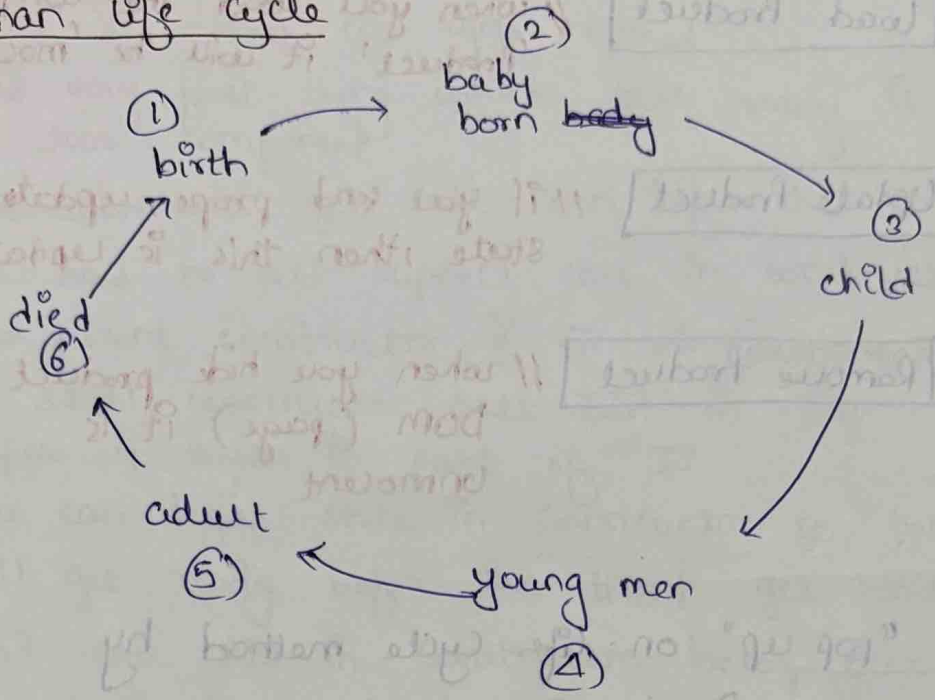
CH: 4 React LIFE CYCLE

#1 INTRODUCTION

* human life cycle



- Life cycle method for humans

1) birth
2) Job
3) had child
4) died

these are methods & they can be n' no. of methods

- 3 phases for human

1) birth
2) update/growth
3) death / expire

- Life cycle method for React

1) Constructor
2) getDerivedStateFromProps
3) shouldComponentUpdate
4) render
5) getSnapShot
& many more

- 3 phases for React

1) mount : means birth of state
2) Updating : means state updating
3) Unmount : means expire of state

* React life Cycle

- phase : there are only 3 phases in life cycle
- methods : there can be 'n' no. of methods

eg:

Product Life Cycle

1] Load Product // when you load the component 'Product' it will be 'mount'

2] Update Product // if you send props, update the state, then this is update

3] Remove Product // when you hide product from DOM (page) it is unmount

- you can call "pop up" on life cycle method by componentdidMount, it tells that component mount होगया तब pop up दिखाओ
- you can also listen to updated state & props & this happens by componentdidUpdate
- & if you don't Remove the changes (state) then the garbage will be their in your browser so you can remove the garbage by using componentWillUnmount
- All these life cycle will be used in Components
- when does life cycle methods execute ?
  → these method execute when
  1) Component is loaded
  2) Component is updated by Props
  3) Component is Removed
- for using life cycle methods in functional Components you need to use "Hooks"

#2 CONSTRUCTOR LIFE CYCLE

* Constructor

- Constructor will automatically get call & Constructor is a the first life cycle method
- The very first thing which gets ready is the Constructor in class component
- Constructor will get call even befor the render() call
- you need to add Super() call in constructor becoz the parent constructor है ना जो (.Component) class है उसका constructor call करने के लिए we use Super & Super is part of JS
- you can define state in Constructor so that it will get ready befor the HTML get render
- don't call API in Constructor becoz over HTML is not yet Ready to Render API call

- eg:

```
import React from 'react'
class App extends React.Component
{
    Constructor ()
    {
        Super();
        this.state ={
            data: "Anil"
        }
    }
    render()
    {
      return(
          <h1> hello world {this.state.data} </h1>
      )
    }
}
export default App
```

#2 Render () life cycle method

*Render ()

- Render method is used only in class Component
- Render() means execute करके Represent करना HTML जो हम लिखते हैं & जब भी कोई state या props update होगा automatically reRender हो गा
- Render method get calls when
  1) our component is ready
  2) when state/~~props~~ is updated
  3) & when props is updated

  in 3 cases our render () gets re-render
- After Constructor render() is made or gets execute

#3 ComponentDidMount Life Cycle Method

*ComponentDidMount

- first Constractor is called → then render() get execute & then componentDidMount get call
- componentDidMount : it means that HTML, CSS everything is ready. मतलब सब load होने के बाद (HTML, CSS) ये method ComponentDidMount () call होगा
- use of this method is when you use API, you can call API here, & whichever operations are dependend on HTML, that we will do here bcoz that can only be done once our page or HTML is ready
- the things which are dependent on HTML should be call in ComponentDidMount()
- this method has no Effect on state & props
- always remember the order of execution :
  1) Constructor will get call first
  2) then render ()
  3) & then ComponentDidMount
- when you update component, ComponentDidMount() will not execute again, it will execute only once when the page is ready

eg:

```
import React from 'React'
class App extends React.Component
{
    constructor()
    {
        console.log("constructor call");
        super();
        this.state = {
            name = "Eeeee"
        }
    }

    componentDidMount()
    {
        console.log("didmount call")
    }
    render()
    {
        console.log("render call");
        return(
            <div>
                <h1> Component Did mount {this.state.name} </h1>
                <button onClick={()=>{this.setState({name:'vishu'})}}
                    Update name click </button>
            </div>
        );
    }
}
```

