React

→ In 2011 by facebook (Jordan Walk)

→ React is a javaScript library which is used to create single page application.

Angular (frameWork)
↓
google
↓
2009
↓ Pattern
MVC Architecture

React (Library)
↓
facebook
↓
2011
↓ Pattern
Component based Architecture

Angular 2 (frameWork)
↓
Google
↓
2014
↓ Pattern
Component based Architecture

Vue (frameWork)
↓
Open Source
↓
2015-16
↓ Pattern
Component based Architecture

Pre Requister for React

* HTML
* CSS
* Javascript
  - → Arrow function
  - → Destructuring
    - → Rest
    - → Spread
  - → Methods
    - → map()
    - → filter()
    - → Reduce()
  - → DOM
  - → modules
    - → import
    - → Export
  - → Event

Modules

import

export

Default Export

In HTML

```
<script src="./Pakistan.js" Type="module"></script>
```

Export

In JS :- (India.js)

```
let product = "Tomato";
Export default Product;
```

Import

In JS :- (Pakistan.js)

```
import product from "./India.js"
console.log(product);
```

e.g:-(2)

for access more than one

```
let product1 = "Tomato";
let product2 = "Mango";
let exportable = { Product1, Product2 }
export default exportable
```

→ we can not write default keyword more than one in one js file. that's why we create an object.

e.g:-(3) ~~India.js~~ # Named Export

India.js

```
export let product1 = "Tomato";
export let product2 = "Mango";
```

Pakistan.js

```
import { product1 } from "./India.js";
console.log (Product1)
import { Product2 } from "./India.js"
console.log (Product2);
```

React :-

→ React is a javaScript library which is used to create single page application.

Library :-

→ Library is a set of predefine codes.
→ Libraries are light weight
→ We need to follow less rules and regulation when we are using libraries.

frameWorks :-

→ FrameWorks are collection of libraries.
→ FrameWorks may contain some necessary as well as unnecessary

libraries.

→ We need to follow more rules and regulation when working with frameWork.

→ FrameWork are not light weight, Because it contains multiple libraries.

Q What is single page application?

→ A single page application is a application that doesn't required page reloading during use.

e.g:— Gmail, git hub, facebook, instagram, whatsapp.

Advantage of Single Page Application.

① Single pages application (SPA) are faster because most of the resources are loaded only once through out the life span of the application.

② The development the single page application is faster and simplified.

③ Single page application are easy to debug.

Disadvantage of SPA.

① SPA are not 100% suitable for SEO (serve engine Optimization) purpose.

② Compare to multi page application (MPA) SPA are less secure because of cross site scripting language (XSS).

Default

export

let a = 10
let b = 20

① Single default export :—

export default a

② Default export for multiple values

export default {a, b}

import

① Single default import :—

import anyName from "path"

② default import to multiple values

import anyName from "Path"

(Here anyname will be an obj)

Named

export

export let a = 10;

export let b = 20

In html

```
<script src = " " Type = "module">
</script>
```

import

① Single import

import {a} from "path"

↑ we should write same name as exported

② Multiple import

import {a, b} from path.

Features Of React

1. Declarative in Nature
2. Component Based Architecture
3. Write one use Anywhere

History of React

	Angular	React	Angular 2	Vue
Year	2009	2011	2014	2015-16
Name	Angular (framework)	React (library)	Angular 2 (FrameWork)	Vue (fameWork)
Owner	Google	Facebook	Google	OpenSource
Pattern	MVC Architecture	Component based Architecture	Component B.A	component B.A
language	Javascript	Js, TS	T.S	J.S, T.S

Modules

→ Modules are the small block or reusable code.
→ Modules, allow us to break our code into seperate files, and this makes us easy to maintain the code base.
→ Modules are called module only when we mention type = "Module"

Syntax

```
<script src="demo.js" Type="module"></script>
```

We have two types of Import and export of modules

1. Default export
2. Named export

(1) Default Export

(i) When we have only one value to export.

```
let a = 10
export default a
```

Default import

```
import anyName from "Path"
c.log(anyName)
```
→ o/p = 10

(ii) When we have multiple value to export.

```
let a = 10
let b = 20
export default {a,b}
```

Default import

```
import anyName from "Path"
c.log(anyName)
```
→ o/p = {a: 10, b: 20}.

(2) Named Export

(i) When we have only one value to export

```
export let a = 10.
```

Named import

```
import {a} from "path".
```

(ii) When we have multiple value to export

```
export let a = 10
export let b = 20
```

Named import

```
import {a,b} from "path"
```

→ We are using import and export keyword to acheive that functionality respectively.

→ We can do only default export only once, per file.

→ We can do multiple named export.

