JS Week 2 Day 1

🧠 Topic Set: Advanced JavaScript Concepts

Total Duration: 4 Hours
Topics:

1. Events
2. Regular Expressions (Regex)
3. Modules
4. NPM (Node Package Manager)

🕒 4-HOUR STRUCTURED PLAN

Time	Topic	Description
0:00 - 1:00	**Events**	Deep dive into JavaScript events, event flow, event handling, delegation
1:00 - 2:00	**Regex (Regular Expressions)**	Theory + live pattern matching demos + exercises
2:00 - 3:00	**Modules**	ES6 module system, import/export, CommonJS, file organization
3:00 - 4:00	**NPM**	Node.js environment, installing packages, versioning, scripts, creating your own package

---

🧩 1. EVENTS IN JAVASCRIPT (1 HOUR)

🧠 Theory

What is an Event?
An event is any action or occurrence that happens in the browser — like a click, keypress, mouse move, or page load.

JavaScript lets you **respond** to these events, making websites **interactive and dynamic**.

🧱 Common Events

Event Type	Example	Description
Mouse	`onclick`, `ondblclick`, `onmouseover`, `onmouseout`	Triggered by mouse actions
Keyboard	`onkeydown`, `onkeyup`, `onkeypress`	Triggered by keyboard input
Form	`onsubmit`, `onfocus`, `onblur`, `onchange`	Triggered by user interaction with forms
Window	`onload`, `onresize`, `onscroll`	Triggered on page or window events

🧰 3 Ways to Handle Events

1️⃣ Inline Event Handling (Old Style)

```
<button onclick="alert('Button clicked!')">Click Me</button>
```

2️⃣ DOM Property Method

```
<button id="btn">Click Me</button>

<script>
  const btn = document.getElementById("btn");
  btn.onclick = () => alert("Button Clicked!");
</script>
```

3️⃣ addEventListener() — ✅ Modern Way

```
<button id="btn">Click Me</button>

<script>
  const btn = document.getElementById("btn");
  btn.addEventListener("click", () => {
    alert("Modern event handling!");
  });
```

```
</script>
```

⚙️ Event Object

Every event passes an **event object (e)** with details like the event type, target element, coordinates, etc.

```
document.getElementById("btn").addEventListener("click", (e) => {
  console.log("Event type:", e.type);
  console.log("Target element:", e.target);
});
```

🌊 Event Flow — Capturing vs Bubbling

Event Bubbling (default) — event moves from inner → outer elements
Event Capturing — event moves from outer → inner elements

```
<div id="outer">
  Outer
  <div id="inner">Inner</div>
</div>

<script>
  document.getElementById("outer").addEventListener("click", () =>
console.log("Outer clicked!"));
  document.getElementById("inner").addEventListener("click", () =>
console.log("Inner clicked!"));
</script>
```

🧩 *Clicking inner div logs both "Inner" and "Outer" — due to bubbling.*

🧬 Event Delegation

Instead of attaching events to every child element — attach one listener to the parent!

```
<ul id="menu">
```

```
  <li>Home</li>
  <li>About</li>
  <li>Contact</li>
</ul>

<script>
  document.getElementById("menu").addEventListener("click", (e) => {
    if (e.target.tagName === "LI") {
      console.log("You clicked:", e.target.textContent);
    }
  });
</script>
```

✅ Efficient and powerful for dynamic lists or large DOM structures.

💻 Practical Exercises

1. Create a button that changes background color on click.
2. Create an input field that shows the text you type in real-time.
3. Create a counter app (increment/decrement) using event listeners.

🔍 2. REGULAR EXPRESSIONS (1 HOUR)

🧠 Theory

Regular Expressions (**Regex**) are **patterns** used to match character combinations in strings.

You use them for validation, searching, replacing text, etc.

