Background Story

It was 7 AM in Pasig City, and Odessa was already sipping her extra-strong barako coffee at the office ☕. As the logistics startup grew, live delivery updates poured in every second: “Pack #123 picked up,” “Pack #456 delivered,” “Pack #789 delayed”—all in one big `<div id="updates"></div>`. Every time a van moved, she had to refresh the page. It felt like watching EDSA traffic through a cracked windshield—chaotic and frustrating.

Her lead, Kuya Ronnie, winked: “Ods, stop hard-coding HTML. Learn to build DOM elements on the fly with `createElement`, `appendChild`, and `removeChild`. Your page will update itself—no reload needed!”

Inspired, Odessa opened her editor. She started with one delivery card:

```
const card = document.createElement("div");
card.className = "update-card";
card.textContent = "Pack #101 is out for delivery";
document.getElementById("updates").appendChild(card);
```

She watched the card appear instantly—like magic. Next, she learned to remove a card when the package was delivered:

```
const container = document.getElementById("updates");
const toRemove = document.querySelector('[data-id="101"]');
container.removeChild(toRemove);
```

Odessa grinned. Her updates panel was becoming a living, breathing feed. Later that afternoon, in a video call with a Manila-based client, she demoed a page where new cards popped in every time a van hit a waypoint, and old cards vanished when delivery was complete. The client clapped—no more manual refreshes!

As the sun set over the Ortigas skyline, Odessa imagined her future startup dashboard: real-time maps, blinking pins, and auto-updating status cards—all built from scratch in JavaScript. With `createElement`, `appendChild`, and `removeChild` mastered, she was ready to build dynamic UIs that solve real-world problems, one delivery update at a time 🚚🐣.

Theory & Lecture Content

Modern JavaScript lets you manipulate the page structure at runtime using DOM methods. Today we cover:

1. document.createElement
2. node.appendChild
3. node.removeChild

1. document.createElement

Use this to create any HTML element in memory.

```
const p = document.createElement("p");
p.textContent = "Hello, Odessa!";
p.className = "greeting"; // set CSS class
p.setAttribute("data-id", "100");
```

Nothing appears on the page yet—this element exists only in JavaScript.

---

2. appendChild

To insert your new element into the page, append it to a parent node.

```
const container = document.getElementById("updates");
container.appendChild(p);
```

Best practice: if you need to add many elements at once, use a `DocumentFragment` to minimize reflows.

```
const frag = document.createDocumentFragment();
for (let i = 0; i < 5; i++) {
  const li = document.createElement("li");
  li.textContent = `Item ${i + 1}`;
  frag.appendChild(li);
}
document.querySelector("ul").appendChild(frag);
```

Reference:
https://developer.mozilla.org/en-US/docs/Web/API/Document/createDocumentFragment

---

3. removeChild

To remove an element from its parent:

```
const container = document.getElementById("updates");
const oldCard = document.querySelector('[data-id="100"]');
if (oldCard) {
  container.removeChild(oldCard);
}
```

You can also clear all children:

```
const container = document.getElementById("updates");
while (container.firstChild) {
```

```
  container.removeChild(container.firstChild);
}
```

Reference:
https://developer.mozilla.org/en-US/docs/Web/API/Node/removeChild

---

Exercises

Exercise 1: Delivery Card Creator

Problem Statement
Write a function that builds a delivery update card from an object and returns the DOM element.

TODOs

- In `exercise1.js`, implement `createDeliveryCard(update)` that:
    1. Creates a `<div>` with class `"card"`.
    2. Sets `data-id` attribute to `update.id`.
    3. Appends two `<p>` children: one for `update.status`, one for `update.location`.
    4. Returns the `<div>`.

Starter Code (exercise1.js)

```
// exercise1.js

/**
 * @typedef {Object} DeliveryUpdate
 * @property {string} id
 * @property {string} status
 * @property {string} location
 */

/**
 * Creates a DOM element for a delivery update.
 * @param {DeliveryUpdate} update
 * @returns {HTMLElement}
 */
export function createDeliveryCard(update) {
  // TODO: implement
}
```

Full Solution (exercise1.js)

```
export function createDeliveryCard(update) {
  const card = document.createElement("div");
  card.className = "card";
  card.setAttribute("data-id", update.id);
```

```
  const statusP = document.createElement("p");
  statusP.textContent = `Status: ${update.status}`;
  card.appendChild(statusP);

  const locP = document.createElement("p");
  locP.textContent = `Location: ${update.location}`;
  card.appendChild(locP);

  return card;
}
```

Demonstration HTML (index1.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Exercise 1</title>
  </head>
  <body>
    <div id="updates"></div>
    <script type="module">
      import { createDeliveryCard } from "./exercise1.js";
      const updates = document.getElementById("updates");
      const data = { id: "A1", status: "Out for delivery", location: "Pasig" };
      const card = createDeliveryCard(data);
      updates.appendChild(card);
    </script>
  </body>
</html>
```

Exercise 2: Remove Delivered Card

Problem Statement
Implement a function to remove a delivery card by its `id`.

TODOs

- In `exercise2.js`, write `removeDeliveryCard(id, container)` that:
  1. Finds child of `container` with attribute `data-id === id`.
  2. If found, calls `container.removeChild(child)`.
  3. Returns `true` if removed, `false` otherwise.

Starter Code (exercise2.js)

```
// exercise2.js

/**
```

```
 * @param {string} id
 * @param {HTMLElement} container
 * @returns {boolean}
 */
export function removeDeliveryCard(id, container) {
  // TODO: implement
}
```

Full Solution (exercise2.js)

```
export function removeDeliveryCard(id, container) {
  const selector = `[data-id="${id}"]`;
  const child = container.querySelector(selector);
  if (child) {
    container.removeChild(child);
    return true;
  }
  return false;
}
```

Demonstration HTML (index2.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Exercise 2</title>
  </head>
  <body>
    <div id="updates">
      <div class="card" data-id="A1"><p>…</p></div>
      <div class="card" data-id="A2"><p>…</p></div>
    </div>
    <script type="module">
      import { removeDeliveryCard } from "./exercise2.js";
      const updates = document.getElementById("updates");
      removeDeliveryCard("A1", updates);
    </script>
  </body>
</html>
```

Exercise 3: Refresh Delivery List

Problem Statement
Write a function to clear existing updates and render a fresh list.

