Exercise 1 - Drone Dashboard

Exercise 2 - Alert System

Core Concepts

Slide 1: Listening to DOM Events

- How we make the page react to the user in real-time using `.addEventListener()`.
- **The `"input"` event:** Fires immediately on every single keystroke. E.g. for live-search text boxes.
- **The `"change"` event:** Fires when a selection is finalized. E.g. for dropdown menus.

```
// The 'change' event fires the moment a user selects a new dropdown option
document.querySelector("#statusFilter").addEventListener("change", event => {

    // event.target.value holds the exact text of the option they just clicked
    let userChoice = event.target.value;

});
```

Slide 2: Synchronous vs. Asynchronous JavaScript

- **Synchronous:** Code runs line by line. If a task (like downloading data) takes a long time, the entire website freezes (blocking).
- **Asynchronous:** Code starts a task, hands it off to the browser, and moves on to the next line immediately. It handles the data whenever it finishes downloading (non-blocking).

Slide 3: Promises (Waiting for the Future)

- A **Promise** is exactly what it sounds like: the browser promises to give you a result *eventually*.
- **States:** A promise is *Pending* while working, *Fulfilled* when successful, or *Rejected* if it fails.
- **The `.then()` method:** This is how we wait for a Promise to finish. It tells JavaScript: *"When you finally get the data, do this specific thing with it."*

Slide 4: The `fetch()` API

- The modern built-in tool for making network requests (asking a server for data).
- `fetch()` always returns a Promise.
- **The Two-Step Process:** 1. Make the request and wait for the raw response.
  2. Parse that raw response into usable JSON format.

```
// 1. Ask a server for some data
fetch("[https://api.example.com/data](https://api.example.com/data)")
    // 2. Wait for the network, then parse the raw response into JSON
    .then(response => response.json())
    // 3. Take the parsed data and actually do something with it
    .then(data => {
        console.log("Data received:", data);
        renderData(data);
    });
```

Slide 5: Arrow Functions (`=>`)

- A shorter, cleaner syntax for writing functions, especially useful for quick callbacks.
- Standard practice inside `.then()`, `.filter()`, and event listeners.

```
// Traditional approach
let activeDrones = allDrones.filter(function(drone) {
    return drone.status === "active";
});

// Arrow function with curly braces (how we are using it today)
// We drop the word "function" and add the arrow =>
// Because we use curly braces { }, we MUST still write "return"
let activeDrones = allDrones.filter(drone => {
    return drone.status === "active";
});

// Arrow function one-liner
// No curly braces means the return is automatic
let activeDrones = allDrones.filter(drone => drone.status === "active");
```

Slide 6: Array `.filter()`

- A built-in array method that creates a brand **new array** filled with elements that pass a test.
- It loops through every item automatically.
- You provide a condition. If the condition is `true`, the item is kept. If `false`, it is discarded.
- *Crucial concept:* It does **not** modify or destroy your original array.

```
// filter() goes through the array and keeps only what passes the condition.
// It creates a brand new array, leaving the original dataset untouched.
let readyToFly = allDrones.filter(drone => drone.status === "active");
```