Output: Constructor call
render call
didmount call

- after clicking on button
render call

- when you update state only render gets call (re-render)
ComponentDidMount will not be call again

#14 componentDidUpdate LC method

* componentDidUpdate

- this method will get execute only on state/props update
- this method won't get execute on mounting
- don't update state directly inside this method, because it will create a loop of update & this method will get call in loop, you can update state in this using some condition so that it won't get stuck in infinite loop
- this method has 3 parameters
- render() & componentDidUpdate will get call as you update the state/props (render toh होगा ही call होगा even at the first time, but DU update call नहीं होगा)
- this DidUpdate method has 3 parameters
  1) prevProps : It will provide you the previous prop & it will print always previous prop befor update

     eg: console.log(prevState)
     console.log(prevProps)
  2) prevState : it will provide you previous state
  3) snapshot : it will print undefined always, when you use getSnapshotBeforUpdate() method, then only some value will be in snapshot otherwise undefined

- Code:

```
Constructor Call
render Call
Component Did Mount Call
when you click on btn
render Call
Component DiDUpdate Call
```

#5 shouldComponentUpdate LC method

* shouldComponentUpdate

- this method asks you Question, shouldComponentUpdate or not
- you can set conditions when to update component & when to not
- you can also set wether to call render again or not or anyother LC method should get call or not
- this method can stop re-rendering
- This method will get Execute only when you update state/props
- Use: you control the re-rendering using certain conditions & application ~~of~~ ~~your~~ performance will increse by avoiding re-rendering all the time
- this method will only run (execute) when you update your component
- if you return true from this method () then only component will get updated & ComponentDidUpdate () will get execute bcoz this method executes iff the component gets updated.
- by default behaviour of shouldComponentUpdate () returns false bcoz of which re-rendering is blocked.
- if you return false from shouldComponentUpdate () then it will block render () & ComponentDidUpdate () methods
- when you don't use shouldUpdate () and you use DidUpdate () then whenever you click button (update component) render will run & then DidUpdate will run bcoz their is no control or condition on update
- Execution: 1) constructor ()
  2) Render ()

  on Update
  1) shouldComponentUpdate ()
  2) render ()
  3) DidUpdate ()

eg: import React from 'react'

```
class App extends React.Component
{
    constructor()
    {
        console.log ("Constructor Run");
        this.state ={
            count : 0
        }
    }

    shouldComponentUpdate () //this fun() will control re-render & didUpdate
    {
        console.log ("shouldComponentUpdate run")
        console.log ( this.state.count)
        if ( this.state.count > 2)
        {
            return true;
        }
    }

    componentDidUpdate () //this will run iff shouldUpdate() returns true
    {
        console.log ("Ran componentDid Update")
    }

    render ()
    { console.log ("render call")
      return (
          <div>
              <h1> should Update {this.state.count} </h1>
              <button onClick = {() => {this.state.set({
                                    count: this.state.count+1 })}}>
                  Click Update Counter </button>
          </div>
      )
    }
}
```

Output: Constructor Run call
render Ran call
click button (count = 1)
shouldComponent Update call
Click btn (count = 2)
shouldComponent Update call
click button (count = 3)
shouldComponent update call
render call
ComponentDidUpdate call

#6 componentWillUnmount LC method

* ComponentWillUnmount ()

- This ComponentWillUnmount() method gets call only when our Componet gets remove from Dom (website)
- When you hide & show a component (Remove component from Dom) then this willUnmount will get called
- Use: you can use this when you call API & you want to remove component
- This method gets execute only when any component is Removed from Dom
- This method won't get call when you set Display Property of a component to none through css
- for using this method you need 2 components

App.js

```
class App extends Component
{
  Constructor ()
  {
    super();
    this.state={
        show: true;
    }
  }
  render()
  {
    return(
      <div>
      {
        this.show? <student/> :
                  <h1>Remove</h1>
      }

      <button onClick={()=>
          this.setState({
          show: !this.state.show})} >
          toggle </button>

    </div>
  )
}
}
```

student.js

```
import React from 'react'
class Student extends React.Component
{
   componentWillUnmount()
   {
      alert("Unmount called")
   }
   render()
   {
      return(
         <div>
            <h1> Student Compo </h1>
         </div>
      )
   }
}
export Student
```

Output: you will get alert only when <Student/> component is removed from Dom on click

#5 CH:5 HOOKS, Arrays & REUSEING

#1 INTRODUCTION

* Need for hooks

- when you class component a lot of inbuilt features you get in class component like state, life cycle methods pure components etc
- but in functional components you don't get these features
- :. To use all the features of class component in functional component we need Hooks
- with hooks, we can use class component features in functional component like Life Cycle methods etc

* Hooks:

- you have to import "hooks" in your feinctional component
- Const [data, setData] = useState(false) // this is called destructuring
- Some of the hooks are:
  1) useState
  2) useCallback
  3) useMemo
  4) useRef
  5) useEffect

  & many more
- hooks start with "use" word & you need to import it from React
- "use" is a reserved keyword used for making hooks
- useEffect : all the life cycle methods which can be used in functional components comes in useEffect hook
- you can also make custom hooks