→ We should use import on the top of the file and export on the bottom of the file.

(1) Declarative in Nature :

(2) Difference between Declarative and imperative

Features of React

Declarative

(1) It is something that is already define and we just have to use it

(2) In react we have some predefine thing such as hooks and npm libraries.

Imperative

(1) It is something where you have to implement everything such as we need to use multiple dom methods.

(2) Component Based Architecture :-

→ We are spreading our code in multiple component.
→ Each component can have its own scope in which we will manage the states and props.

(3) Write Once Use Anywhere :-

→ As we are following component based architecture we can achieve the reusability of code.

→ The component onces declare can be used any time any where.

cmd (Create folder for React)

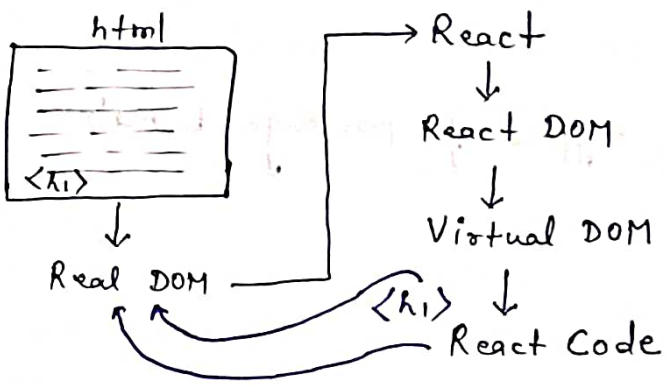
```
react> npm create vite@latest
Project Name :- m19React
Package Name :- m19React
Framework :- React
Variant :- JavaScript
react> cd m19React
react> npm install
after installation.
```

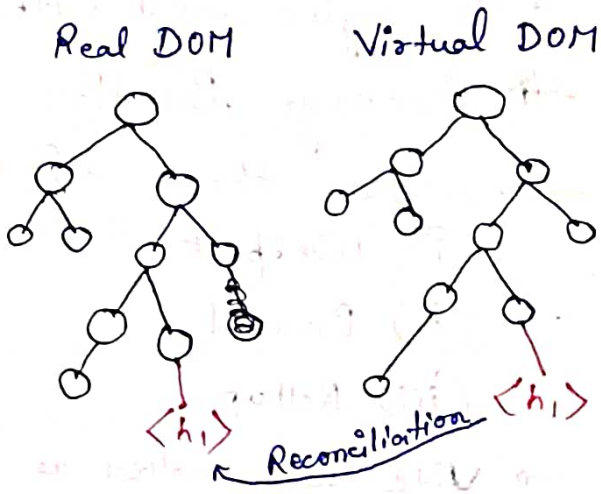
```
m19React> code . . npm run dev
```

Terminal

→ npm run dev
Copy url and pop paste in browser.
↳ local

How React Works :-





→ Here browser will create a Tree (DOM) like structure which is called as real DOM.

→ That tree like structure / Real DOM is used by react and provided to react DOM library.

→ React DOM library with will create copy of Real DOM which is called as Virtual DOM.

→ That Virtual DOM will be used to apply changes made with react.

→ The changes made in virtual DOM we need to implementing real DOM.

→ The changes compared between the virtual DOM and Real DOM, that comparision process is called as Diffing Algorithm.

→ The changes found in diffing alogrithm will be implemented in real DOM, and that process is known as Reconciliation process.

Ajax → Asynchronous javascript and xml.

Virtual DOM :-

→ Virtual DOM is the light weight representation of the Real DOM.

→ whenever we are doing any changes it will get reflected in virtual DOM first and then only into the real DOM.

→ Whenever there are any changes instead the component will re-render.

→ The engine used with react is called as fiber.

