**React Life Cycle Hooks in Class Component  
- Mounting  
- Updating  
- Unmount  
  
FAQ: What is difference between constructor and componentWillMount() ?  
Ans:  
 - Constructor configures the functionality for component at the time of creating component.  
 - ComponentWillMount() initializes funcitonality after component is  
   created. It defines the functionality when component is requested.  
 - Constructor defines actions like  
        a) creating state  
        b) binding events to state  
 - ComponentWillMount() defines actions like  
        a) connecting with API  
        b) using context  
  
FAQ: What is Change Detection?  
Ans:  It is a technique used in applications where the changes in value are identified.  
  
FAQ: Who manages change detection in react?  
Ans:  componentWillUpdate()  
  
FAQ: How it identifies the changes?  
Ans:  By using a software design pattern called  
        "Single-Source-Of-Truth"  
        It verfies the previous value with current value.  
  
        previous value == current value => no change detected.  
        previous value != current value => change detected.  
  
        Model is reffered as "single source of truth" as it contains the  
        data.  
  
FAQ: In which phase of component life cycle "Change Detection" occurs.  
Ans:  Updating  
  
  
                                React Hooks  
- React 16+ versions introduced  "React Hooks" as alternative form component life cycle hooks.  
  
- Component life cycle hooks are available only for class components.  
  
- React Hooks are not replacement for component life cycle hooks, they are just alternative.  
  
- React 16+ versions made funciton component better than class components.  
  
- React 16+ versions provides several built-in hooks and also allows to create custom hooks.  
  
- React 18 introduced few new hooks into React funciton components.  
- React Hooks are backward compatible.  
  
FAQ: What is the role and responsibiltiy of React Hooks?  
Ans:  
React DOM manipulations  
React DOM server  
React Test Render  
React Shallow Renderer        
  
DOM Manipulations:  
- Adding element  
- Removing element  
- Updating data into element  
  
DOM Server  
- WebSocket  
- Identifies the changes and update the changes in DOM.  
- It can push updates without reloading.  
- Implicit Ajax  
  
Test Render  
- Render is the process of generating output.  
- Creates a layout  
- Render the layout  
  
Shallow Render  
- It is the process between render and painting  
  
  
Note: Component LifeCycle Hooks are only about life cycle events.  
         React Hooks are more than Life Cycle.  
  
Pre-define React Hooks:  
----------------------------------  
useToggle  
useFirestoreQuery  
useMemoCompare  
useAsync  
useRequireAuth  
useRouter  
useAuth  
useEventListener  
useWhyDidYouUpdate  
useDarkMode  
useMedia  
useLockBodyScroll  
useTheme  
useSpring  
useHistory    “react-router-dom”  
useLocation     “react-router-dom”  
useParams     “react-router-dom”  
useScript  
useKeyPress  
useMemo  
useDebounce  
useOnScreen  
usePrevious  
useOnClickOutside  
useAnimation  
useEffect  
useState  
useLocalStore  
useHover  etc.  
  
  
  
Rules for Configuring Hooks:  
1. React Hooks can't be defined in Class.  
2. React Hooks only for function component.  
3. Hooks must be at top level.  
4. You can't define hooks inside a JavaScript function.  
5. Can't use hooks in conditions and iterations.  
  
  
                                useState Hook  
                                ---------------------  
- It configures a state for component.  
- State can store values and make it available until the component unmount.  
- You can store any value into state.  
  
Syntax:  
           const [product, setProduct] = useState({});  
         const [products, setProducts] = useState([]);  
  
         setProduct({ });  
         setProducts([]);  
  
                                useEffect Hook  
                                ---------------------  
- It can manage  
        a) mounting  
        b) updating  
        c) unmount  
  
Syntax:  
               useEffect(()=> {  
                     // actions on mount  
                    // actions on update  
                    return(()=>{  
                        // actions on unmount  
                    })  
               },[ ])  
  
  
Ex:  
import { useState, useEffect } from "react"  
  
export default function ReactHookDemo()  
{  
    const [msg, setMsg] = useState();  
  
    function handleSuccessClick(){  
        setMsg(<SuccessComponent/>)  
    }  
  
    function handleErrorClick(){  
        setMsg(<ErrorComponent />)  
    }  
  
    return(  
        <div className="container-fluid">  
            <button onClick={handleSuccessClick}>Success</button>  
            <button onClick={handleErrorClick}>Invalid</button>  
            <hr />  
            <div>  
                {msg}  
            </div>  
        </div>  
    )  
}  
  
function SuccessComponent(){  
    useEffect(()=>{  
        alert('Success Component Will Mount');  
        return(()=>{  
            alert('Success Component will unmount');  
        })  
    },[])  
    return(  
        <div>  
            <h2>Login Success..</h2>  
        </div>  
    )  
}  
  
function ErrorComponent(){  
    useEffect(()=>{  
        alert('Error Component Will Mount');  
        return(()=>{  
            alert('Error Component will unmount');  
        })  
    },[])  
    return(  
        <div>  
            <h2>Invalid Credentials</h2>  
        </div>  
    )  
}  
  
  
                                useContext Hook  
- Context Memory**