#12 USE EFFECT Hook

* useEffect

- useEffect hook is used in functional Component so that LC method can be used in functional Component
- whenever der component is made, or state/props updated that time this hook will get Call automatically
- you can apply conditions on which you control on which state/props this hook should get call & when it should get call
- useEffect() takes function as a parameter & it is a fun
- this useEffect() will get call when your component is made, or updated. automatically it will get Execute. it get call beoz of componentDidMount() method, as useEffect act as all methods of LC
- you can use more than one useEffect() method

eg:

```
import React, {useEffect, useState} from 'react'
function App()
{
  Const [count, setCount] = useState(0)
  useEffect(()=>{
      console.log("use Effect run")
  }
  useEffect(()=>{
      console.log("use Effect 2 run")
  }
  return(
    <div>
      <h1> useEffect - {count} </h1>
      <button onClick={()=>setCount(count+1)}>
                    Count Update </button>
    </div>
  );
}
```

#3 UseEffect with Conditions

* with Conditions

- you can control on which component state/prop useEffect should get execute by giving conditions
- you can even call useEffect on a specific state & call useEffect on specif props/state & useEffect will execute only when that prop/state is updated

- eg:

```
import React, {useState, useEffect} from 'react'
function App()
{
  const [data, setData] = useState(10);
  const [count, setCount] = useState(1000);
  useEffect(() => {
      console.log("when any state/prop is updated")
  }) → // this useEffect will get call when any state/prop is updated in this component
  useEffect(() => {
      console.log("data updated")
  }, [data])
      ↳ // now this useEffect will get call only when "data" parameter is updated
  return (
    <div>
      <h1> Count : {county} </h1>
      <h1> Data : {data} </h1>
      <button onClick={()=> setCount(count+1)}> Count </button>
      <button onClick={()=> setData(data+1)}> Data </button>
    </div>
  )
}
```

- now the useEffect which has send data as parameter will get call only when data is updated

#4 STYLE TYPE IN REACT

* Types of CSS in React

1) Stylesheets eg App.css
2) Inline style
3) CSS with modules

1) StyleSheet

App.js

```
import './App.css';
import './style.css'

function App()
{
    return(
        <div>
            <h1 className="Primary">
                Style using sheet
            </h1>
        </div>
    )
}
```

style.css

```
.primary{
    background-col : Black
    color : yellow
}
```

2) Inline Style

- you have to give two bracks in style attribute & use Camel Case for writting properties

eg:

```
<h1 style={{color:'red', backgroundColor:"black"}}>
                Style using inline </h1>
```

3) module CSS

- make a file with ".module.css" extention & write css in this file
- to use that file import it & when you want to use it you can add the className in tag

eg: `import style from './custom.module.css'`

```
<h1 className={style.success}> style apply</h1>
```

you have import it as "style" :. use style.class_name here

↳ This is class name which is present in custom.module.css file

#5 Handle Array with List

* map() & for() loop

- you can't use for, while, do while loop inside return () method
- you can only use map() to loop the array elements inside return ()
- you can also use filter() method inside return () for looping
- eg:

```
function App()
{
  const students ["Emmm", "vishu", "daddy"]
  const students_2 = [
        {name: "Emm",
         email: "Emm@gmail.com",
        }
        {
          name: "vishu"
          email: "vishu@gmail.com"
        }
  ]
  return(
    <div>
      <h1> Handle Array </h1>
        {
          students.map((data) =>
            <h1> Name is : {data} </h1>
          )
        }
        {
          for(i=0; i<students.length; i+1)  →// will give error becoz you can't use for inside return()
          {
            <h1> Name : students[i] </h1>
          }
        }
    </div>
  );
}
```

* Unique key Error

- you will get a unique-key warning while accessing the object elements of array list
- eg:

```
function App ()
{
  const user = [
     {name: 'Emma', mail: 'Emma@.com'},
     {name: 'Vishu', mail: 'vishu@.com'},
  ]
  return (
     <div>
       <h1>
       <table>
         <tr> <td> Name </td>
              <td> Email </td>
         </tr>
         {
           user.map((i) =>
           <tr>
             <td> i.name </td>
             <td> i.email </td>
           </tr>
         }
       </table>
     </div>
  )
}
```

Warning : each child in a list should have unique key prop. मतलब loop के अंदर जो child है उसके साथ मे एक unique key होनी चाहिए, Because जब भी list को reorder करते है ना तो React को एक identify करने के लिए कि ये एक key है अलग से
∴ it needs a key

∴ correct code

```
{
  user.map((item, i) =>
  <tr key = i >
     <td> item.name </td>
     <td> item.mail </td>
  </tr>
}
```

- now the warning will be resolve

eg:

```
user.map((i) =>
  i.name === 'Emma' ?
  <tr key = {i}>
    <td> {item.name} </td>
    <td> {i.mail} </td>
  </tr> : null
```

↳ This shows name = "Emma" entry only

* Importing BootsTrap

- go on the website & add the css import or link in the src → index.js file के अंदर import कर देना।
- eg:
  import {Table} from 'bootsTrap'
  & you can use the styling from bootstrap
- the properties should be mention in { } curly brackets while importing
- you can also use condition & show data.

