Function Component

❖ Function Component

- Functional component ek JavaScript function ki tarah kam karta hai. Ek single argument ko as a props accept karta hai jo react element ko return karta hai.

- Functional component is just a simple javascript function; it accepts the data in the form of props and returns the react element.

❖ Function Component Props

- Normally we have use props to pass the data from parent component to child component.
- In function component it is same but when we want to receive the data into child component we pass the parameter as a props and receive the data.

❖ Function Component State

- In function component there is a default hook is available that is called useState.
- useState accepts two values first is the variable name that is given the default state value and second is updated variable value that we want to update into the runtime.
- For example

```
const [firstVariablename, secondVariableruntimeChange] = useState(“Default value”);
```

❖ useEffect

- When we want to display the data after component render then we use useEffect hook.
- useEffect works asynchronously.

```
import React, { useEffect, useState } from "react";
const FunctioncompouseEffect = () => {
    const [api, setApi] = useState(false);
    const [api2, setApi2] = useState(false)
    useEffect(() => {
        fetch("https://jsonplaceholder.typicode.com/albums").then((res) =>
res.json()).then((response) => { console.log(response); })
    }, [])
    useEffect(() => {
        fetch("https://jsonplaceholder.typicode.com/todos").then((res) =>
res.json()).then((response) => { console.log(response); })
    }, [api])
    useEffect(() => {
        fetch("https://jsonplaceholder.typicode.com/posts").then((res) =>
res.json()).then((response) => { console.log(response); })
        return (() => {
            console.log("unmount called");
        })
    }, [api2])
    return (<>
        <div className="row">
            <div className="col-6 offset-6">
                <button onClick={() => { setApi(!api) }}>Click</button>
                <button onClick={() => { setApi2(!api2) }}>Click</button>
            </div>
        </div>
    </>);
}
export default FunctioncompouseEffect;
```

- There are two ways to get the data after render the page.
- First is to give [] dependency it means when our component is render then only our data is displayed. It is same like a constructor.
- Second give the dependency of state [api] and setState on button click event once the button is click then the data is show otherwise when our component render or re rendered then our data is show at that time but once the button is click no of times the data is show.

- useEffect also display the data when our component is closed we write the data on return statement inside useEffect once the component is closed the data is display it is also called the clean up.

❖ useLayoutEffect

- useLayoutEffect is used when we want to print some data when component is render before it means it works synchronously. The data is always display first.
- The useLayoutEffect works similarly to useEffect but rather working asynchronously like useEffect hook, it fires synchronously after all DOM loading is done loading. This is useful for synchronously re-rendering the DOM and also to read the layout from the DOM. But to prevent blocking the page loading, we should always use useEffect hook.

```
import React, { useEffect } from "react";
import { useLayoutEffect } from "react";
const FunctionCompoUseEffectUseLayoutEffect = () => {
    useEffect(() => {
        // getData();
        async function data() {
            await fetch("https://jsonplaceholder.typicode.com/posts").then((res)
=> res.json()).then((response) => { console.log(response); })
        }
        data();
    })

    // async function getData(){
    //     let Alldata = await
fetch("https://jsonplaceholder.typicode.com/posts").then((res)=>res.json()).then((
response)=>{console.log(response);})
    //     return Alldata;
    // }
    useLayoutEffect(() => {
        console.log("called useLayoutEffect");

        fetch("https://jsonplaceholder.typicode.com/todos/").then((res) =>
res.json()).then((response) => { console.log(response); })
        return (() => {
            console.log("called");
            // fetch("https://jsonplaceholder.typicode.com/todos/").then((res) =>
res.json()).then((response) => { console.log(response); })
        })
    }, [])
    // useLayoutEffect(async () => {
    //     console.log("called useLayoutEffect");
```

```
    //     // Warning: useLayoutEffect must not return anything besides a
function, which is used for clean-up.

    //     // It looks like you wrote useLayoutEffect(async () => ...) or returned
a Promise. Instead, write the async function inside your effect and call it
immediately:


    //     await fetch("https://jsonplaceholder.typicode.com/todos/").then((res)
=> res.json()).then((response) => { console.log(response); })
    //     return (() => {
    //         console.log("called");
    //         // fetch("https://jsonplaceholder.typicode.com/todos/").then((res)
=> res.json()).then((response) => { console.log(response); })
    //     })
    // }, [])
    return (<>
        <div className="row">
            <div className="col-6 offset-6">
                <h3>difference bten useEffect vs useLayoutEffect</h3>
                <p>useLayoutEffect is used when we want to print some data when
component is render before it means it works synchronously. The data is always
display first.</p>
                <p>The useLayoutEffect works similarly to useEffect but rather
working asynchronously like useEffect hook, it fires synchronously after all DOM
loading is done loading. This is useful for synchronously re-rendering the DOM and
also to read the layout from the DOM. But to prevent blocking the page loading, we
should always use useEffect hook.</p>
            </div>
        </div>
    </>);
}

export default FunctionCompoUseEffectUseLayoutEffect;
```

