01: Event Listeners

1. Event Listeners in JavaScript

What is an Event Listener?

A way to run JavaScript code when a user interacts with an element (like clicking, typing, etc.)

Syntax:

```
element.addEventListener("event", callbackFunction);
```

Subtopics:

1.1 Inline `onclick` (Not Recommended)

```
<h1 onclick="console.log('Hey')">Click Me</h1>
```

- Runs JS directly from HTML.
- ⚠️ Not clean for large projects.

1.2 `addEventListener()` (Recommended)

```
btn.addEventListener("click", function () {
  alert("Button clicked");
});
```

- Separates JS and HTML → cleaner code.
- Can add/remove multiple listeners.

1.3 Named Function (for removal)

```
function sayHi() { alert("Hi"); }
btn.addEventListener("click", sayHi);
btn.removeEventListener("click", sayHi);
```

1.4 Arrow Function

```
btn.addEventListener("click", () => console.log("Clicked!"));
```

☑ *1.5 Common Events Table*

Event	When It Fires	Use Case
`click`	On click	Buttons, toggles
`mouseover`	Mouse enters element	Tooltips, hover menus
`mouseout`	Mouse leaves element	Hide tooltips
`keydown`	Key is pressed	Shortcuts, key handling
`keyup`	Key is released	Search, typing
`submit`	Form is submitted	Form validation
`input`	Input is changing	Live typing, search
`change`	Input changed + blurred	Dropdowns, radios, checkboxes

02: Event Object

What is it?

- A special object passed to the event handler function.
- Holds all the info about the event.

Common Properties:

Property	Meaning
`event.target`	Element that triggered the event
`event.type`	Type of event (e.g., "click")
`event.preventDefault()`	Stops default action (e.g., form submit)

Example:

```
input.addEventListener("input", function (e) {
  console.log(e.target.value); // logs input's current value
});
```

03: Event handling on form

Working with `<form>` Tag

Purpose:

- Used to collect and send user data to the server.

3.1 Key Attributes

Attribute	Purpose
`action`	Target URL to send data to
`method`	GET (in URL) or POST (in body)

Important Behavior of `<form>` Tag:

The `<form>` tag is used to **collect and send data** from the user to a server.

When the form is submitted (e.g., by clicking a submit button), it sends the input data to the server using the specified method and action.

Key `<form>` Attributes:

1. **`action`** – URL where the form data will be sent. Example:

   ```
   <form action="/submit">
   ```

2. **`method`** – HTTP method used to send data:

   - `GET`: Adds data to URL (visible)
   - `POST`: Sends data in request body (hidden) Example:

   ```
   <form method="GET">
   ```

🌐 URL Structure with `<input>` Tags (in `GET` method):

Each `<input>` with a `name` attribute adds a **key-value pair** to the URL query string when submitted.

Example:

```
<form action="/search" method="GET">
  <input name="query" value="shoes" />
  <input name="color" value="red" />
  <button type="submit">Search</button>
</form>
```

When submitted, the URL becomes:

`/search?query=shoes&color=red`

If you change input names/values, the URL will change accordingly.

📌 Summary:

- `<form>` controls how/where data is sent.
- `action` = target URL
- `method` = how data is sent (GET → URL, POST → body)
- Each `<input name="key" value="val">` becomes `key=val` in URL (if GET)

◇ Form Events

◇ `submit` Event (on `<form>`)

```
form.addEventListener("submit", function (e) {
  e.preventDefault(); // Stop page reload
  console.log("Form submitted");
});
```

✅ Use to:

- Validate input before sending
- Prevent default behavior
- Collect data using `FormData`

◇ 4.2 Input-related Events (on `<input>`)

Event	Fires When...	Best For
`input`	As user types	Live preview, search
`change`	After change + blur	Dropdown, checkbox, radios
`focus`	Input gets focus	Highlight or helper text
`blur`	Input loses focus	Validation on leave
`keydown`	Key is pressed	Detect shortcuts or keys
`keyup`	Key is released	Check Enter, real-time typing

🩹 Focus and Blur Example:

```
<input onfocus="console.log('Focused')" onblur="console.log('Blurred')" />
```

PROJECT 01 – get live input and final input from user input

```
  <form id="myForm">
    <input id="username" name="username" />
    <button type="submit">Submit</button>
  </form>

  <p id="output"></p>

  const form = document.querySelector('#myForm');
  const username = document.querySelector('#username');
  const output = document.querySelector('#output');

// Live typing
username.addEventListener('input', (e) => {
  output.innerText = "Typing: " + e.target.value;
});

// Form submit
form.addEventListener('submit', (e) => {
  e.preventDefault();
  output.innerText = "Final Value: " + username.value;
});
```

Concepts:

1. Username.value we can use on input tag as we know input have username , and value property and that that comes from user input

2. e.target.value it's a event object of input to acces value .