* Nested List/Array

- Nested array means Array के अंदर array/list
- eg:

```
import React from 'react'
function App()
{
  const users = [
      {name : "Ruuu", address : [
          {Hn: "10", city : "Noida"},
          {Hn: "11", city : "Pune"}
          ]
      },
      { name : "Vishu", address : [
          {Hn : "101", city : "miraj" }
          ]
      }]

  return (
    <div>
      <h1> Table with Nested Array </h1>
      <Table>
        <tr>
          <td> Serial no. </td>
          <td> Name </td>
          <td> Address </td>
        </tr>
        { users.map((item, i) =>
          <tr key={i}>
            <td>{i}</td>
            <td>{item.name}</td>
            <td>
              <Table>
                {item.address.map((data) =>
                  <tr>
                    <td> data.Hn </td>
                    <td> data.city </td>
                  </tr>
                )}
              </Table>
            </td>
          </tr>
        )}
      </Table>
    </div>
  );
}
```

you will get the index also now using keys

this will loop on address array of users

#6 Reuse Component using loop

* Reuse Component

- you can Reuse a component using map() loop & pass array items as props

App.js

```
import {User} from './User'
function App()
{
  const arr = [
    { name: "Euuu", mail: "Eu@.com"
    },
    { name: "Vishu", mail: "vishu@.com"
    }
  ]
  return (
    <div>
      <h1> Reusing Component
      {
        arr.map((i) =>
        <User data={i} />
      }
    </div>
  );
}
```

User.js

```
function User(props)
{
  return(
    <div>
      <span>{props.data.name}
      <span>{props.data.mail}
    </div>
  );
}
export default User
```

- now you will send data of array as props from App to User
- using map() will will Reuse the <User> component

* Fragment (<> </>)

- you can't use inside a <table> so when you use component you can wrap all the elements inside a fragment " <> </> "
- you can't take class inside fragment, only key can be taken in <> fragment

#7 LIFTING STATE UP / SEND DATA FROM CHILD TO PARENT

* Send data from child to Parent

- you can send data from parent to child component using props
- you can send data from child to parent by the following process:
  1) create a fun() in parent component
  2) pass the fun() from parent to child using props
  3) now you can send data as a parameter from child to the parent by sending data into the receiving props function as parameter
  4) you can send any data in fun as parameter like obj., array or string

App.js

```
import User from './User'
function App()
{
   function parent(child_data)
   {
      console.log(child_data)
      alert(child_data)
   }
   return(
      <div>
        <h1> Lifting up state </h1>
        <User fun_parent={parent} />
      </div>
   );
}
export default App;
```

this child_data is data coming from User as parameter to this fun()

//this fun() parent is sending as prop to User →

User.js

```
function User(props)
{
 const data = {
    name: "Eeeee"
    mail: "Eeeee@.com"
 }
 return(
    <div>
      <h2> User data: </h2>
      <button onClick={
        ()=> props.fun_parent(data)
      }> click </button>
    </div>
 );
}
export default User;
```

this data is send as parameter to the parent fun

← data

CH:6 PURE COMPONENT & useMemo

* PURE COMPONENT

- Pure Component is a feature, you can just import it & use it.
- Pure Component is only for class component & for functional component there is a feature called memo (hook)
- pure Component हमारे component की Re-rendering को रोकता है
- for eg: you are set a name on a button click in a component, the same name is going to set on each time you click & so each time your component will re-render.
- But when you use Pure Component, it will match कि, पहले वाला "name" या value & जो Button click होने पर name डाला है (value) दोनो same है या अलग-अलग है अगर same हुआ तो Re-rendering को रोकेगा & अगर अलग-अलग है तो re-render करेगे
- Even you can use Pure Components for child component & do the same thing for Props. if props updated value is same as previous state then re-rendering will be stop by Pure Component
- जिस भी component की Re-rendering check करके रोकनी है उस component को Pure Component बना दो

- eg:

```
import React, { PureComponent } from 'react'
class App extends PureComponent --> // by this it will compare old & new state & then only it will allow re-rendering
{
  constructor()
  {
    super(); this.state = { count: 1 } }
  }
  render()
  {
    console.log("check-rerendering");
    return (
      <div>
        <h1>
          <button onClick={() => this.setState(
            { count: this.state.count
            })}> Update Click </button>
      </div>
    );
  }
}
export default App;
```

#2 UseMemo Hook

* useMemo

- Pure Component is for class components, same feature is of "useMemo" hook
- "useMemo" using this hook you can do same thing as Pure Component for functional component
- "useMemo" stops or blocks the re-rendering if the previous state of functional Component is same as updated state/props
- useMemo() takes a callback fun() as an parameter & second parameter it takes is the variables on which the useMemo() function should run
- ~~If not use memo~~ useMemo() जो variable लेगा as a parameter अगर वो variable update हुआ तो ही हमारा useMemo fun() run होगा।
- eg:

```
import React, {useState, useMemo} from 'react'
function App()
{
    const [count, setCount] = useState(0);
    const [item, setItem] = useState(10);

    const CountMemo = useMemo(function count(){
        console.log("CountMemo run");
        return count * 5 ──→ // this fun is not updating count
    }, [count])                  its just multiplying count into 5
                                 & return the value
    return (  → this tells that
        <div>   जोवा count update
            <h1>  होगा  तभी ये memo run होगा
            <h2> Count : {count} </h2>
            <h2> Item : {item} </h2>
            <h2> {CountMemo} </h2>
            <button onClick={()=>setCount(count+1)}> Update Count
                                                    </button>
            <button onClick={()=>setItem(item*10)}> Update Item
                                                    </button>
        </div>
}
```