- Here use fetch api in both the useEffect and useLayoutEffect first display useLayoutEffect Api data then display useEffect API Data.

- In useEffect we use async await function but it is not use in useLayoutEffect when we want to use it gives the warning because useLayoutEffect works synchronously async await returns the promise.

```
// Warning: useLayoutEffect must not return anything besides a function, which is
used for clean-up.

    //     // It looks like you wrote useLayoutEffect(async () => ...) or returned
a Promise. Instead, write the async function inside your effect and call it
immediately:
```

❖ Use Memo

- Use memo returns the memorized value. When give dependency it always works in specific dependency.

- In general terms when we have multiple of data in Array or something else when data is count always it starts from first index or value, but when we use useMemo hook it stores the previous value data and accordingly the calculation or task is perform. So basically if we imporove our code performance and if we have bunch of data at that time we use useMemo.

- Use memo returns the memorized value. When give dependency it always works in specific dependency.

- Jb hamare pass bahot sare data hote hai array me ya koi or jagah jb hamara component rerender hota hai tab hamari value or index first se start hoti hai but jab hum useMemo ka use karte hai tab vo previous value ke basis pe next value ko calculate karta hai. So basically hum code ke performance ko improve karne ke liye jab hamare pass bunch of data ho tub hum uska use karte hai.

```jsx
import React, { useMemo, useState } from "react";
const FactorialusingUseMemo = () => {
    const [number, setNumber] = useState(1);
    // const [inc, setInc] = useState(0);
    const factorial = useMemo(() => factorialOf(number), [number]);
    const onChange = event => {
        setNumber(Number(event.target.value));
    };
    // const onClick = () => setInc(i => i + 1);
    return (
        <div>
            Factorial of
            <input type="number" value={number} onChange={onChange} />
            is {factorial}
            {/* <button onClick={onClick}>Re-render</button> */}
        </div>
    );
}

function factorialOf(n) {
    console.log('factorialOf(n) called!');
    return n <= 0 ? 1 : n * factorialOf(n - 1);
}

export default FactorialusingUseMemo;
```

❖ Use callback

- Use callback always returns a memorized function.
- For example we have multiple functions and each function have some different functionality perform so when we want to perform some functionality on some click event into single component then we use useCallback hook.

- Jab bhi hum callback function use krate tab memorized function returns karta.
- For example Jab hum Multiple function banate hai but particular function button click pe call karva na ho tab hum dependency dete hai but jab hame Multiple function and multiple button pe call karvana ho tab hu callback me same dependency deke use kar sakte hai.

❖ Use Context

- Use Context provide three functionalities
    1. Create context
        - Create context is used to create a function.
    2. Provider
        - When create a function in using use Context the provider helps to pass the data from one component to another component.
    3. Consumer or useContext
        - In old react version if we want to receive the data use consumer but in latest version we use hook called use Context it is used to receive the data from whenever our context is created.
- The main difference between props and useContext is that props is a react functionality to receive the data but useContext is a by default hook it is used to receive the data whenever the create context is created.

❖ Use Reducer

Use reducer is used when we have perform multiple operation into single function then we use Use reducer.

❖ Use Imparative Handle

- useImperativeHandle is a React Hook that lets you customize the handle exposed as a ref.
- In React, data is passed from parent to child components via props, in what is known as unidirectional data flow. The parent component cannot directly call a function defined in the child component or reach down to grab a value for itself.
- In certain circumstances, we want our parent component to reach down to the child component, getting data that originates in the child component for its own use. We can achieve this type of data flow with the useImperativeHandle Hook, which allows us to expose a value, state, or function inside a child component to the parent component through ref. You can also decide which properties the parent component can access, thereby maintaining the private scoping of the child component.
- This hook is used when we want to call the function from child to parent then we use useImparative.

❖ Use Ref

- Use ref hook is used to get the reference value that is not needed for rendering.