TODOs

- In `exercise3.js`, implement `refreshDeliveryUpdates(updatesArray, container)` that:
  1. Removes *all* children of `container`.
  2. For each item in `updatesArray`, uses `createDeliveryCard` (from `exercise1.js`) to create a card.
  3. Appends each card to `container`.

Starter Code (exercise3.js)

```
// exercise3.js
import { createDeliveryCard } from "./exercise1.js";

/**
 * @param {Array} updatesArray
 * @param {HTMLElement} container
 */
export function refreshDeliveryUpdates(updatesArray, container) {
  // TODO: implement
}
```

Full Solution (exercise3.js)

```
import { createDeliveryCard } from "./exercise1.js";

export function refreshDeliveryUpdates(updatesArray, container) {
  // clear existing
  while (container.firstChild) {
    container.removeChild(container.firstChild);
  }
  // add new
  updatesArray.forEach((update) => {
    const card = createDeliveryCard(update);
    container.appendChild(card);
  });
}
```

Demonstration HTML (index3.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Exercise 3</title>
  </head>
  <body>
    <div id="updates"></div>
    <script type="module">
      import { refreshDeliveryUpdates } from "./exercise3.js";
      const updates = document.getElementById("updates");
```

```
      const data = [
        { id: "B1", status: "Picked up", location: "Makati" },
        { id: "B2", status: "In transit", location: "Mandaluyong" },
      ];
      refreshDeliveryUpdates(data, updates);
    </script>
  </body>
</html>
```

Test Cases

Create these Jest test files. Ensure you run with `--env=jsdom`.

exercise1.test.js

```
/**
 * @jest-environment jsdom
 */
import { createDeliveryCard } from "./exercise1.js";

describe("createDeliveryCard", () => {
  test("builds DOM element with correct structure", () => {
    const data = { id: "X1", status: "Delivered", location: "Quezon City" };
    const card = createDeliveryCard(data);

    expect(card.tagName).toBe("DIV");
    expect(card.className).toBe("card");
    expect(card.getAttribute("data-id")).toBe("X1");

    const paragraphs = card.querySelectorAll("p");
    expect(paragraphs).toHaveLength(2);
    expect(paragraphs[0].textContent).toBe("Status: Delivered");
    expect(paragraphs[1].textContent).toBe("Location: Quezon City");
  });
});
```

exercise2.test.js

```
/**
 * @jest-environment jsdom
 */
import { removeDeliveryCard } from "./exercise2.js";

describe("removeDeliveryCard", () => {
  let container;
  beforeEach(() => {
    container = document.createElement("div");
    const child1 = document.createElement("div");
```

```
    child1.setAttribute("data-id", "C1");
    container.appendChild(child1);
    const child2 = document.createElement("div");
    child2.setAttribute("data-id", "C2");
    container.appendChild(child2);
  });

  test("removes existing child and returns true", () => {
    const result = removeDeliveryCard("C1", container);
    expect(result).toBe(true);
    expect(container.querySelector('[data-id="C1"]')).toBeNull();
    expect(container.children).toHaveLength(1);
  });

  test("returns false when id not found", () => {
    const result = removeDeliveryCard("ZZ", container);
    expect(result).toBe(false);
    expect(container.children).toHaveLength(2);
  });
});
```

exercise3.test.js

```
/**
 * @jest-environment jsdom
 */
import { refreshDeliveryUpdates } from "./exercise3.js";

describe("refreshDeliveryUpdates", () => {
  let container;
  const sampleData = [
    { id: "D1", status: "Loading", location: "Pasay" },
    { id: "D2", status: "En route", location: "Taguig" },
  ];

  beforeEach(() => {
    container = document.createElement("div");
    // pre-populate with dummy child
    const old = document.createElement("div");
    old.setAttribute("data-id", "OLD");
    container.appendChild(old);
  });

  test("clears old children and adds new cards", () => {
    refreshDeliveryUpdates(sampleData, container);
    const cards = container.querySelectorAll(".card");
    expect(cards).toHaveLength(2);
    expect(cards[0].getAttribute("data-id")).toBe("D1");
    expect(cards[1].getAttribute("data-id")).toBe("D2");
  });
});
```

Closing Story

With those functions mastered, Odessa refreshed her live deliveries panel without a single page reload. Every time a new package was scanned, JavaScript created a shiny card in her browser; every completed delivery vanished in an instant. The office monitors now felt alive—each DOM update was a heartbeat in her logistics network.

Kuya Ronnie dropped a message: “Next up: Event Delegation & Custom Events. You’ll make your UI interactive and lean. Ready to catch clicks and fire your own events?”

Odessa smiled, stretching her arms. Dynamic DOM was powerful, but adding interactivity would make her dashboard truly come alive. Tomorrow, she’d learn to handle user clicks, delegate events, and build custom events that flow through her app like data pipelines. Her journey from static HTML to real-time dynamic UIs was just getting started—and the future looked bright, one element at a time. 🚀