ReactJS (class 2)

* Event Handling in ReactJS.

In React, we can handle events by defining event handlers as functions and then attaching them to specific DOM element within your components.
In React, event handling is typically done within components using event handlers functions.

→ Role of Props in Event Handling

Props, on the other hand, are used to pass data from a pare-nt component to a child compo-nent. While props themselves are not used directly for event handling, we can pass event handler functions as props from a parent component to a child component to allow the child component to trigger actions in the parent component.

Item.js,

```
import './Item.css';

function Item(props)
{
    const singerName = props.name;

    function clickHandler()
    {
        console.log("Button Clicked");
    }

    return(
        <div>
            <p>{singerName}</p>
            <p>{props.children}</p>
            <button onClick={clickH
                -andler}>
                Click Me
            </button>
        </div>
    );
}

export default Item;
```

clickH-andler() is called when button clicked

→ Handling events with React elements is very similar to handling events on DOM elements.

→ React has the same events as in HTML, i.e., click, change, mouseover, etc.

→ React events are written in camel case syntax like, onClick instead of onclick.

Ques. what happens if we write onClick = {clickHandler()} instead of onClick = {clickHandler}?

In this scenario, the function clickHandler() will be called because this a direct function call, so writing like this, onClick = {clickHandler()}, will execute even if we did not clicked on button.

* States in ReactJS.

Item.js,

this is a functional component

```
import './Item.css';

function Item(props)
{
   let singerName = props.name;

   function clickHandler()
   {
      singerName = "Diljit";
      console.log("Button clicked");
   }

   return (
      <div>
         <p>{singerName}</p>
         <p>{props.children}</p>
         <button onClick={clickHa
            -ndler}>
            click me
         </button>
      </div>
   );
}

export default Item;
```

In this program, on button click event we are calling a function we are printing on console & also we are changing the singerName.
Internally, the value of singer-Name variable is changed but it will not displayed on the UI.

So, in this type of situation where we want to change the value on the UI as well on some event happening, in these situations we use "states" in ReactJS.

" agr hum ye chahte hai Ke hmare variable me value change Krne pe hmare UI bhi change ho to hmKo 'states' Ka concept use Krna pdega. "

states Ko use Krne UI change isliye hoga Ke agr hum states Ka use nahi Krte to React UI Ko bss ek baar render Krta hai, isli-ye UI pe changes nahi hote but states Ka use Krne se React UI Ko wapis se re-render Krta hai.

* Hooks.

In React, a hook is a special functi-on that allows you to "hook into" React state and lifecycle feature from functional components. Hooks enable functional components to manage state. Hooks provide a way to reuse stateful logic without changing your component hierarchy. They make it possible to use state, context, refs, and other React feat-ures within functional componen-ts. This allows us to write more concise and readable code.

Some commonly used hooks inclu-de :-

i) useState = It allow us to add state management to functio-nal components. We can create and manage state variables within our components using the "useState" hook.

ii) useEffect = Enables us to perform side effects in our components, such as data fetchi-ng, DOM manipulation or setting up subscriptions.

iii) useContext = Allows us to access the context provided by a 'context.provider' higher up in the component tree. This is used for passing down data to deeply nested components without prop drilling.

iv) useRef

v) useReducer.

vi) Custom Hooks

Hooks are a significant improvement in React because they allow you to reuse logic more easily, keep related code together, and make functional components more powerful.

—X— X —— X —— X —— X ——

- In previous code, we want to change the singerName variable on user interface as well.
- So, we have just seen that usestate hook is used to manage state variables within our component using "usestate" hook.
- Hence, we use "usestate" hook in previous code to change in UI

Agr hum states ka concept use nahi krr rhe and koi value update ho rhi hai component ke ander to hmare UI change nahi hoga.
Agr hum UI ko change krna chahte hai to state ka concep -t use krna pdega.