🧱 Syntax

A regex pattern is written between slashes:

```
const pattern = /hello/;
```

You can also use the `RegExp` constructor:

```
const pattern = new RegExp("hello");
```

🔍 Useful Methods

Method	Description	Example
`test()`	Checks if pattern exists	`/js/.test("I love js")` → `true`
`match()`	Returns matched string(s)	`"hello".match(/h/)` → `["h"]`
`replace()`	Replace matched text	`"hi".replace(/h/, "H")` → `"Hi"`

🧩 Common Regex Patterns

Pattern	Meaning	Example
`\d`	Digit (0-9)	`/\d/` matches “5”
`\w`	Word character	`/\w+/` matches “Hello123”

`\s`	Whitespace	`/\s/` matches spaces
`.`	Any character	`/./` matches any single char
`^`	Start of string	`/^Hi/` matches “Hi there”
`$`	End of string	`/end$/` matches “The end”
`[a-z]`	Range	`/[A-Z]/` matches uppercase
`+`	One or more	`/\d+/` matches “123”
`*`	Zero or more	`/\d*/` matches “” or “123”
`{n,m}`	Range of occurrences	`/\d{2,4}/` matches “1234”

🧪 Practical Examples

Validate Email

```
const email = "test@example.com";
const pattern = /^[\w.-]+@[a-zA-Z]+\.[a-zA-Z]{2,}$/;
console.log(pattern.test(email)); // true
```

Validate Phone Number

```
const phone = "9876543210";
console.log(/^[6-9]\d{9}$/.test(phone)); // true
```

Find all numbers

```
const str = "There are 45 apples and 13 bananas";
console.log(str.match(/\d+/g)); // ["45", "13"]
```

💻 Exercises

1. Validate a username (only letters and digits, 5–12 chars).

2. Find all email IDs in a paragraph.

3. Replace all spaces with hyphens in a string.

🧱 3. MODULES (1 HOUR)

🧠 Theory

Modules help **organize JavaScript code** into reusable files — instead of writing everything in one script.

Introduced in **ES6**, modules allow you to use `export` and `import` keywords.

📁 File Example

📄 math.js

```
export const add = (a, b) => a + b;
export const multiply = (a, b) => a * b;
```

📄 app.js

```
import { add, multiply } from "./math.js";

console.log(add(5, 10));      // 15
console.log(multiply(5, 10)); // 50
```

🧩 Default Exports

```
// utils.js
export default function greet(name) {
  return `Hello ${name}`;
}

// app.js
import greet from "./utils.js";
console.log(greet("Piyush"));
```

⚙️ CommonJS (Node.js Style)

Used in backend (Node.js):

```
// math.js
module.exports = {
  add: (a, b) => a + b,
};

// app.js
const { add } = require("./math");
console.log(add(5, 10));
```

💡 Advantages of Modules

- Organized, maintainable code
- Avoids global namespace pollution
- Reusable across projects
- Easy to test

💻 Exercises

1. Create a module for basic math operations.
2. Create a module for string formatting (capitalize, reverse).
3. Import both modules and test them in `index.js`.

📦 4. NPM (Node Package Manager) (1 HOUR)

🧠 Theory

NPM is the **world’s largest software registry** for JavaScript packages.
It’s used to:

- Install libraries
- Manage dependencies
- Run scripts
- Create your own packages

⚙️ Setup

1. Initialize a project:

```
npm init -y
```

2. Install a package:

```
npm install lodash
```

3. Use it:

```
import _ from "lodash";

console.log(_.capitalize("piyush")); // "Piyush"
```

4. Uninstall:

```
npm uninstall lodash
```

🧱 package.json

Stores project info & dependencies.

```
{
  "name": "demo-app",
  "version": "1.0.0",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "lodash": "^4.17.21"
  }
}
```

🧩 Dev Dependencies

Installed for development only:

```
npm install nodemon --save-dev
```

💡 Create Your Own Package

1. Make a folder with your JS file.
2. Add a `package.json`.
3. Publish with:

```
npm login
npm publish
```

💻 Practical Exercises

1. Create a small Node project using `npm init`.
2. Install `chalk` to color your console logs.
3. Use `nodemon` for auto-reload while running.
4. Create your own mini package and test locally using `npm link`.

✅ WRAP-UP SUMMARY

Concept	Key Takeaways
Events	Make pages interactive — understand bubbling, delegation, event listeners
Regex	Use for pattern matching and data validation
Modules	Organize and reuse code easily
NPM	Manage dependencies and project scripts