- using useMemo you can stop unwanted fun call

CH:7 Ref

#1 Ref

* Ref in React

- in functional component you can't use "ref" directly
- ~~use "Ref" or~~ avoid use of "Ref" or use it minimum as possible
- "Ref" directly DOM को modify करता है, :. use it in Emergency situation only
- directly DOM को (Ref) से manupulate करेंगे तो React JS slow हो जाएगी
- genrally state/props से ही DOM को update करो
- "Ref" can directly manipulate DOM, you can use Ref in form for validating, to get values from form
- "Ref" को "createRef" करके import करो
- "Ref" से पुरा का पुरा DOM tree होता है
- using "Ref" you can change color of attributes, change their values etc
- eg

```
import React, {createRef} from 'react'
class App extends React.Component
{
  constructor()
  {
    super();                                  // you can give any name to this variable & in whichever tag you want to use ref set ref={this.inputRef} & then you can use ref & manipulate DOM directly
    this.inputRef = createRef();
  }
  componentDidMount()
  {
    console.log(this.inputRef);               // this will output the whole DOM tree
  }
  getVal()
  {
    console.log(this.inputRef.current.value); // you will get the input text value
    this.inputRef.current.style.color = "Red"; // apply red color to text
  }
  render()
  {
    return(
      <div>
        <input type="text" ref={this.inputRef} />   // property
        <btn onClick={()=>this.getVal()}> click Ref </button>
      </div>
    );
  }
}
```

#2 useRef hook

* useRef hook

- useRef() is a hook which can only be used in functional component & it is similar to "Ref"
- hooks can not be used in class component
- using useRef() hook you can directly manupulate Dom, can get data from <input> feilds of form, can style elements directly
- you can do all these things by changing state of component also
- Code:

```
impost React, {useRef} from 'react'
function App()
{ let RedRef = useRef(null)
  let inputRef = useRef(null);
  function handle()
  { RedRef.current.style.color = "red";
    conside.log("function call");
    inputRef.current.value = "1000" →// ये value set होगी 1000 automatically input field
    inputRef.current.focus(); →// auto cursor आयेगा input में
    inputRef.current.style.color = "Red" →// text color Red होगा
    inputRef.current.style.display = "none" →// this will hide that element
  }
  return(
    <div>
      <input type="text" ref={inputRef} />
      <input type="text" ref={redRef} />
      <button onClick={handle}> Click </button>
    <div>
  );
}
```

- as you will click the button, then both the Ref (RedRef & inputRef) will get execute bcoz both are in the handle fun()
- you can have more then one useRef in a component

* forwardRef

- forwardRef is used when your <button> on which after click you will modify DOM of <input> tag which is going to get update (like style of input) and both <inputs> & the <button> to update are in different Components, then you can use forwardRef do update the DOM of <input>
- Normal parent के button click से child component के अंदर tags का DOM कैसे manipulate करते, उसके लिए forwardRef() use करते हैं

App.js

```
import React, {useRef} from 'react'
import User from './user'
function App()
{
    let inputRef = useRef(null);
    function update()
    {
      inputRef.current.value = "Enter"
    }
    return(
        <div>
            <h1>
            <User ref={inputRef} />
            <button onClick={
                update
            }> Update </btn>
        </div>
    );
}
```

you have send inputRef as parameter in ref

User.js

```
import React, {forwardRef} 'react'
function User(props, inputRef)
{
  return(
    <div>
      <input type="text"
          ref={inputRef} />
    </div>
  );
}
export default forwardRef(User);
```

this is Ref parameter you can take any name your will get Ref inside this parameter not in Props

forwardRef उसे wrapper देता है

inputRef is send as parameter to User

CH: 8 Controlled/UnControlled HOC Component

#1) Controlled Component

* Controlled Component

- Controlled Component कि बात होरही हो तो ये समझ जाना की उस component के उपर कोई न कोई input field है. या form है जिसमे <input> है
- **Controlled Component**: when we control <input> feilds through React State then it is called Controlled Component
- **UnControlled Component**: when we directly handle <input> feilds through directly DOM by JS like getElementbyId
- **Code**:

```
import React, {useState} from 'React'
function App()
{
  let [val, setVal] = useState("default") ⇒ // this manupulation is using state
                                            ∴ this is called Controlled Component
  return (
    <div>
      <h1> Controlled Component </h1>

      <input type="text" value={val} />   ← You can't enter something in this

      <input type="text" value={val} onChange={
        (e) => setVal(e.target.value)} />   ← you can't set default value & val both attribute at same time

      <input type="text" defaultValue="Enter"
        onChange={(e) => setVal(e.target.value)} />

      <h3> value : {val} </h3>
    </div>
  );
};
```

#12 UnControlled Component

* UnControlled Component

- UnControlled Component : are those components which has input fields & these input fields should ~~be~~ ~~cant~~ not be controled by React State (useState)
  - It should be controlled by Ref or directly JS

eg:

```
import React, {useRef} from 'react'
function App ()
{
  let inputRef = useRef(null) ⟶ // controlled by Ref
  function submitform (e)
  {
    e.preventDefault()
    console.log("value:", inputRef.current.value)
    let data = document.getElementById('20').value
    console.log(data);   ↳ manupulating dom using JS
  }
  return (
    <div>
      <h1> UnControlled Component </h1>
      <form onSubmit={submitform}>
      <input ref={inputRef} type="text" />
      <input type="text" id="20" />
      <button> Submit </button>
    </div>
  );
};
```

#13 HOC

* Simple HOC

- HOC : High Order Component is a Component, जो दुसरे component को as a props input लेता है & एक अलग से component को Return करता है
- मतलब एक component जो Component as input लेता हो & एक अलग सा component as output देगा उसे HOC बोलते है

