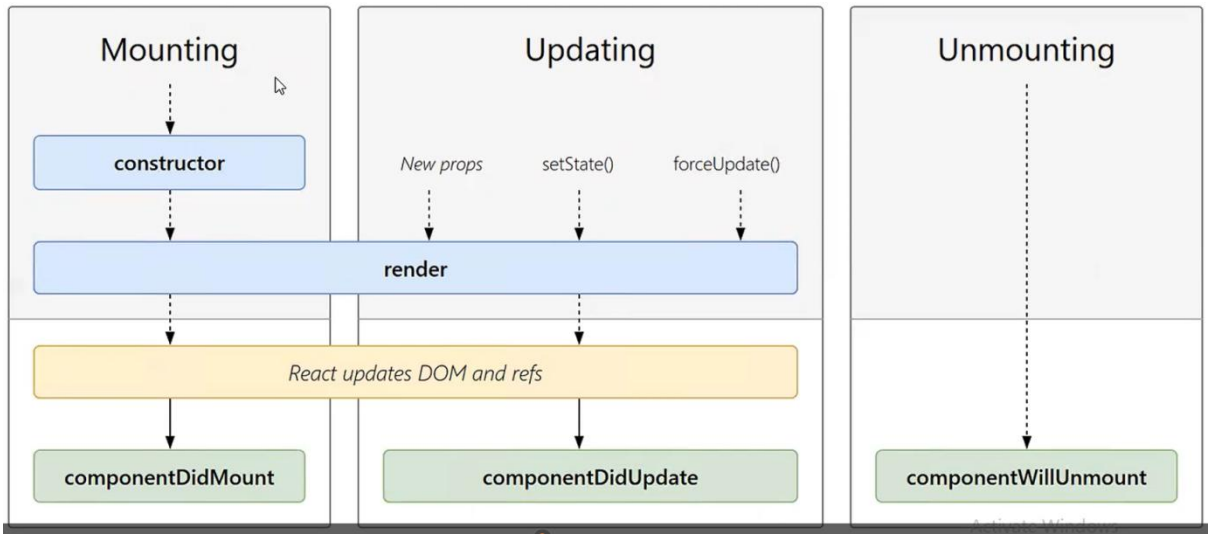
List and Hooks

❖ Explain Life cycle in Class Component and functional component with Hooks

Class component

- Class component always extends with React component.
- Inside class component if we want to return something then render is must required without render it gives us error and page is not return.
- These types of components can hold and manage their own state and have a separate render method to return JSX on the screen. They are also called Stateful components as they can have a state.

❖ Class component Life cycle



❖ Mounting

- First Phrase is mounting when component run successfully without any error that is called the mounting.
- Inside mounting constructor, render, Dom and componentDidMount are there without error.
- If error is occurred component is not mount.
- Mount call only one time.

❖ Updating

- In Updating when we update the component data then Updating is occur.
- Inside Updating we are able to add New props, update set state and create external function also.
- Then it goes to render after render Dom is update and componentDidUpdate are also there.
- When component not update if some error occur then componentDidnotUpdate occur.

❖ Unmounting

- Here we move from one component to another or close the component then componentwillUnmount occur.

Functional Components

- These types of components have no state of their own and only contain render methods, and therefore are also called stateless components. They may derive data from other components as props (properties).
- In function component it automatically return the page it doesn't required write anything else that's why function component is easy then class component. React also works with functional component.
- Functional component is just a simple JavaScript function; it accepts the data in the form of props and returns the react element.

- Whereas the class component will be created using the class keyword, and it extends the React. Component to make the class as a react component.
- Without writing a class it provide all class functionality it is called function component.
- It starts with function and function component name same as function name is called function component.
- It provides some advance features like Hooks, API.

