
    <button id="stop">Stop</button>
    <button id="reset">Reset</button>
  </div>
`;

const timeEl = document.getElementById("time");
let t = 0;              // tenths of a second
let timerId = null;

function render() {
  timeEl.textContent = (t / 10).toFixed(1);
}

document.getElementById("start").addEventListener("click", () =>
{
  if (timerId !== null) return; // already running
  timerId = setInterval(() => {
```

```
    t++;
    render();
  }, 100);
});

document.getElementById("stop").addEventListener("click", () =>
{
  clearInterval(timerId);
  timerId = null;
});

document.getElementById("reset").addEventListener("click", () =>
{
  t = 0;
  render();
});
render();
```

Explanation

- Uses tenths of seconds (`100ms`) for a smoother display.
- Stores interval id in `timerId` so it can be stopped.
- Prevents multiple intervals with the `if (timerId !== null)` guard.

---

45) Countdown Timer (Input + Interval) — DOM

Goal: User enters seconds; countdown to 0.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Countdown</h2>
    <input id="secs" type="number" min="1" placeholder="Seconds"
/>
    <button id="go">Start</button>
    <p id="out"></p>
  </div>
```

```
`;

const secsInput = document.getElementById("secs");
const out = document.getElementById("out");
let id = null;

document.getElementById("go").addEventListener("click", () => {
  let remaining = Number(secsInput.value);

  if (!Number.isFinite(remaining) || remaining <= 0) {
    out.textContent = "Enter a positive number.";
    return;
  }

  clearInterval(id);
  out.textContent = `Remaining: ${remaining}s`;

  id = setInterval(() => {
    remaining--;
    out.textContent = `Remaining: ${remaining}s`;

    if (remaining <= 0) {
      clearInterval(id);
      out.textContent = "✅ Done!";
    }
  }, 1000);
});
```

Explanation

- Validates input and converts to number.
- Clears any existing countdown to avoid multiple timers.
- Decrements once per second until 0, then stops interval.

---

46) LocalStorage Notes App — DOM + Storage

Goal: Save notes in the browser.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Notes (Saved in Browser)</h2>
    <textarea id="note" rows="5" cols="40" placeholder="Write
notes..."></textarea>
    <div>
      <button id="save">Save</button>
      <button id="clear">Clear</button>
    </div>
    <p id="msg"></p>
  </div>
`;

const note = document.getElementById("note");
const msg = document.getElementById("msg");

note.value = localStorage.getItem("notes") || "";

document.getElementById("save").addEventListener("click", () =>
{
  localStorage.setItem("notes", note.value);
  msg.textContent = "✅ Saved!";
  msg.className = "ok";
});

document.getElementById("clear").addEventListener("click", () =>
{
  note.value = "";
  localStorage.removeItem("notes");
  msg.textContent = "🗑️ Cleared!";
  msg.className = "";
});
```

Explanation

- Loads saved notes on startup.
- Saves text to `localStorage` under a key (`"notes"`).
- Clear removes both UI text and stored data.

---

47) Simple Search Filter (List Filtering) — DOM

Goal: Filter visible items based on a search box.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Search Filter</h2>
    <input id="search" placeholder="Search..." />
    <ul id="items"></ul>
  </div>
`;

const data = ["JavaScript", "HTML", "CSS", "DOM", "Events",
"Arrays", "Objects"];
const ul = document.getElementById("items");
const search = document.getElementById("search");

function render(list) {
  ul.innerHTML = "";
  for (const item of list) {
    const li = document.createElement("li");
    li.textContent = item;
    ul.appendChild(li);
  }
}

search.addEventListener("input", () => {
  const q = search.value.toLowerCase().trim();
  const filtered = data.filter(x =>
x.toLowerCase().includes(q));
```

```
  render(filtered);
});

render(data);
```

Explanation

- Renders from an array to the DOM.
- On each input:
    - Normalize query to lowercase.
    - Filter items by `includes`.
    - Re-render the list with matching results.

---

48) Keyboard Shortcut (Ctrl/Cmd + K) — DOM

Goal: Open a “search” UI using keyboard shortcuts.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Keyboard Shortcut</h2>
    <p>Press <strong>Ctrl+K</strong> (Windows) or
<strong>Cmd+K</strong> (Mac).</p>
    <input id="search" class="hidden" placeholder="Type to
search..." />
  </div>
`;

const search = document.getElementById("search");

document.addEventListener("keydown", (e) => {
  const isMac =
navigator.platform.toLowerCase().includes("mac");
  const combo = (isMac && e.metaKey && e.key.toLowerCase() ===
"k")
```

```
            || (!isMac && e.ctrlKey && e.key.toLowerCase() ===
"k");

  if (combo) {
    e.preventDefault();
    search.classList.remove("hidden");
    search.focus();
  }

  if (e.key === "Escape") {
    search.classList.add("hidden");
    search.value = "";
  }
});
```

Explanation

- Listens globally on `document`.
- Detects Ctrl+K or Cmd+K.
- `preventDefault()` stops browser's default search behavior.
- Escape hides and clears the input.

---

49) Fetch JSON (Async/Await) — Browser

Goal: Load data from an API and display it.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Fetch Demo</h2>
    <button id="load">Load Data</button>
    <pre id="out" style="white-space:pre-wrap;"></pre>
  </div>
`;

const out = document.getElementById("out");
```

```
document.getElementById("load").addEventListener("click", async
() => {
  out.textContent = "Loading...";

  try {
    const res = await
fetch("https://jsonplaceholder.typicode.com/todos/1");
    if (!res.ok) throw new Error("HTTP " + res.status);

    const data = await res.json();
    out.textContent = JSON.stringify(data, null, 2);
  } catch (err) {
    out.textContent = "Error: " + err.message;
  }
});
```

Explanation

- `fetch(url)` returns a Promise for the response.
- `await res.json()` parses response body as JSON.
- `try/catch` handles network errors and failed status codes.
- Displays formatted JSON using `JSON.stringify(..., null, 2)`.

---

50) Simple Pagination (Slice + Render) — DOM

Goal: Show items 5 at a time with Next/Prev.

```
document.getElementById("app").innerHTML = `
  <div class="box">
    <h2>Pagination</h2>
    <ul id="list"></ul>
    <button id="prev">Prev</button>
    <button id="next">Next</button>
    <p id="info"></p>
```

```
  </div>
`;

const items = Array.from({ length: 23 }, (_, i) => `Item ${i +
1}`);
const pageSize = 5;
let page = 0;

const list = document.getElementById("list");
const info = document.getElementById("info");

function render() {
  list.innerHTML = "";
  const start = page * pageSize;
  const pageItems = items.slice(start, start + pageSize);

  for (const it of pageItems) {
    const li = document.createElement("li");
    li.textContent = it;
    list.appendChild(li);
  }

  const totalPages = Math.ceil(items.length / pageSize);
  info.textContent = `Page ${page + 1} of ${totalPages}`;
}

document.getElementById("prev").addEventListener("click", () =>
{
  page = Math.max(0, page - 1);
  render();
});

document.getElementById("next").addEventListener("click", () =>
{
  const totalPages = Math.ceil(items.length / pageSize);
```

```
  page = Math.min(totalPages - 1, page + 1);
  render();
});

render();
```

Explanation

- `slice(start, end)` grabs only items for the current page.
- `page * pageSize` computes where the page starts.
- `Math.ceil` computes total pages.
- Prev/Next clamp the page so it doesn't go out of range.