Package Bundler :-

→ We have three different types of package bundler

(i) Webpack

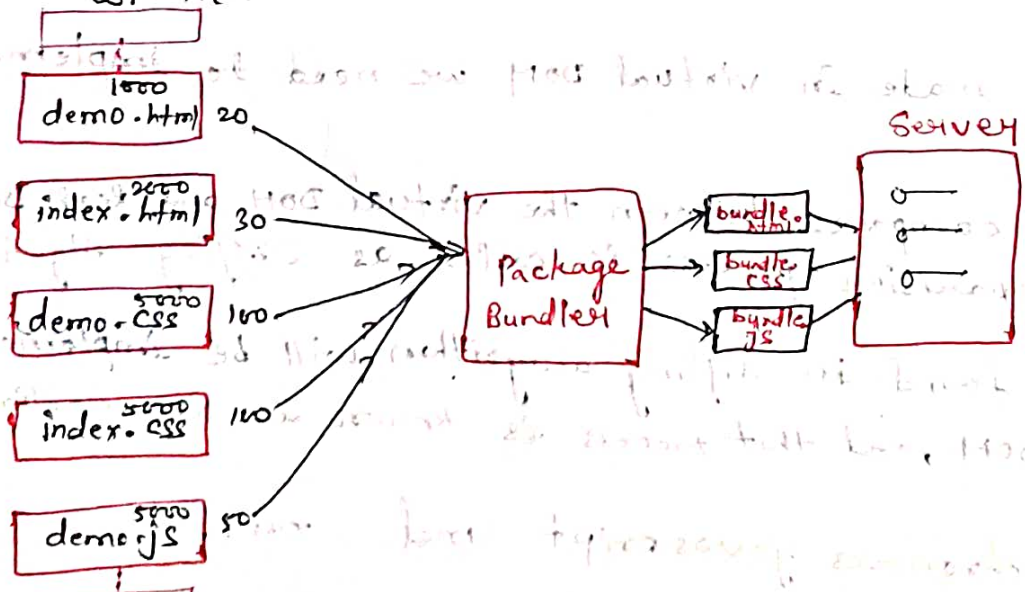
(ii) Parcel

(iii) Rollup

→ Vite uses roller as default package bundler.

→ It is used to bundle the multiple files.

→ When we don't use package bundler individual request to each files will get send to the server, which will increase the band width, to avoid that package bundler will helps, and it will bundle the multiple files and that bundle will get send to the server. which will help to reduce the band width.



Babel :-

→ It is javascript transpiler. which converts modern javascript code into a version that is compatible with all browsers.

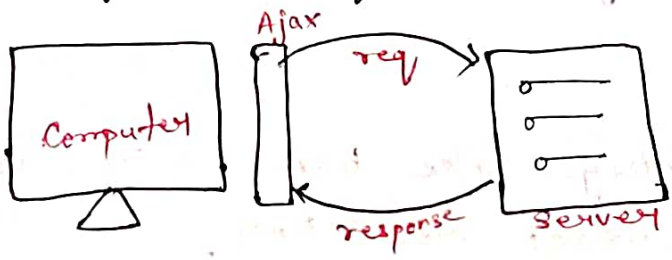
→ Babel enables us to use latest javascript code in our component.

Ajax

(Asynchronous javascript And Xml)

→ It is responsible for asynchronous behaviour in javascript.

→ when you send request to the server from browser, multiple request will get send to the server which contains most of unnecessary request. To avoid such unnecessary request Ajax will help, and it will send only required ~~request~~ necessary request to the server.



Search google → ~~tailwind~~ ~~vite~~ tailwind vite → Terminal → copy code → Paste int vs → Terminal (Paste here).

In vs code → paste → Tailwind.code (copy) from browser ↓ Tailwind Vite

Rafce (boiler plate) for jsx.

(Tailwindcss intellisense) install in vs code.

Render Method in React.

Class based component (shortcut for boilerplate → rcc)

what is class in javascript.

document.createElement("h1", null, "Hello World")
Tag, Attribute, Text
↳ used for read ~~html~~ html code which is written in jsx.

App = () ⇒ {

① return ——

② return (

or
<>
</>
)

}

fragment
"empty element" for wrapping other element

<fragment Key={}>
</fragment>

React # Folder Structure

→ when we installed react with vite we have some default folder structure which contains several files and folder and these are as follows :-

(1) Node Modules :-

→ This folder contains all the packages that we have installed during react installation as well as it will also contain the packages we will install in future for our application.

for example :-

default Packages like babel, vite, eslind, rollup, react, react DOM.

(2) Public :-

→ It is used for static things like images, logos and svg

(3) Index.html file :-

→ when working in react we have only one html file that is index.html.

→ It contains two main html elements i.e (i) & (ii).

(i) `<div id = "root" ></div>`

→ Div contains an id called root and which is used to create an root in main.jsx file with the help of "document.getElementById" and "react DOM.CreateRoot()"

(ii) `<script type = "module" src = "/src/main.jsx" ></script>`

→ This tag is used to link the main.jsx with our html file and it also contains type = "module" so that we can do import and export of modules as well as components in react.

③ Package-lock.json :-

→ This file contains all the dependencies as well as dev-dependencies and other packages also in the form of json.

④ Package.json :-

→ This file contains the dependencies and dev.-dependencies.

⑤ Vite.config.js :-

→ This file contains the configuration of Vite installation.

⑥ # Src Folder

→ What content we want to display on the UI and all the remaining code we will write inside src folder only.

→ except src folder we will not write anything outside it or outside src folder.