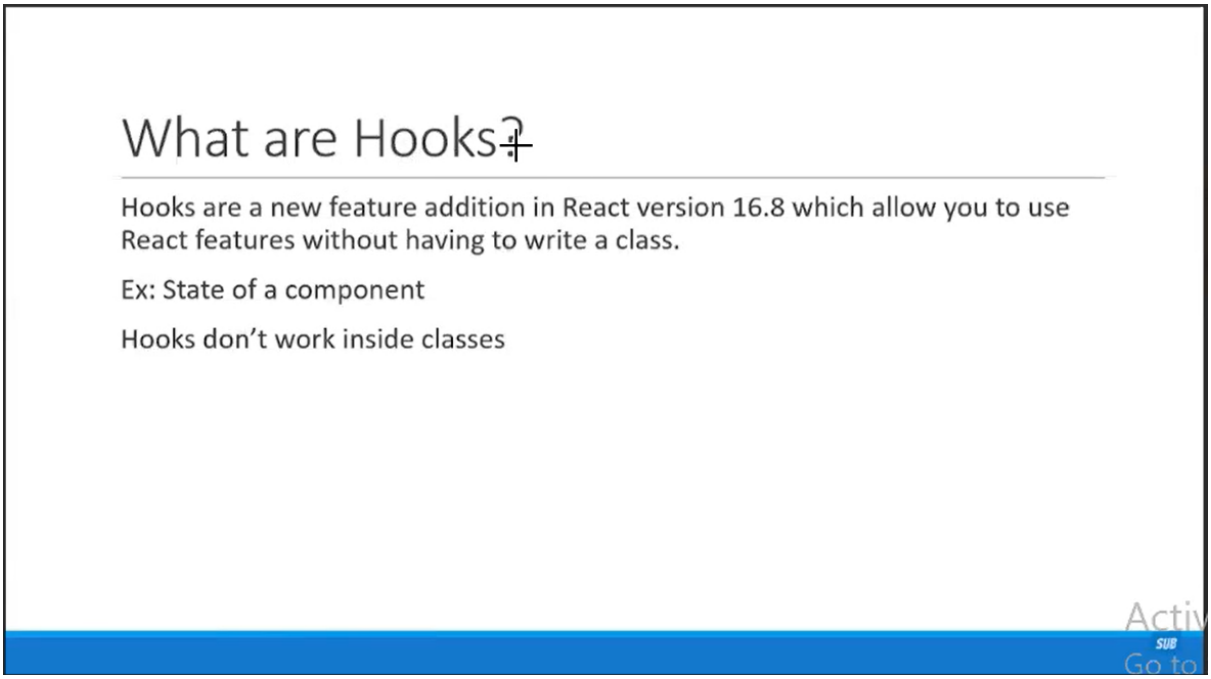
Ex of Function Component

```
import React from "react";

const Functioncompointro = () => {
    return (<>
       <h2>FunctionCompoIntro</h2>
    </>);
}

export default Functioncompointro;
```

❖ Hooks



Summary

Hooks are a new feature addition in React version 16.8

They allow you to use React features without having to write a class

Avoid the whole confusion with 'this' keyword

Allow you to reuse stateful logic

Organize the logic inside a component into reusable isolated units

Rules of Hooks

"Only Call Hooks at the Top Level"

Don't call Hooks inside loops, conditions, or nested functions

"Only Call Hooks from React Functions"

Call them from within React functional components and not just any regular JavaScript function

- Whenever we saw use it means it is hook.

❖ List of Hooks

I. useState Hook
II. useEffect Hook
III. useRef Hook
IV. useMemo Hook
V. useContext Hook
VI. useCallback Hook
VII. useReducer Hook

❖ UseState

- useState is React Hook that allows you to add state to a function component. It returns an array with two values: the current state and a function to update it. The Hook takes an initial state value as an argument and returns an update state value whenever the setter function is called.
- Here setter function are onClick, onChange, onfocus, onload, onchange.

```
import React from "react";
import { useState } from "react";
const Usestate = () => {
    const [count, setCount] = useState(0)
    const [item, setItem] = useState(1)
    return (<>
        <div className="row">
            <div className="col-6 offset-6">
                <h2>Usestate</h2>
                <p>{count}</p>
                <p>{item}</p>
                <button onClick={() => setCount(count + 1)}>Update</button>
                <button onClick={() => setItem(item * 2)}>Update</button>
            </div>
        </div>
    </>);
}

export default Usestate;
```

- Here count means current state(value), setCount means function for update the state and useState(0) here 0 is an initial state value as an argument.