3. e.preventDefault stop page to refresh after get submitted but button inside form

☑ Final Tips:

- Use `.value` on `input`, not `form`.
- Use `e.target` to access the element that triggered the event.

◇ 6. `FormData` in JavaScript

◇ What is `FormData`?

- A built-in JavaScript object to collect and work with **form inputs** easily.
- Used to **read**, **append**, or **send** form data (usually with AJAX or `fetch`).

◇ 6.1 How to Create `FormData` Object

```
const form = document.querySelector("form");
const formData = new FormData(form);
```

✅ It grabs all input values **(that have a `name` attribute)** from the form.

◇ 6.2 Common Methods

Method	Purpose
`formData.get("name")`	Get value of a field
`formData.set("name", value)`	Set/replace value
`formData.append("name", value)`	Add new value
`formData.entries()`	Loop over all key-value pairs

PROJECT 02 – get Inpurt data using FormData

```
<form id="myForm">
  <input name="username" value="lamb" />
  <input name="email" value="lamb@example.com" />
  <button type="submit">Send</button>
</form>


Js-------------------------------------------------
const form = document.querySelector('#myForm');

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

  const formData = new FormData(form);

  console.log(formData.get("username")); // lamb
  console.log(formData.get("email"));    // lamb@example.com

  for (let [key, value] of formData.entries()) {
    console.log(`${key}: ${value}`);
  }
});
```

☑ Use Cases:

- Send form data via `fetch()` (AJAX).
- Build form logic without manually reading every input.

Concepts:

1. `const formData = new FormData(form);`
2. array destructuring

It's a **destructuring `for...of` loop** that iterates over each entry (a `[key, value]` pair) returned by `formData.entries()`.

- **`formData.entries()`** returns an iterator of `[key, value]` pairs from a `FormData` object.
- **`for...of`** is used to loop over iterable objects like arrays, strings, maps, or anything with a `.next()` iterator.
- **`let [key, value]`** is **array destructuring**, used to extract the key and value from each entry.

04: Mouse Events

1. Mouse Events in JavaScript

What are Mouse Events?

Mouse events are fired when the user interacts with the mouse (clicking, moving, hovering, etc.) on the web page.

1.1 Common Mouse Events

Event	When it Happens	Use Case Example
`click`	On left mouse click	Buttons, links, toggles
`dblclick`	On double-click	Rename files, edit elements
`mousedown`	Mouse button pressed	Drag start
`mouseup`	Mouse button released	Drag end
`mousemove`	Mouse moves over element	Games, drawing, tracking
`mouseover`	Mouse enters an element	Show tooltip, highlight
`mouseout`	Mouse leaves an element	Hide tooltip
`contextmenu`	Right-click (context menu)	Custom right-click menus

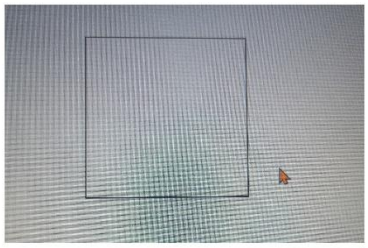
1.2 Basic Example: Click Event

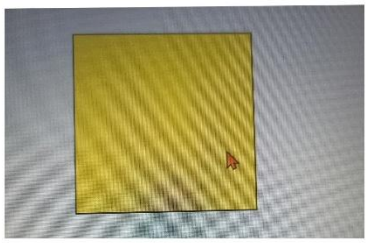
```
<button id="btn">Click Me</button>
```

```
document.getElementById("btn").addEventListener("click", () => {
  alert("Button was clicked!");
});
```

◇ 1.3 Example: `mouseover` and `mouseout`

```
<div id="box"></div>

 <script>
   const box = document.getElementById("box");

   box.addEventListener("mouseover", () => {
     box.style.background = "yellow";
   });

   box.addEventListener("mouseout", () => {
     box.style.background = "white";
   });
 </script>
```





◇ 1.4 Mouse Event Object Properties

Property	Description
`event.clientX`	X position relative to viewport
`event.clientY`	Y position relative to viewport
`event.target`	Element that triggered the event
`event.button`	Which mouse button (0=left, 2=right)

🩹 Bonus: Right Click Custom Menu

```
window.addEventListener("contextmenu", (e) => {
  e.preventDefault(); // Prevent default right-click menu
  alert("Custom Right Click!");
});
```