Eg:
```
import React, {useState, useRef} from 'react'
function App()
{
  return(
    <div>
      <HOCgreen cmp={Counter} />
      <HOCRed cmp={Counter} />
    </div>
  );
}
function HOCRed(props)
{
  return(
    <h2 style={{color='red'}}>
      <props.cmp/>
    </h2>
  );
}
function HOCgreen(props)
{
  return <h2 style={{color='green'}}> <props.cmp/> </h2>
}
function Counter()
{
  const [count, setCount] = useState(0)
  return(
    <div>
      <h3> {count} </h3>
      <button onClick={()=>setCount(count+1)}> Update </button>
    </div>
  );
}
```

Component send as props

HOCgreen is a component which takes Counter component as props

this HOCRed component takes <Counter> component as parameter

using Counter component with help of props

// this component returns a component i.e it is HOComponent

CH : 9 ROUTING

#1) INTRO

* Routing

- Components को Pages में convert करें तो हम & उनके बीच Links लगा देंगे ताकी हर एक functionality के लिए अलग page बन जाए, this is nothing but Routing
- Routing : We need different different pages for different functionalities & to convert component into pages so that they can be open on a specific URL
- you can install routing by : npm install react-router-dom
- Routing means in a <navbar> if you click HOME, about content, then you will see diff pages (on URL click)
- eg :

```
import React from 'react'
import {BrowserRouter as Router, Link, Route} from 'react-router-dom'
function App()
{
  return(
    <div>
      <Router>
        <Link to="/about"> About Page </Link>      // this "to" means Route to /about URL when click on "About Page"
        <br />
        <Link to="/Home"> Home Page </Link>
        <Route path="/home"> <Home/> </Route>
        <Route path="/about"> <About/> </Route>
        <Route path="/LogIn"> <LogIn/> </Route>
      </Router>
    </div>
  );
}
function Home()
{
  return(
    <div>
      <h1> Home Page !! </h1>
      <p> This is home page of web!! </p>
    </div>
  );
}
function About()
{
  //Code
}
```

// when you will click on "About Page" that time about wala component will be shown on website

// This path means जब URL में /LogIn आएगा तो <LogIn/> component show करो

// Home Component

eg: 2

src
> index.js

```
import { BrowserRouter as Router } from 'react-router-dom'
ReactDOM.render(
    <React.StrictMode>
        <Router>          // we have wrap everything inside router which we imported
            <App'/>
        </Router>
    </React.StrictMode>
    document.getElementById('root')
);
```

now सारी चीजे App.js मे होती है :. everything is wrap in Router

src
> App.js

```
import React from 'react'
import {Link, Route} from 'react-router-dom'
import Nav from './Nav'
import Home from './Home'
import About from './About'
import Links from './Links'
function App()
{
    return(
        <div>
            <Links/>
            <Route path="/about"> <About/> </Route>
            <Route path="/Nav"> <Nav/> </Route>
            <Route path="/" exact={true}> <Home/> <Route/>
        </div>
    );
}
export default App;
```

//this is a component

//home को कोई path नही है

exact मतलब जब exact match हो Link URL की "/" उसके आगे कुछ न, हो तब ही home दिखेगा

- Note : if you don't give "exact" then उस "/" के आगे अगर कोई भी Route आया तो भी तो page दिखेगा।
:. use exact for exact URL match

Src
> Links.js

```
import {Link} from 'react-router-dom'
function Link()
{
   return(
        <div>
          <Link to = "/" > click Home </Link>
          <Link to = "/about"> Click about </Link>
        </div>
   );
}
export default Nav;
```

//इस Link से ultimately <a> tag तैयार होरहा है

* 404 - Page Not found

- 404 - Page : if a Page which is not present in our website and user tries to access it like "URL/contact-us" but if contact us page is not their then we will show the user "Page Not found" error
- हम जो Route present नहीं है website में उसे 404 - Page दिखाएंगे
- make a component which ~~inka~~ includes 404 - Page Error to display
- add this page to "*" Route so that जब कोई Route match नहीं होगा तो ये "*" वाला Route follow हो जाएगा

- eg:

```
<Router>
    <Link to "/"> Home </Link>
    <Switch>
        <Route path = "/" exact = {true} > <Home /> </Route>
        <Route path = "*" > <PageNotfound /> </Route>
    </Switch>
</Router>
```

"*" means जब Route match न हो तो क्या दिखाना है

#2 Dynamic Routing with Params

* Dynamic Routing

- Dynamic Routing means you will define Route in Loop & this Route will change on basis of some id like instagram profile pages
- How many Routes you will make for instagram users?
  ∴ in such scenarios you will pass the "Route" as parameter & this will change dynamically for ~~best~~ person to person (id of insta)

src
> App.js

```
import {Link, Route} from 'react-router-dom'
function App()
{
  let users = [
    {id:1, name='Emma'},
    {id:2, name='Vishu'},
    {id:3, name='daddy'}
  ]
  return(
    <div>
      <Router>
        <h1> React Dynamic Routing </h1>
        {
          users.map((i)=>
          <Link to={"/user/"+i.id+"/"+i.name}
              <h3> {i.name} </h3>
          </Link>
          )
        }
        <Route path="/user/:id/:name"> <User/> </Route>
      </Router>
    </div>
  );
```