→ src folder contains react components and css files as well as all the assets.

Main.jsx :-

→ In that file we create the root with the help of reactDOM.createroot() and this file import the App.jsx file/component.

→ In main.jsx we import css files so that css will be apply to entire application.

JSX

→ JSX stands for Javascript xml.

→ It is similar to the html but it is not html, but it is javascript only.

→ Using JSX we can write html look like code which will be internally read as javascript as follows.

 App ← JSX

let div = document.createElement("div", null, "App")

↑ How it internally looks like.

Tag, Attribute, Text

→ We should follow some rules while writting the jsx.

rule (1) jsx tag names should not contain upper case letter.

e.g:- App ✓

<Div> App </Div> → error.

<DIV> App </DIV>

rule (2) The element written with first letter upper case will be consider as component.

${ } → template literal

 Hello ✓

 ×

</br> ✓ ⇒

<img src = " " > ×

 ✓

rule (3) In jsx if we are writting more than one element (these element should have atleast one parent element)

```
# return (
    <div> Hellow </div>
    <div> world </div>
  )
```
→ wrong statement

```
# return (
    <div>                          ← Parent element
      <div> Hello </div>
      <div> World </div>
    </div>
  )
```
→ Right statement

rule (4) if you are writting / if you want to return only one element we can go with implicit return.

```
const React = () ⇒
      <div> hello </div>
```

→ we can write one line statement.

```
return (
    <div> {
        arr.map(val ⇒ <h1> {val} </h1>)
    } </div>
)
```

Fragment in React

→ When we don't want to create extra Node be used fragment.

→ It is the feature which comes from react library which allow as to group multiple element without array extra Node in DOM.

fragment Improve code readability :-

→ We can write empty fragment when we don't want to used key attribute.

```
<>
<h1> Hello </h1>
</>
```

→ We can used fragment with fragment keyword.

```
<fragment>
    <h1> Hello </h1>
</fragment>
```

Do import fragment

```
<React.fragment>
    <h1> Hello </h1>
</React.fragment>
```

No need import the fragment

import {fragment} from 'react'.

React.fragment with key —

```
<fragment key={}>
    <h1> Hello </h1>
</fragment>
```

or

```
<React.fragment key={}>
    <h1> Hello </h1>
</React.fragment>
```

What is Key ?

→ It is an special attribute that needs to be include while writting in the loop.

→ It help react to identify which item in loop is interity.

```
const JsxIntro () => {
    let arr = [10, 20, 30, 40, 50]
    return (
        <div>
            {
                arr.map ((val, i) => {
                    return (<fragment key={id}>
                        <p> {val} </p>
                    </fragment>)
                })
            }
        </div>
    )
}
```

What is jsx expression ?

→ While writting jsx (Html structure), if you want to used java script along with it we can used jsx expression which allowed as to java Script in jsx.

```
let price = 100
return (
    <div> The price of the product is {price}.Rs </div>
                                      ↑
                                  Jsx expression.
```

→ We can write jsx expression with the help of curly braces.

→ Inside jsx expression we should not write any looping statement exp- accept map, filter and reduce.

→ Inside jsx expression we should not write any conditional statement except ternary condition.

```
Import React from 'react'
const JsxIntro = () => {
  let arr = [10, 20, 30, 40, 50]
  return (
      <div> {
          arr.map (() => {
              let value = false;
            })
      } </div>
    )
}
export default JsxIntro.
```

⇒ rce (short cut for ~~component~~ class based component boilerplate)

Class Based Component

Difference between function based Component and class based Component.

(1) function based Component

(1) It is a type of component which is represented by arrow function

(2) It has return statement which returns the JSX

(3) It does not has default state, to acheive the state in FBC we use hooks such as useState hook.

(4) FBC has hooks

(5) We use hooks in FBC to acheive life cycle methods.
for example :- use effect hook

(6) It is simple to use and recommended by most of the companies.

```
import React, {useState} from 'react'
Const StateIntro1 = () => {
  let [aparna, setAparna] = useState("tinkku")

  let handleName = () => {
      setAparna("Aparna")
  }
  return (
      <div>
        <h1> name : {aparna}</h1>
        <button onClick={handleName}> Change Name</button>
      </div>
```

Class based Component

(1) It is a type of component which is represented by class (class keyword).

(2) It has return statement which return the JSX but return statement must be render().

(3) It has default state management with state keyword and we can manage the state with the help of ~~state~~ setState().

(4) CBC doesn't has hooks.

(5) CBC has predefine life cycle methods such as constructor(), render(), ComponentDidMount(), componentDidUpdate().