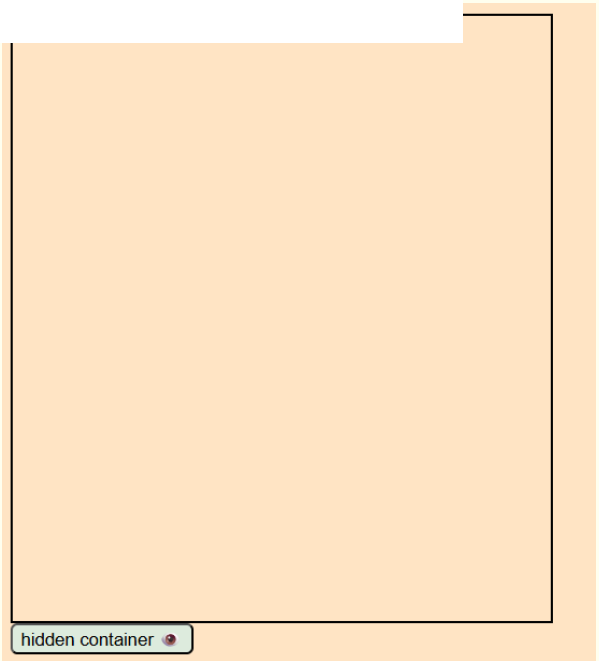
PROJECT 03 – Create menu on right click of mouse

pase1: when click on some field it show another component:
Make intial component display hide and then turn on on click

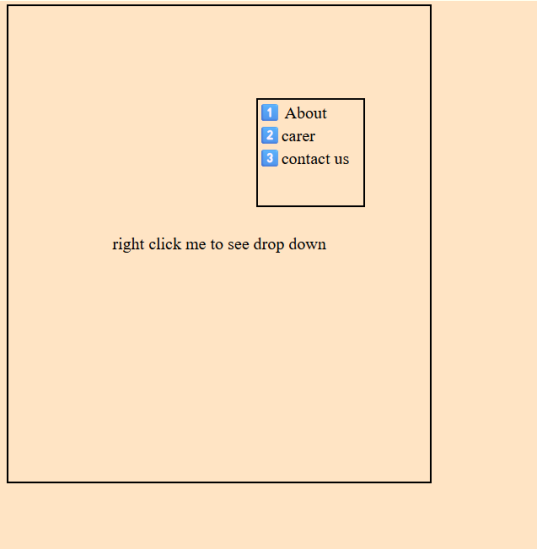
```
  <div class="container">

    right click me to see drop down

  </div>

  <button class="btn"> hidden
container ◎</button>
```

```
.container{

    height:450px;
    width: 400px;
    border: 2px solid black;
    display: flex;
    align-items: center;
    justify-content: center;
    display: none;
}
```

```
const container= document.querySelector('.container')
const btn= document.querySelector('.btn')

btn.addEventListener('click',(e)=>{

  e.preventDefault();
  container.style.display='block';

})
```



Project:
Right click to open Menu:
Project Link



Concept:

1. Display property none/block on event fire

05: Keyboard Events

2. Keyboard Events in JavaScript

What are Keyboard Events?

Keyboard events trigger when the user presses or releases keys on the keyboard.

2.1 Common Keyboard Events

Event	When it Happens	Use Case Example
`keydown`	Key is pressed down	Shortcuts, detect typing
`keyup`	Key is released	Form validation, live UI
`keypress`	(Legacy) Key is pressed	Not used in modern apps

⚠️ Use `keydown` and `keyup` instead of `keypress` (deprecated in ES6+).

2.2 Basic Example: Detect Key Press

```
document.addEventListener("keydown", (e) => {
    e.preventDefault();
    console.log("Key pressed:", e.key);
});

Key pressed: s
2script.js:3 Key pressed: v
script.js:3 Key pressed: b
script.js:3 Key pressed: a
```

◇ 2.3 Keyboard Event Object Properties

Property	Description
`e.key`	The actual key (e.g., "a", "Enter")
`e.code`	Physical key location (e.g., "KeyA")
`e.altKey`	`true` if ALT was pressed
`e.ctrlKey`	`true` if CTRL was pressed
`e.shiftKey`	`true` if SHIFT was pressed

🩹 Example: Detect `Enter` Key on Input

```
<input type="text" id="username" />

document.getElementById("username").addEventListener("keydown", (e) => {
  if (e.key === "Enter") {
    alert("Enter was pressed");
  }
});
```

☑ Final Tips:

- Use `mouseover` / `mouseout` for hover effects.
- `mousedown` + `mouseup` = useful for drag actions.
- Prefer `keydown`/`keyup` over old `keypress`.
- Use `e.key` for readable key values like `"a"`, `"Enter"`.
- Combine `ctrlKey`, `altKey`, `shiftKey` for custom shortcuts.