Hmare variable ki value ke change hote hi, state jo hai wo UI ko wapis se re-render krdega.

useState Hook

useState is a hook in React, used to manage state within functional components. Before the introduc -tion of hooks in React, state management was primarily done in class components using the "this.state" and "this.setState" syntax. However, with the advent of functional components, hooks like "usestate" provide a way to manage state in a more concise and readable manner.

Here's how "useState" works :-

(1) Import the Hook :- To use the "useState", we need to import it from the react library.

```
import React, {useState} from
                    'react';
```

(2) Declare a State Variable :- Inside your functional component, you can declare a state variable using 'useState'. The function takes an initial value as an argument and return an array containing the current state value and a function to update it. The initial value can be of any data type (string, number, object, array, etc.)

```
const [singerName, setSingerNa
    -me] = useState(props.
                    name);
```

useState() return 2 things, first the variable which we monitor and second is the function which will update the value.

this is the initial value of the variable.

In the previous code, 'singerName' is the state variable, and 'setSingerName' is the function used to update its value.

(3) use the state variable :- we can use the 'singerName' variable in our component's JSX or logic to display or manipulate the state.

(4) Updating state :- To update the state, we call the state update function 'setSingerName' with the new value we want to set. React will then re-render your compone-nt with the updated state.

```
setSingerName("sandhu");
```


— X — X — X — X — X —

code:-

```
import './Item.css';

//importing the hook
import {useState} from 'react';

function Item(props)
{
    //declare state variable
    const [singerName, setSinger
        -Name] = useState(props.
                        name);

    function clickHandler()
    {
        setSingerName("Sandhu");
        console.log("clicked");
    }

    return(
        <div>
            <p>{singerName}</p>
            <p>{props.children}</p>
            <button onClick={clickHa
                -ndler}>
                click me
            </button>
        </div>
    );
}
```

export default Item;

If we notice in the above code, we are changing the state of the variable on UI using the useState hook, and when the useState is giving an array in return, it consist of 2 things, i.e., singerName, and setSingerName function and both are declared as const values, but using setSingerName() function we are still able to change the value of singerName variable.

Ques After using const, why the value is still changing?

The value is still changing because when a component is re-rendered, after using useState, the function is executed again, & it creates a new 'singerName' variable with new value, & it has nothing to do with the previous variable & its value.

onChange This event is React which detects when the value of an input element change.

App.js,

```
import './App.css';
import NewProduct from './components/NewProduct';

function App()
{
  return(
    <div className='App'>
      <NewProduct>
      </NewProduct>
    </div>
  );
}

export default App;
```

NewProduct.js,

```
import './NewProduct.css';
import ProductForm from './Produ
                        -ctForm';

function NewProduct()
{
    return (
        <div className="new-product">
            <ProductForm>
            </ProductForm>
        </div>
    );
}

export default NewProduct;
```

ProductForm.js,

```
import './ProductForm.css';

function ProductForm()
{
    function titleChangeHandler(
                                event){
    {
        console.log(event.target.
                                value);
    }

    return (
        <form>
            <div>
                <label> Title : </label>
                <input type="text"
                    onchange={titleCha
                        -ngeHandler}>
                </input>
            </div>
        </form>
    );
}

export default ProductForm;
```

* suppose hmare paas ek form hai.

Title: [ ]

Date: [mm/dd/yyyy | v]

[Add Product]

Iss form me usne koi data dala and button pe click hua to iss button ka default behaviour hai that fields me se data hatt jyega & page reload hoga.
But hum iske default behavior ko rokk ke user ke enter kiye hue data ka ek object create krna chahte hai to kaise krenge?

→ sbse pehle hum usestate ke through 2 variables create krlenge & unke function ke through unki value set krdenge jo bhi user ne input fields me input diya hoga. uske baad form ke submit event pe uske default function ko prevent krke ek object create krenge & usme dono variables ki value store krwa lenge & then we print that object.