User.js

```
import {withRouter} 'react-dom'
function User(prop)
{
  console.log(prop.match.params.id)
  return(
    <div>
      <h1> id : {
        prop.match.params.id
      } </h1>
      <h3> name : {
        prop.match.params.name
      } </h1>
    </div>
  )
export default withRouter(User)
```

this is HOC

CH:10 API

#1 API INTRODUCTION

* API Get Call

- API (Application Programming Interface)
- We can't connect React or Angular directly to database, for connecting to DB we need PHP or Node or anyother server side programming language
- Server side programming we can create API also & from here data can be taken out & used in frontend like React
- API gives data in JSON format
- We use "get" method to get the data from API
- call API inside "useEffect()" hook or ComponentDidMount function
- eg:

```
import {useEffect, useState} from 'react'
function App()
{
  const [data, setData] = useState([])   → you have set an empty array as default value of data
  useEffect(()=>{
    fetch("link/for data/fetching").then((result)=>{
      result.json().then((res)=>{ // this is the same thing as you do in JS for fetching API data
        //console.log(res)
        setData(res)
      })
    })
  }, [])
  return(
    <div>
      <h1> Get API Call
      <table>
        <tr>
          <td> Id </td>
          <td> Name </td>
        </tr>
        {
          data.map((i)=>
          <tr>
            <td>{i.userId}</td>
            <td>{i.name}</td>
          </tr>)
        }
      </table></div>
  );
}
```

//data is array

this is the JSON data comming from API & do according to it

* POST method with API

- using POST method you can send data to API
- eg:

```
import {useState} from 'react'
function App()
{
  const [name, setName] = useState("")
  const [email, setmail] = useState("")
  function saveUser()
  {
    console.log({name, email});
    let data = {name, email}
    fetch("Link/to/API", {
      method: POST,
      headers: {
        'Accept': 'aplication/json',
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(data)
    }).then((result) => {
      console.log("result", result);
    })
  }

  return (
    <div>
      <h1>
      <input type="text" value={name} onChange={
        (e)=>{setName(e.target.value)}} />

      <input type="text" value={email} onChange={
        (e)=>{setmail(e.target.value)}} />

      <button onClick={saveUser}>New User</button>

    </div>
  );
}
```

you will get new data in name & email once the user data in the input fields

fetch के अंदर उस object as a parameter जाता है

Link to the API

// this will add data in the API
// this is same as we do in JS

body के अंदर data send करनी होता है

// you will take the data from user throw input text of form and that data will be ~~save User~~ updated in the variables name & email

* DELETE method API call

- Using delete method you can delete data inside API
- Code:

```
function App()
{
    const [users, setUsers] = useState([])
    useEffect(()=>{
        getList()
    },[])
    function getList()
    {
        fetch("API/Link").then((result)=>{
            result.json().then((res)=>{
                setUser(res);
            })
        })
    }

    function deleteUser(id)
    {
        fetch('API/Link/${id}", {
            method:'DELETE'
        }).then((result)=>{
            result.json().then((res)=>{
                console.log(res);
                getList()
            })
        })
    }
    return(
        <div>
            <table>
                // all code same from get API call
                // same code
                <td>{i.email}</td>
                <td>
                    <button onClick={()=> deleteUser(i.id)}>
                        Delete </button>
                </td>
            </table>
        </div>
    );
};
```

//we have put it in fun because after deleting we will call it and we will see the remaining records after deleting

this id is send in the URL

// fetch will return a promise

// this is whole JS

Delete method will delete the entry with specific id

// .then मतलब promise को resolve कर रहे है

#12 PRE FILLED FORM

* PRE FILLED FORM

- we will load the pre filled (पहले से भरा हुआ) data in the form
- तत्काल्य पहले से भरा ~~हुआ~~ हुआ data form में वापिस दिखाएंगे (form के fields में)

ID	Name	Mobile	Operation	
1	Emm	8600	Delete	Update
2	Vishu	8700	Delete	Update
3	Sank	8800	Delete	Update
4	Prachi	8900	Delete	Update

Code:

```
function App()
{
    const [users, setUser] = useState
    Const [name, setName] = useState("");
    const [mobile, setMobile] = useState("");

    useEffect(() => {
        getUsers();
    }, [])
    function getUsers()
    {
        fetch("http://400/todo").then((result) => {
            result.json().then((resp) => {
                setUser(resp)
                setName(resp[0].name);
                setMobile(resp[0].mobile);
            })
        })
    }
// function delete User previous & Codes
```

```
return (
    <div> // <btn> delete </btn>   <btn> Update </btn>
        //all previous code for displaying
        // & delete code
        <input type="" value={name} />
        <input type="text" value={mobile} />
        <btn> Update User </btn>
    </div>
);
} //closing of function App ;
```

prefilled data will be seen in form

your will see the data in the filled

~~* Adding btn update so that the data wi~~

* Adding "update" btn

- now we will add update btn, so that after clicking on update btn we will get the prefilled data in the form filled
- make btn update in users.map()

Code

```
{ //previous codes
  function selectUser(id)
  {
    setName(users[id-1].name);
    setMobile(users[id-1].mobile);
  }
  return (
    //previous all code
    <td> <btn onClick={()=>deleteUser(i.id)}> Delete </btn> </td>
    <td><btn onClick={()=>selectUser(i.id)}> Update </btn></td>
    //other code of <div
    //  <input type="text" value={name}/>
  );
}
```

now on clicking on update btn the data of that person will get filled in the form

CH : 11 PREVIOUS STATE

(Hi) PREVIOUS STATE IN FUNCTIONAL COMPONENT

* PREVIOUS STATE

- Previous State: मतलब जो अभी हमारी current state है उससे पहले वाली (state की values)
- मतलब हम जब variables को update करते हैं, तो उन variables को update करने से पहले, जो उनकी values थी, तो Previous State है