(6) It is harder to read and almost depricated.

```
import React, {Component} from 'react'
export default class ClassBasedComp extends
  Component {
    constructor() {
        super()
        this.state = {
            count: 0
        }
    }
    handleCount = () => {
        this.setState({count: this.state.
            count+1})
    }
    render() {
        return (
            <div>
              <h1> Count : {this.state.count}
              </h1>
              <button onClick={this.handle
              Count}> Increment </button>
            </div>
        )
    }
}
```

UseState hook :—

→ The useState hook is build in react hook that allows functional component to manage the state without using class components.

Syntax :—

```
const [state, setState] = useState(initialValue);
```

→ We need to import useState hook from react.

e.g:- How to use :—

```
const [count, setCount] = useState(0)

<button onClick={() => { setCount(count + 1) }}>
    increment </button>
```

Rules for Writting Hook :—

→ Hooks are available only in function
→ Hooks are the javascript function.

Rules

(1) Hooks should always will called / declared on the top of the component.

Rule (2) You can't declared hook inside loops, nested functions, conditional statements etc.

e.g:-
```
let increment = () => {
    // const [first, setfirst] = useState(second) // wrong
}
```

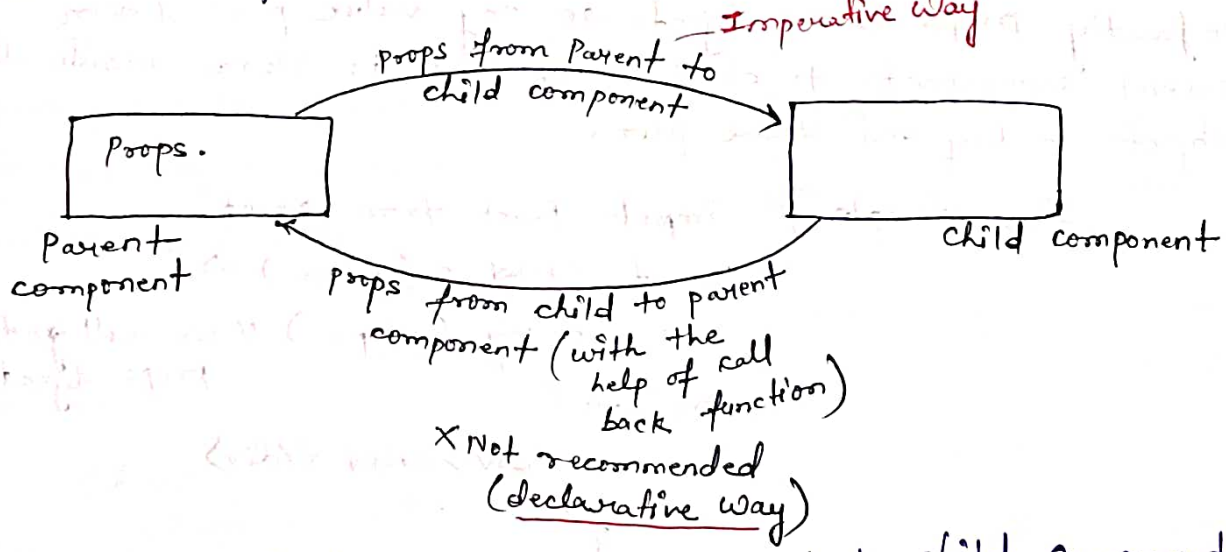
Rule (3) → Hook should not be used after return statement

Rule (4) → avoid using hooks in function passed to the use-Callback and useMemo hook.

Props

What is props in react?

→ Props are nothing but the properties.
→ Props are used to transfer the data from parent component to child component.
→ Props follows uni-directional data flow that is we can send props from parent component to child component.
→ We can acheive bi-directional data flow with the help of call back function but is not recommended.
→



→ Props are send from parent component to child Component is called as Imperative Way and accepting props from child component to parent component with the help of call back function is called as declarative way which is not recommended.

How to send props from parent Component.

→ At the place we write the html attribute we have to send the props.

In parent Component

```
<child props={data}></child>
```

→ when we have passing string ~~prot~~ data type as a props we can send without jsx expression { } but for the remaining data types we have to pass them inside the JSX.

```
<child props = "Aparna"></child>.
<child props = {any other datatype}></child>
```

~~for display in Web Page~~
~~const JSX Intro Day~~

→ when we are passing more than one value as a props we have to wrap them in an object.

. <child props={{var a, var b, func}}></child>

How to access props inside child component.

→ At the place of we are write the argument of the call back arrow function of function based component, we have to write / accept the props.