❖ UseEffect

- When we want to display some content on component render time then we use UseEffect.
- Generally we use UseEffect when we want to saw the data to user when component is render.
- Basically useEffect works asynchronously.

```
import React, { useEffect, useState } from "react";

const Useeffect = () => {
    const [count, setCount] = useState(0);
    const [text, setText] = useState("");
    const [data,setData] = useState(false);
    // useEffect(() => {
    //     console.log("useEffect")
    // }, [])//dependency
    useEffect(() => {
        if (!data) {
            setText("Code Start...")
        }
        else {
            setText("Code END...")
        }
        console.log("useEffect");
    },[])
    return (<>
        <div className="row">
            <div className="col-6 offset-6">
                <h3>{count}</h3>
                <h4>{text}</h4>
                <button onClick={() => setCount(count + 1)}>Count Update</button>
            </div>
        </div>
    </>);
}

export default Useeffect;
```

- In useEffect we pass [] means only the render time the useEffect data will be show [] this is called dependency.
- But when we pass the dependency on button click event [count] then only on button click event useEffect functionality will show.

❖ Pass the data using props in Useeffect

```
import React, { useEffect, useState } from "react";
import Useeffectchild from "./Useeffectchild";

const Useeffect = () => {
    const [count, setCount] = useState(0);
    const [item,setItem] = useState(0);
    const [text, setText] = useState("");
    const [data, setData] = useState(false);

    useEffect(() => {
        if (!data) {
            setText("Code Start...")
        }
        else {
            setText("Code END...")
        }
        console.log("useEffect");
    }, [count])
    return (<>
        <div className="row">
            <div className="col-6 offset-6">
                <h3>{count}</h3>
                <h4>{text}</h4>
                <Useeffectchild item={item}/>
                <button onClick={() => setCount(count + 1)}>Count Update</button>
                <button onClick={() => setItem(item + 5)}>Item Update</button>
            </div>
        </div>
    </>);
}

export default Useeffect;
```

- Here I have load the Usereffectchild component and pass the item data.
- In count button I have gave the dependency on same file so every time when the count button is click the data is show.

❖ Useeffectchild.jsx

```
import React from "react";
import { useEffect } from "react";
const Useeffectchild = (props) => {
  useEffect(() => {
        alert("User effect child update")
    }, [props.item])//dependency
return ( <>
        <h4>{props.item}</h4>
    </> );
}

export default Useeffectchild;
```

- Here I receive the data using props.
- I give the dependency of item so when click on item button useeffect data will show.

❖ UseLayoutEffect

- useLayoutEffect is a version of useEffect that fires before the browser repaints the screen.
- 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.
- The useLayoutEffect hook works in the same phase as componentDidMount and componentDidUpdate methods. We should only use useLayoutEffect if useEffect isn't outputting the expected result.
- It works synchronously.
- It works when we click on component before render the page.
- We generally use in admin panel when we want to show data before render of the page.

```
import React, { useEffect } from "react";
import { useLayoutEffect, useState } from "react";
const UseLayoutEffect = () => {
    const [count, setCount] = useState(0);
    const [text, setText] = useState("");
    const [data, setData] = useState(true);
    useEffect(() => {
        console.log("useEffect")
    }, [count]) //dependency - if we want to print the data on some particular
event is trigg
    useLayoutEffect(() => {
        console.log("useLayouteffect")
        if (data) {
            setText("useLayouteffect admin data");
        }
        else {
            setText("End of useLayouteffect");
        }
    }, [text])

    return (
        <>
            <div className="row">
```

```
                <div className="col-6 offset-6">
                    <h2>UseLayouteffect</h2>
                    <h3>{count}</h3>
                    <h3>{text}</h3>
                    <h3>{data}</h3>
                    <button onClick={() => setCount(count + 1)}>Update
Count</button>
                </div>
            </div>
        </>);
}

export default UseLayoutEffect;
```

- Here first print useLayoutEffect because first load useLayouteffect.
- Then print useEffect because component render at that time it print.
- Here I pass two dependencies first dependency that I have pass in useEffect when we click on count button then it will print.
- Second dependency when if condition is called useLayoutEffect dependency again prints.