- Code

```
import :/App.css;
import React ;{ useState } from 'react'
function App ()
{
    const [count, setCount] = useState (1);
    function updateCounter ()
    {
        let rand_no = math.floor(math.random () *10)
        setCount ( (pre) => {
            console.war (pre);
            return rand_no ;
        }
```

→ this setCount will take a fun as argument which has ce parameter and that parameter has the previous value

→ This return value will be the current value

OR

wrong way to do this

```
for(let i=0; i<5; i++)
    setCount(count+1);
```

// This count increase count by 5 because Async value है तो ∴ तो loop 5 बार चलते से पहले उसकी value गिना जाती है

OR

```
for (let i=0; i<5; i++)
{
    setCount ((pre) => {
        prev +1 ;
    });
}
}
return (
    <div>
        <h1> {count} </h1>
        <btn onClick = {updateCounter} > Click me </btn>
    </div>
);
}
```

// correct way to do the above now setCount will increase by 5

#12 PREVIOUS PROPS IN FUNCTIONAL COMPONENT

* Previous Props

- Previous Props: मतलब current value जो props की है, उससे पहले जो props की value थी, उसे Previous props बोलते हैं
- मतलब props की current value से पहले वाली value

Parent Component

App.js

```
import React from "react"
import User from "./User";

function App()
{
  const [count, setCount] = React.useState(0)
  return (
    <div>
      <User count={count} />
      <btn onClick ={() => setCount(count+1)
              }> Update Count
              </btn>
    </div>
  );
}
export default App;
```

data is passed to User →

Child Component

User.js

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

function Users(props)
{
  const lastVal = useRef();
  useEffect(() =>{
    lastval.current = props.count
  })

  const previousProps = lastVal.current
  return(
    <div>
      <h1> current val {props.count} </h1>
      <h2> previous val {previousProp} </h2>
    </div>
  );
}

exprot default User;
```

→ पहले इन useRef को useEffect ने use कर दिए हैं तो useRef last wali value उठाएगा DOM से and current value update होने से पहले ही useEffect चल जाएगा and last value को store कर लेगा

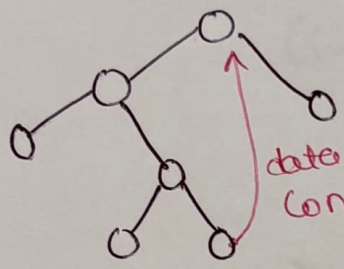
- data is send in form of props to User Component

CH: 12 CONTEXT API

#1 CONTEXT API

* CONTEXT API

- We use Context API, so that data can be transfered blw sibling component or components which are not child-parent
- you can use Context API, so that data can be trasfered from any component to other component, no matter what the Relation blw them



- you can also use Redex, so that data can be travelled anywhere blw components but Redex is complex :o we use Context API
- Provider: provider sets the data, मतलब क्या क्या data provide करेगा है पूरी application को
- Consumer: Consumer मतलब उस data को (जो provider provide करता है) उसे Consume करना

> Components
>> CommonContext.js
>> Main.js
>> UpdateButton.js

> App.js

> commonContext.js

```
import React from 'react'
Export const CommonContext = React.createContext();
```

> App.js

```
import React
import {CommonContext} from './Component/CommonContext'
import main from ./Component/main;
import UpdateButton from ./Component/UpdateButton';
class App extends 'React.Component'
{
    constructor()
    {
        super()
        this.updateColor = (color) =>'
        {
            this.setState(
            {
                color: color
            })
        }
        this.state ={
            color="green",
            updateColor : this.updateColor
        }
    }
    render(){
        return(
            <CommonContext.Provider value = {this.state} >
                <h1> Context API </h1>
                <main/>
                <UpdateButton/>
            </commonContext.provider>
        );
    }
}
```

>main.js

```
import React from 'react';
import {CommonContext} from './CommonContext'.
class main extends React.Component
{
    render ()
    {
        return(
            <CommonContext.Consumer>
            {
                ({color}) =>(
                    <h1 style={{backgroundColor : color}}>
                        Hello, this is main page
                    </h1>
                )
            }
            </CommonContext.Consumer>
        );
    }
}
export default main
```

>UpdateButton.js

```
import React
import {CommonContext} from './commonContext'.
function UpdateButton (){
  return(
    <CommonContext.Consumer>
    {  ({updateColor}) => (
            <div>
                <button onClick={()=> updateColor ('yellow')}>
                    Update Color </button>
            </div>
    )
    }
    </CommonContext.Consumer>
)
```

Working

*CommonContext.js
- It has the createContext() fun

*App.js
- We will make a provider जिसमे value data होगी जो consumer consume करेगा
- हम constructor बनाके state use करेंगे, so that हम button के click पर state change कर सके
- State change होगा हम color variable को update करेंगे
- updateColor() fun will get call onClick
- updateColor() मे color as a parameter आ रहा है, Button.js से color जो है तो parameter के through आ रहा है updateColor() fun मे

*main.js
- इसमे हमे color को consume करना है :: हम consumer को use करेंगे
- और color as parameter आएगा जिसकी value होगी जो App.js के color की value होगी
- हम style लगाएंगे a Background color को : color set कर देंगे जो App.js से आ रहा है
- हमने यहा पर color को consume किया

* Button.js
- We will create a button जिसके click पर updateColor() fun call होगा
- Button भी updateColor() fun को consume करेंगा
- हम updateColor मे onClick पर color pass करेंगे eg yellow & तो color App.js के updateColor() fun मे जाके save हो जाएगा