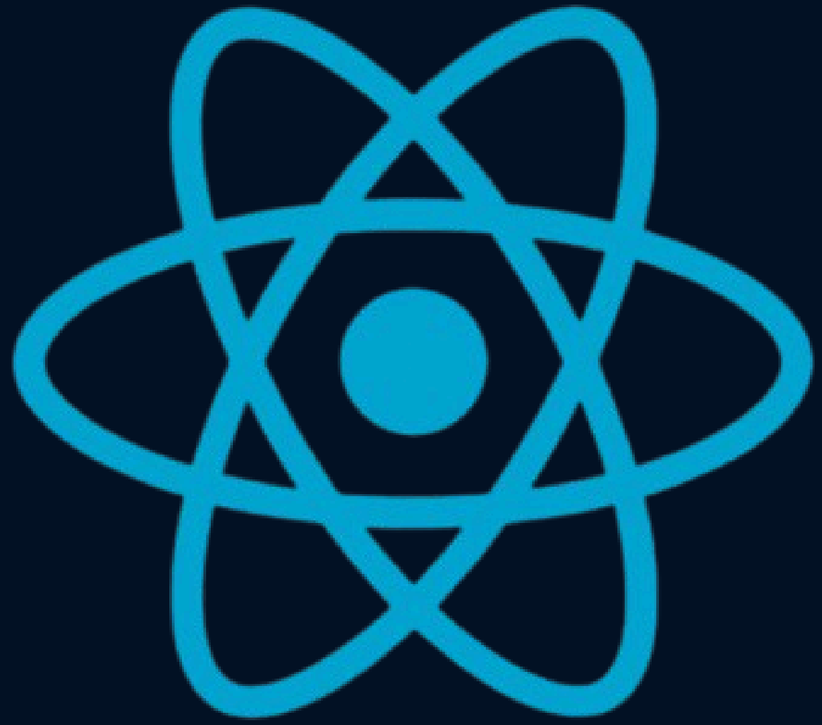
Otaku Tube

REACT JS NOTES (DAY 3)



Present by Sewak Dhakal

TOPICS FOR DAY 3

- Handling Click Event
- Handling Non-ClickEvents
- Event Object
- State in React
- Hooks
- useState()

- Closure in js
- Re-render: How does it work?
- CallBack in Set State Function
- More About State

HANDLING CLICK EVENTS

In React, if you want something to happen when you click a button, you attach a function to it..

```
function App() {
  function handleClick() {
    alert("You clicked me!");
  }

  return <button onClick={handleClick}>Click me</button>;
}
```

HANDLING NON-CLICK EVENTS

Not everything is a click — maybe the user types, hovers, or submits a form.
Examples:

```
<input onChange={(e) => console.log(e.target.value)} />   // typing
<div onMouseOver={() => console.log("Hovered!")}></div>  // mouse hover
<form onSubmit={(e) => { e.preventDefault(); alert("Submitted!"); }}></form>
```

React normalizes events so they work the same in all browsers.

EVENT OBJECT (E)

Whenever an event happens, React gives you an event object (e) that tells you what happened.
Example:

```
function handleChange(e) {
  console.log(e.target.value); // the thing user typed
}
```

e.target → the HTML element where the event happened.

e.preventDefault() → stops default browser action (like form submission refreshing page).

STATE IN REACT

State is like React’s memory — it remembers things between renders.

```
const [count, setCount] = useState(0);
```

count = the value right now.

setCount = the only way to change it.

When you change state, React re-renders your component with the new value.

HOOKS

Hooks are special React functions that let you “hook” into React’s features — like state, lifecycle, etc.

useState() → store and update values
useEffect() → run code when something changes
useRef() → store values that don’t cause re-renders

USESTATE()

import { useState } from “react”;

```
const [value, setValue] = useState(initialValue);
```

initialValue is used only once (on the first render).

Changing the state value → triggers a re-render.

CLOSURE IN JAVASCRIPT (WHY IT MATTERS IN REACT)

Closure = when a function remembers variables from where it was created, even if it runs later.

```
function App() {
  let name = "Sewak";

  function sayHi() {
    console.log("Hi " + name); // remembers "name" even if called later
  }

  return <button onClick={sayHi}>Say Hi</button>;
}
```

React uses closures under the hood when dealing with state updates and event handlers.

RE-RENDER: HOW IT WORKS?

1. State changes → React marks component as “dirty”.
2. React calls the component function again.
3. JSX is re-calculated → React compares old vs new DOM (Virtual DOM diffing).
4. React updates only the changed parts in the browser.

Important: Re-render doesn’t mean full page reload — React is smart.

CALLBACK IN A SETSTATE FUNCTION

Sometimes you want the new state based on the old state.

Bad:

```
setCount(count + 1);
```

(If multiple updates happen fast, this may use old value)

Good:

```
setCount(prevCount => prevCount + 1);
```

prevCount is always the latest value, even in async updates.

MORE ABOUT STATE

State updates are asynchronous — React batches them for performance.

Changing state → always causes a re-render (unless you prevent it by optimization).

If the new state is the same as the old state, React won’t re-render.