→ Defaultly props is an object so any value pass from parent component to child component is store inside the object in key and value pair.

for example ;-

```
import React from 'react'
const child = (props) => {
    console.log(props) // we will get props object
    return (
        <div> child </div>
    )
}
export default child
```

ⓓefault Props

→ when we don't pass props from the parent component but we are trying to access the props inside child component at that time we might get an empty object or undefined and because of that our UI will look blank.

To avoid that we have to use default props. along with the default props we can pass as some fall back UI. Using which our UI will not look blank and it will display the fall back UI, provided by us.

e.g:- ① import React from 'react'

```
const PropsIntro1Child = ({ props = "please pass some props..." }) => {
    console.log (props)

    return (
        <div> PropsIntro1Child </div>
    )
}
export default PropsIntro1Child
```

e.g:- ②

Parent

```
import React from 'react'
import PropsIntro1Child from
    './propsIntro1Child'

const PropsIntro1 = () => {
    return (
        <div>
            <propsIntro1Child>
                <h1> i'm heading </h1>
            </propsIntro1Child>
        </div>
    )
}
export default PropsIntro1
```

Child

```
import React from 'react'

const PropsIntro1Child = (props)
    => {
    console.log(Props.children.props.
        children)

    return (
        <div> PropsIntro1Child
            </div>
    )
}

export default PropsIntro1Child
```

What is Props.children ?

→ If we write JSX in between opening and closing tag when we have calling the child component inside parent component. At that time to access the JSX inside child component we have to use props.children

Difference between state and props :-

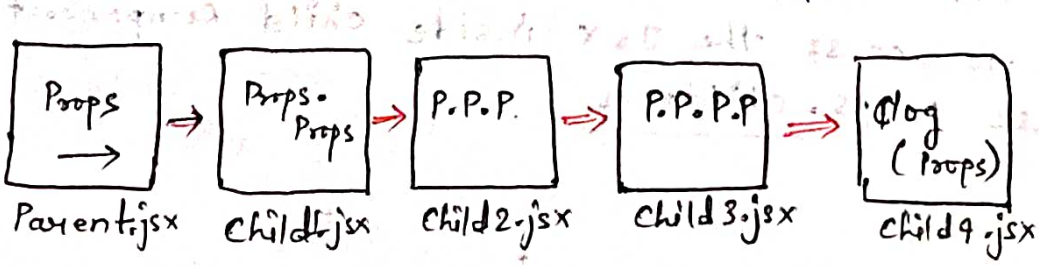
State	Props
(1) States are mutable because we can update the state.	(1) Props are immutable because we can't update the props until and unless we are sending state as a props.
(2) states are not available in state less component, for ex:- (function based component). for exam (we need to take the help of hooks to access the state)	(2) props are available in stateless as well as state full component. for example :- class based component
(3) We can read, write and update state.	(3) We can only read the props.

Props Drilling

→ * To pass some props from nth parent to its any nth child we should pass the props from each and every component ~~comm~~ coming in between which is called the props drilling.

→ To avoid props drilling we can use state management tool such as context API, Movex, redux.

→ It is not recommended to follow props drilling for more than two to three component.



→ Props drilling is an unofficial term use for passing the data through out several nested children components.

Event handling in React / Synthetic Event in React

→ React uses Synthetic event.
→ Synthetic event are much similar to the DOM event.
→ There are some minuar differences between synthetic event and DOM event.

(1) we should follow the camel casing for writting the event Name.
e.g:- DOM event → onmouseover
Synthetic Event → onMouseOver

(2) we should not pass the function name within codes (' ', " "), in synthetic event we pass function name within { } JSX expression.
e.g:- DOM event → onclick = "handleclick" ✗
Synthetic Event → onClick = {handleClick} ✓

(3) while writting the function name with the event no need to call the function with parenthesis ().
e.g:- DOM → onclick = "handleclick()" ✗
Syntho. Event → onClick = {handleclick} ✓

(4) If we want to pass some parameters during function call we should write the function name within the call back function.
e.g:- DOM → onclick → "handleclick(a,b)"
Synth. Event → onClick = {() => {handleClick(a,b)}}

(5) no need to write addEventListner.

(6) To avoid the default behaviours of ~~props~~ form and anchor tag of refreshing the page we can use

we can use :- e.preventDefault().

(7) We have multiple Synthetic Events.

e.g:- onClick, onSubmit, onDoubleClick, onScroll, onMouseEnter, onMouseLeave, onKeyPress, onKeyDown.

(8) We can access the values in input field with e.target.value.