ProductForm.js,

```
import './ProductForm.css';
import {useState} from 'react';

function ProductForm()
{
    const [newTitle, setTitle] =
                        useState('');
    const [newDate, setDate] =
                        useState('');

    function titleChangeHandler
                        (event)
    {
        setTitle(event.target.
                        value);
    }

    function dateChangeHandler
                        (event)
    {
        setDate(event.target.
                        value);
    }

    function submitHandler
                        (event)
    {
        event.preventDefault();
```

```
const productData = {
    title: newTitle,
    date: newDate
};

console.log(productData);
}

return (
<form onSubmit={submitHa
                -ndler}>
  <div>
    <label>Title:</label>
    <input type="text"
      onChange={titlechan
        -geHandler}>
    </input>
  </div>

  <div>
    <label> Date: </label>
    <input type="date"
      min="2023-01-01"
      max="2023-12-31"
      onChange={datechan
        -geHandler}>
    </input>
  </div>
```

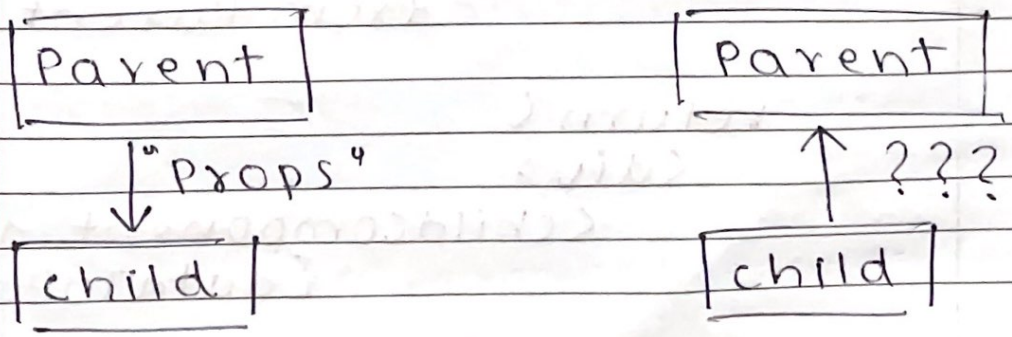
```
<div className='btn'>
<button type="submit">
Add Product
</button>
</div>
</form>
);
}

export default ProductForm;
```

X X X X X

In previous class we have seen that "props" are used to pass data from one component to another component, i.e., we can pass data from Parent component to child component.

But what if we want to pass data from child component to parent component?



How can we pass data from parent component to child component in ReactJS?

In React, you can pass data from a parent component to a child component by using props. Props (short for properties) are a way to pass data from a parent component to its child component.

- Parent component:- In the parent component, you define a child component and pass data to it using 'props'.

```
import React from 'react';
import childcomponent from
                './childcomponent';

function ParentComponent()
{
    const dataToPass = "Hello
                from Parent";

    return (
      <div>
        <childcomponent data=
                {dataToPass} />
```

```
        </div>
      );
    }

    export default Parentcompone
                        -nt;
```

- child component :- In the child component, you can access the data passed from the parent component through the 'props' object.

```
import React from 'react';

function Childcomponent (props)
{
  return (
    <div>
      <p> {props.data} </p>
    </div>
  );
}

export default childcomponent;
```

In this example, "dataToPass" variable in the Parent Component is passed to the child component as a prop named 'data'.
In the child component, we can access this data using 'props.data'.

This is the most common way to pass data from a parent compon-ent to a child components in React. Props can be used not only to pass data but also to pass functions and other React eleme-nts.

If you need to update the data in the child component based on user interactions or other events, you can also pass callback functio-ns from the parent to the child as props, allowing the child compo-nent to communicate changes back to the parent component.

How can we pass data from child component to parent component in ReactJS?

To pass data from a child compon-ent to a parent component in React, you can define a callback function in the parent component and pass it to the child component as a prop.
The child component can then call this callback function with the data that needs to be passed back to the parent compo-nent.