❖ UseMemo

- Basically when we create a function in our component each and every time our component is render or particular button is called then the function data is show.
- Use Memo use when the memorized value continuously show then we gave dependency and stop this memorized value to show each and every time.
- Use memo always return memorized value.

```
import React, { useMemo, useState } from "react";
const Usememo = () => {
    const [count, setCount] = useState(0);
    const [item, setItem] = useState(0);
    const [text, setText] = useState("Sachin");
    const Usememohook = useMemo(function multicount() {
        console.log("multicountDemo");
        return count * 2
    }, [count])
    return (<>
        <div className="row">
            <div className="col-6 offset-6">
                <h2>Usememo</h2>
                <h2>{count}</h2>
                <h2>{item}</h2>
```

```
                <h2>{text}</h2>
                <h2>{Usememohook}</h2>
                <button onClick={() => setCount(count + 1)}>Update Count</button>
                <button onClick={() => setItem(item + 5)}>Update Item</button>
                <button onClick={() => setText("Virat")}>Update Text</button>
            </div>
        </div>
    </>);
}

export default Usememo;
```

- Here I have pass count dependency so only count memorized value data display when count button is called.

❖ UseCallback

- When we create a child component and load in parent. In child component we create a function then if we provide any functionality in parent component the child component is also render every time.
- So we use memo to stop these rendering.

❖ Callbackchild.jsx

```
import React from "react";
import { memo } from "react";
const Callbackchild = (props) => {
   console.log("Callbackchild");
   return ( <>
        <h2>Callbackchild</h2>
        <button onClick={props.add}>Addtion</button>
    </> );
}

// function Callbackchild() {
//     console.log("Callbackchild");
// }
export default memo(Callbackchild);
```

- But when we have pass the function that is created in parent component and pass into child component then the same problem is occurred the child component function data is show that is not required.
- The solution is useCallback.
- useCallback always return memorized function.

```
import React, { useCallback, useState } from "react";
import Callbackchild from "./Callbackchild";
const Usecallback = () => {
    const [count, setCount] = useState(0);
    const [item, setItem] = useState(0);
    const add = useCallback(function addtion() {
        var a = 10;
        var b = 20;
        var c = a + b;
        console.log("addtion of two numbers ", c);
    }, [count])
    return (<>
        <div className="row">
            <div className="col-6 offset-6">
                <h3>Usecallback</h3>
                <h3>{count}</h3>
                <h3>{item}</h3>
                <Callbackchild add={add} />
                <button onClick={() => setCount(count + 1)}>Update Count</button>
                <button onClick={() => setItem(item + 5)}>Update Item</button>
            </div>
        </div>
    </>);
}
export default Usecallback;
```

- Here I pass the count dependency when we click on the count child component function is display data.

❖ Use Context Concept

- Use context basically declare with three things:

I. createContext(): It create a function to create a context.
II. Provider: It is used to pass the data to other component.
III. Consumer: It is used to receive the data to component.

❖ Usecontext.jsx

```
import React from 'react';
import CompoA from './CompoA';
import { createContext } from 'react';
const Fname = createContext();
const Usecontext = () => {
    return (<>
        <div className="row">
            <div className="col-6 offset-6">
                <h1>Usecontext</h1>
                <Fname.Provider value="dishank">
                    <CompoA />
```

```
                </Fname.Provider>
            </div>
        </div>
    </>);
}

export default Usecontext;
export {Fname};
```

- First we create a context using createContext() and store it in the variable.
- If we want to pass the data then we use Provider and give the value to the Provider.
- Here I load CompoA that is child of Usecontext.
- Then mention whichever component we want to pass the data in between.
- Then export the variable using export {variable name}.

❖ CompoA.jsx

```
import React from 'react';
import CompoB from './CompoB';
const CompoA = () => {
    return ( <>
        <CompoB/>
        {/* <h2>CompoA</h2> */}
    </> );
}

export default CompoA;
```

- Here I load the CompoB that is child of CompoA.

❖ CompoB.jsx

```
import React from 'react';
import CompoC from './CompoC';
const CompoB = () => {
    return ( <>
        <CompoC/>
    </> );
}

export default CompoB;
```

- Here I load the CompoC component that is child of CompoB.