Here are clean and simple notes on **Event Bubbling, Capturing, `stopPropagation()`, and `{ once: true }`** — formatted just like your previous topics:

06: Event Bubbling, Capturing & Control Methods

◇ 1. Event Bubbling vs Event Capturing

◇ What are these?

When an event happens (like `click`), it **moves through the DOM** in phases:

- **Capturing Phase**: From `<html>` down to the clicked element.
- **Bubbling Phase**: From the clicked element **back up** to `<html>`.

◇ 1.1 Event Bubbling (Default)

- Event travels **from inner to outer** (child → parent).
- Handlers on parents will also trigger after child unless stopped.

🧪 Example:

```
<div id="outer">
  <button id="inner">Click Me</button>
</div>

document.getElementById("inner").addEventListener("click", () => {
  console.log("Inner Clicked");
});

document.getElementById("outer").addEventListener("click", () => {
  console.log("Outer Clicked");
});
```

🔸 Output (on click):

```
Inner Clicked
Outer Clicked
```

◇ 1.2 Event Capturing (Trick: Set `{ capture: true }`)

- Event goes **from outer to inner** (parent → child).
- You must manually enable it.

🧪 Example:

```
document.getElementById("outer").addEventListener("click", () => {
  console.log("Outer (capturing)");
}, { capture: true });

document.getElementById("inner").addEventListener("click", () => {
  console.log("Inner");
});
```

🔸 Output:

```
Outer (capturing)
Inner
```

◇ 2. stopPropagation()

◇ What does it do?

It stops the event from **bubbling up or capturing down** beyond the current element.

🧪 Example:

```
document.getElementById("inner").addEventListener("click", (e) => {
  console.log("Inner Clicked");
  e.stopPropagation();
});

document.getElementById("outer").addEventListener("click", () => {
  console.log("Outer Clicked");
});
```

🔸 Output:

```
Inner Clicked
```

✔️ `stopPropagation()` = Stop other handlers above/below from firing.

🔷 3. `{ once: true }`

🔹 What does it do?

The event listener runs **only one time**, then automatically removes itself.

🧪 Example:

```
button.addEventListener("click", () => {
  console.log("Clicked once only!");
}, { once: true });
```

✔️ Useful for:

- One-time popups
- First-click actions
- Confirmations

☑️ Summary Table

Feature	Behavior
Bubbling	child → parent (default)
Capturing	parent → child (`{ capture: true }`)
`stopPropagation()`	Stops event from moving further
`{ once: true }`	Runs listener just one time

07: Event Delegation in JavaScript

What is Event Delegation?

Event Delegation is a technique where you attach a **single event listener** to a **parent element**, and handle events for **its child elements** through event bubbling.

How It Works:

1. Add event listener to the **common parent** (like a container).
2. In the handler, check `event.target` to see **which child** triggered the event.
3. Use `if` or `matches()` or `classList.contains()` to filter the target.

Example:

```
document.querySelector('.container').addEventListener('click', function(e) {
  if (e.target.classList.contains('card')) {
    e.target.remove(); // only runs if a .card was clicked
  }
});
```

Common Methods Used:

Method	Use
`e.target`	The exact element clicked
`.matches('selector')`	Check if element matches a CSS selector
`.closest('selector')`	Find nearest ancestor (or self) that matches selector
`.classList.contains('class')`	Check if element has a specific class

PROJECT 04 – Add and remove card with use of Event Delegation:

Brute force

```
const container= document.querySelector('.container')
const card= document.querySelector('.card')
const btn= document.querySelector('.btn')

btn.addEventListener('click', (e)=>{
    const newCard = card.cloneNode();
    container.appendChild(newCard)

    newCard.addEventListener('click', ()=>{
        newCard.remove()
    })


})
```

Optimized :

```
const container= document.querySelector('.container')

const card= document.querySelector('.card')
const btn= document.querySelector('.btn')

btn.addEventListener('click', (e)=>{
    const newCard = card.cloneNode();
    container.appendChild(newCard)

    newCard.addEventListener('click', ()=>{
        newCard.remove()
    })


})


container.addEventListener('click', (e)=>{
    if(e.target.classList.contains('card')){
        e.remove();
    }

})
```