- Parent component :-

```
import React, {useState} from
                        'react';
import childcomponent from
              './childComponent';

function ParentComponent()
{
   const [dataFromChild, setDat
     -aFromchild] = useState('');

   // callback function to receive
     data from the child
```

```
const handleDataFromChild = (data) => {
    setDataFromChild(data);
};

return (
    <div>
        <p>Data from child: {dataFromChild}</p>

        <ChildComponent onDataFromChild={handleDataFromChild} />
    </div>
);
}

export default ParentComponent;
```

- child component :-

```
import React from 'react';

function ChildComponent(props)
{
    // function to send data to the
    parent
    const sendDataToParent = () =>
                {
                    const data = "Hello
                        from child";

                    // call the callback
                    function passed
                    from the parent
                    with the data
                    props.onDataFromC
                    -hild(data);
    };

    return (
        <div>
            <button onClick={sendDa
                -taToParent}>
                Send data to parent
            </button>
        </div>
    );
}
```

```
export default childcomponent;
```

In this example, we define a callback function 'handleDataFrom-Child' in the parent component and pass it to the child component as a prop named 'onDataFromChild'. When the button in the child component is clicked, it calls 'sendDataToParent', which in turn calls the callback function with the data "Hello from child".

The parent component receives this data & update its state and you can then use that data as needed in the parent component.

This pattern allows for communicati-on between parent & child compon-ents in React, with data flowing from the child to the parent through callback functions.

Handling multiple states using 'useState' Hook

Classroom
Page No.
Date

To handle multiple states using the 'useState' hook in React, you can call the 'useState' function multiple times, once for each state you want to manage. For example :-

```
import {useState} from 'react';

function MyComponent(props)
{
  const [count, setCount] =
                    useState(0);

  const [text, setText] =
                    useState('');

  function handleIncrement()
  {
    setCount(count + 1);
  }

  function handleTextChange
                    (event)
  {
    setText(event.target.
                    value);
  }
```

```
return (
  <div>
    <p> count: {count} </p>
    <button onClick={handle
        -Increment}>
      Increment
    </button>

    <br />

    <input type="text"
      value={text}
      onChange={handleTe
        -xtChange} />

    <p> Text: {text} </p>
  </div>
);
}
```

In this example, we're managing two states using 'useState': 'count' and 'text'. we're also defining two functions: 'handleIncrement', which updates the 'count' state when a button is clicked, and 'handleTextchange', which updates the 'text' state when the text input changes.

By calling 'usestate' twice, we're creating two independent pieces of state that can be managed separately. we're also using destructuring to assign the current value of each state and its corresponding setter function to separate variables (count and setcount, and text and setText).

Overall, using multiple usestate hooks can help you manage multiple pieces of state in a clean and organized way.

Other Alternative :-

We can also manage multiple states using a single usestate hook by passing an object as the initial state and using destruc-turing to access individual state variables and their corresponding update functions. For example :-

```
import {usestate} from 'react';

function Mycomponent(props)
{
   const [state, setstate] =
                          useState(
        {count:0, text:''});

   function handleIncrement()
   {
      setstate(
         prevstate =>(
            {...prevstate, count:
               prevstate.count+1}
         )
      );
   }

function handleTextchange
                          (event)
   {
      setstate(
         prevstate => (
            {...prevstate, text:
               event.target.value}
         )
      );
   }
```

```
return (
  <div>
    <p> count: {state.count}
                        </p>
    <button onClick =
      {handleIncrement}>
      Increment
    </button>

    <br />

    <input type = "text"
      value = {state.text}
      onChange = {handleText
        Change} />

    <p> Text: {state.text}
                        </p>
  </div>
);
}
```

In this example, we're still managing two pieces of state (count and text), but we're using a single useState hook to initialize both states as properties of an object (state). we're also using destructuring to access individual state variables

(count and text) and their corresponding update functions (setState).