❖ CompoC.jsx

```
import React from "react";
import { Fname } from "./08Usecontext";
const CompoC = () => {
    return (<>
        <Fname.Consumer>
            {
                (data) => {
                    return <h2>Hello {data}</h2>
                }
            }
        </Fname.Consumer>
    </>);
}

export default CompoC;
```

- Import the variable then I use Consumer to receive the data.
- Consumer default use JSX and callback function then return the value then we have pass.

❖ UseContext

- UseContext is a hook that is used to pass the data from nested component (parent child to child or vice versa).
- The process is above as same but expect consumer I use useContext hook.

❖ Usecontextmain.jsx

```
import React, { createContext } from 'react';
import One from './One';
const Fname = createContext();
const Usecontextmain = () => {
    return (<>
        <div className="row">
            <div className="col-6 offset-6">
                <h2>Usecontextmain</h2>
                <Fname.Provider value="Dishank Shah">
                    <One />
                </Fname.Provider>
            </div>
        </div>
    </>);
}
export default Usecontextmain;
export {Fname};
```

❖ One.jsx

```
import React from 'react';
import Two from './Two';
const One = () => {
    return ( <>
        <Two/>
    </> );
}

export default One;
```

❖ Two.jsx

```
import React from 'react';
import Three from './Three';
const Two = () => {
    return ( <>
        <Three/>
    </> );
}

export default Two;
```

❖ Three.jsx

```
import React from 'react';
import { Fname } from './09Usecontextmain';
import { useContext } from 'react';
const Three = () => {
  const data = useContext(Fname)
  return (<>

        <h2>Hello {data}</h2>
    </>);
}

export default Three;
```

- Instead of use consumer we define useContext hook create a variable here import and receive the data from Fname and just print the variable name using any of html tag.

❖ Usereducer

- Usereducer is used to create a reducer function inside reducer function we create multiple functions using state and action we give.
- When we want to short your code and create multiple if conditions inside reducer function then we use reducer.

```
import React, { useReducer } from 'react';
const initialState = 0
const reducer = (state, action) => {
    console.log(state, action);
    if (action.type === "INCREMENT") {
        return state + 1;
    }
    if (action.type === "DECREMENT") {
        return state - 1;
    }
    return state;
}
const Usereducer = () => {
    // const [count, setCount] = useState(1);

    // const increment = () => {
    //     setCount(count + 1)
    // }
    // const decrement = () => {
    //     setCount(count - 1)
    // }
    const [state, dispatch] = useReducer(reducer, initialState)
    return (<>
        <div className="row">
            <div className="col-6 offset-6">
                <h3>{state}</h3>
                <button onClick={() => dispatch({ type: "INCREMENT"
})}>increment</button>
                <button onClick={() => dispatch({ type: "DECREMENT"
})}>decrement</button>
            </div>
        </div>
    </>);
}

export default Usereducer;
```

- First we create a useReducer hook. Inside reducer hook we pass two arguments first is reducer it is a function, and second is initialState that we define top of the component.
- Here we pass state and dispatch. State is takes intialState value and dispatch it takes a object that is define in button click event what action is perform in condition.

- In reducer function we pass two arguments first is state and second is action.
- We receive state value and action that we want to perform.
- In action we write the condition action type that we have created in dispatch object type. If condition is satisfied then particular data is shown.

❖ Useref

- When we want to perform action on form like input, select and textarea then we use useRef.
- useRef returns a mutable ref object whose current property is initialized to the passed argument.

```
import React from 'react';

import { useRef } from 'react';

const Useref = () => {
    const inputRef = useRef(null);
    const data = () => {
        // console.log("data");
        // inputRef.current.focus()
        // console.log(inputRef.current.value);
        // inputRef.current.style.color = "red"
        console.log(inputRef.current.value.toUpperCase());
    }
    return (<>
        <div className="row">
            <div className="col-6 offset-6">
                <h2>Useref</h2>
                <table>
                    <tr>
                        <td>User Name</td>
                        <td><input type="text" ref={inputRef} /></td>
                        <td><button onClick={data}>Submit Data</button></td>
                    </tr>
                </table>
            </div>
        </div>
    </>);
}

export default Useref;
```

- Here I create a function on button click event I can perform focus, then print the current value, I can also give the style in text and I can also text to uppercase using button click event on text.