(9) We can style the element with e.target.style

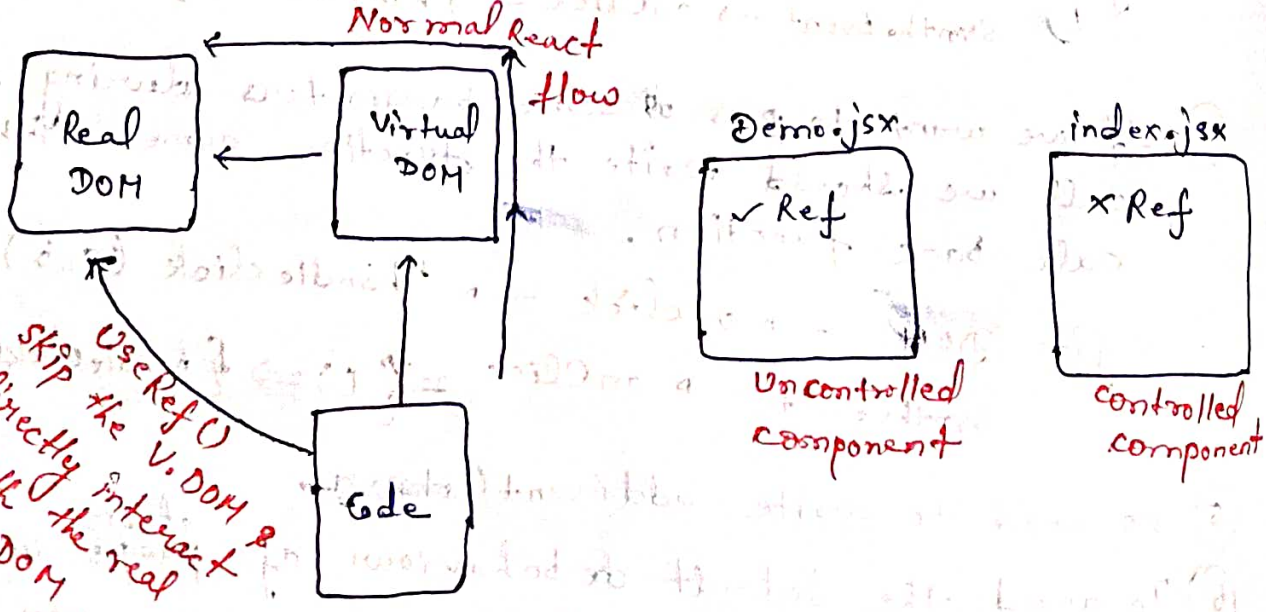
UseRef Hook ()

→ In some senario we need to skip the virtual dom and directly interact with the real dom, in such senario we will use useRef () hook.

for example :- multimedia playback, focus, text suggesstion.

→ It is an imperative way and not the react declerative way.

→ We should avoid using Refs because it breaks the normal working flow of react, it means. when we are using Refs we skip the virtual dom and directly interact with the real dom.



Controlled Components or forms

1. The components in which we don't use refs are called as controlled components.
2. Controlled components follows the normal react working flow, i.e

   code → Virtual DOM → Real DOM
3. controlled component doesn't have refs to send as a props.
4. It is the best practice to have controlled components.
5. Generally application are faster.
6. Less complex code.
7. Easy to debug.

Uncontrolled Components or forms

1. The components in which we use refs are called uncontrolled components
2. The uncontrolled component doesn't follow the normal react working flow and it skips the virtual DOM.

   code → Virtual DOM → Real DOM
3. If we send refs as a props to child component we get error.
4. We should avoid refs because it is not recommended practice.
5. Refs will make our application slower.
6. Complex code.
7. Hard to debug.

Context API

→ It is one of the state management tool it is introduce after react 16.3.

→ It is used to avoid props drilling.

→ Context API allows us to pass the data anywhere throughout the component tree.

APP.jsx

1. import the createContext variable
2. use the useContext() hook along with C.C.Var and store it in new var.

App.jsx

```
import myContext from ---
import { useContext } from react

const APP = () => {
  let allData = useContext(myContext)
    c.log(allData)  // 10 kg
  return (
      <div>
        APP
      </div>
    )
}
```

To create Context Component

1. Create separate component for context.
2. import createContext() from react and store it in one variable and export it
3. Declare all states, variable and all things in context
4. Destructure the children / props.
5. Using the createContext variable, use provider method.
6. pass All the variable along with value attribute to the all childrens.

AppContext.jsx :-

```
① import {createContext} from 'react'.
  export let myContext (7) = createContext().

const AppContext = ((4){children}) => {
  ③ let sugar = "10 kg"

<myContext.Provider (5) value = (6){sugar}>{children}
</myContext>}
```

export default AppContext.

Main.jsx

① Wrap App.jsx within the context component

```
<AppContext>
    <App/>
</AppContext>
```

Life Cycle Methods in React

→ It refers to the set of methods that are called and various points in life cycle of component.