To update a piece of state, we're using the functional update form of setState, which takes a callback function that receives the previous state as an argume-nt and returns the new state. we are spreading the previous state using the spread operator (...prevState) to create a new object with all the previous state properties, and then updating the property we want to change (count or text) using object property shorthand.

Using a single useState hook to manage multiple pieces of state can be a convenient way to keep related state together and reduce boilerplate code. However, it can also make the code more complex and harder to read, especially if you have many pieces of state or complex state updates. So its up to you to decide which approach works best for your specific use case.

Map Function in JS

The 'map' function is a built-in method in JavaScript, that allows you to apply a function to every element of an array and returns a new array with the results.
The 'map' function is one of the most useful and commonly used functions in JavaScript because it provides an easy way to transform and manipulate data in an array.

The syntax of the 'map' function is as follows :-

```
array.map(callback(element,
                   index, array),
                   thisArg)
```

Here, 'array' is the array that you want to map, 'callback' is the function that will be called on each element of the array, 'element' is the current element being processed, 'index' is the index of the current element, and 'array' is the original array that is being mapped, 'thisArg' is an optional parameter that

refers to the 'this' value that will be used when executing the callback function.

The 'callback' function takes three arguments: 'element', 'index', and 'array'. The 'element' argument is the current element being processed, the 'index' argument is the index of the current element, and the 'array' argument is the original array that is being mapped. The 'callback' function returns a new value that will be added to the new array that is being created.

Here is an example of using the 'map' function:-

```
const numbers = [1, 2, 3, 4, 5];

const doubledNumbers =
  numbers.map(number =>
    number * 2);

console.log(doubledNumbers);
                // 2, 4, 6, 8, 10
```

In this example, we have an array of numbers [1, 2, 3, 4, 5]. we

use the 'map' function to apply the 'number * 2' function to every element of the array. The 'map' function creates a new array called 'doubleNumbers' with the result of the 'number * 2' function applied to every element of the original array.

The 'map' function is often used in combination with other Javascript methods, such as 'filter', 'reduce' and 'forEach', to manipulate and transform data in an array. Here is an example of using the 'map' function in combination with the 'filter' function.


```
const people = [
    { name: 'Alice', age: 20 },
    { name: 'Bob', age: 30 },
    { name: 'charlie', age: 40 }
];

const names = people.filter(per
    -son => person.age >= 30)
  map(person => person.name);

console.log(names);
                    // Bob, charlie
```

In this example, we have an array of people with 'name' and 'age' properties. we use the 'filter' function to create a new array of people who are over the age of 30. we then use the 'map' function to create a new array of just the names of those people.

In conclusion, the 'map' function is a powerful tool for transforming and manipulating data in Javascript arrays. It allows you to apply a function to every element of an array and create a new array with the results. The 'map' function is often used in combination with other Javascr -ipt methods to create complex data transformations.

Important Difference

Page No.
Date
Classroom

The main difference between 'onClick={() => setCategory(da-ta.title)}' and 'onClick={setCategory(data.title)}' is when the setCategory function is called.

→ In the first case, onClick={() => setCategory(data.title)}, an arrow function is used as a callback function for the onClick event handler. This means that when the element is clicked, the arrow function will be executed and then call the setCategory function with the data.title argument. This is useful when you need to perform some additional logic or calculations before calling the setCategory function.

→ In the second case, onClick={setCategory(data.title)}, the setCategory function is called immediately when the compone-nt is rendered, and the return value of the setCategory functi-on is assigned to the onClick event handler. This is not desirable

because it will call the setCategory function on every render, which can lead to unnecessary re-renders & performance issues.

Therefore, the correct way to pass a function with arguments to an onClick event handler in React is to use the first approach with an arrow function as follows: `onClick={() => setCategory(data.title)}`. This way, the setCategory function will only be called when the element is clicked, and not on every render.