→ These methods allows you to perform certain actions during different stages of component existence such as when it is created, updated, destroyed.

→ Some hooks are introduces after react 16.8 that helps to achieve life cycle methods in function based component.

→ In life cycle methods we have three phases

① Mounting Phase :-

(a) constructor ()
(b) getDerivedStateFromProps ()
(c) render ()
(d) componentDidMount ()

② Updating Phase :-

(a) getDerivedStateFromProps ()
(b) shouldComponentUpdate ()
(c) getSnapshotBeforeUpdate ()
(d) componentDidUpdate ()
(e) render ()

(3) (b) Unmounting :-

(a) componentWillUnmount () {
clearSetTimeOut()
}

(1) Mounting Phase

→ It is also called as birth phase.

(a) Constructor () :-

→ It is the best place to initialize the state and bind this keyword

→ It is optional

→ Constructor loads only once in entire life cycle.

→ It is mandatory to use super() statement into the constructor.

(b) getDerivedStateFromProps () :-

→ This method is invoked (call) during mounting phase and updating phase also.

→ This method gets called after constructor and before render.

→ If we want to make any changes into the state before render we can do inside this method.

(c) render () :-

→ It is called after getDerivedStateFromProps().

→ This is required method.

→ This method is used to display a content on the UI.

→ We write JSX inside the return statement which is present inside render method.

(d) componentDidMount() :-

→ This method get called after render method.

→ This method get called only once in entire life cycle.

→ Inside this method we write asynchronous this such as API call, fetching, promises, setTimeOut and setTimeIntervals.

(2) Updating Phase

→ This phase express when user perform any action such as click on the button to add the product into the cart.

→ Updating phase has multiple methods like ~~get derived~~

(a) getDerivedStatesFromProps()

(b) shouldComponentUpdate()

(c) getSnapshotBeforeUpdate()

(d) componentDidUpdate()

(e) render()

(a) getDerivedStatesFromProps() :-

→ This method gets invoke (call) in mounting phase also and updating phase also.

→ when over user perform any action this method text the state values and tries to update them.

(b) shouldComponentUpdate() :-

→ This method get called before component gets updated and it returns boolean value, if value is true, it tends that component should be updated.

→ If it returns false it tends that no need to update the component.

(c) getSnapshotBeforeUpdate() :-

→ This method get called when shouldComponentUpdate return true.

→ This method simply keeps the snapshot of the previous state.

for example :- current state is count : 1

and getSnapShotBeforeUpdate() will keep the snapshot of previous state also that is ~~count column~~ count : 0.

(d) componentDidMount() :-

→ This methods get called once the component will updated.

→ It is used to handle the side effect, caused by asynchro-rous things like API fetching, promises, setTimeOut & setIntervals.

(e) render() :-

→ This methods get called in mounting phase also and in updating phase also.

→ It is used to render the content on the UI.

(3) Unmounting Phase :-

componentWillUnmount() :-

→ This method is used to clear the events such as intervals. (setTimeOut & setInterval) canceling network request and removing addEventListner.

→ This method gets called just Before the component is unmount or remove.

UseEffect hook

→ UseEffect is used to perform sideEffects in function based component. Such as fetch API, Manually changing the DOM & some other Asynchronous things.

→ Syntax for useEffect :-

```
useEffect ( () => { }, [ ])
```

( ) => { } : Call back function

[ ] : dependencies

→ When we have empty dependency the call back function will get called only once during the rendering of component. So that we can acheive one of the life cycle method - that is componentDidMount().

→ when we have passing any dependency call back function will get called each time when there are any changes in dependency. So that we can acheive one of the life cycle method that is componentDidUpdate()

→ when we have using return statement inside call back function this return statement get executed before the component is unmount, in that return statement we can clear the intervals setTimeOut, clear networks and remove event listerners. So that we can acheive one of the life cycle method that is componentWillUnmount().

Create another folder :-

Project Name :- on19CardsApi

Install new library :- npm install axios

What is memoization ?

→ Memoization is an optimization technique use primarily in computing to speed up programs, by catching the results of expensive function calls and returning the catched result when the same input occur again this helps to improve performance.

UseMemo Hook :-

→ useMemo Hook is a react hook used for memoizing expensive computations so that they can be catched and reused across every rerender.

→ It is used to acheived ~~memeo~~ memoization.

→ It helps optimize performance by preventing unnecessary recalculations.

e.g:-

```
Const UseMemo = () => {
    const [count1, setCount1] = useState(0)
    const [count2, setCount2] = useState(0)

    // without useMemo Hook
    let checkCount1 = () => {
        let i = 0
        while (i < 2000000000)
            i++
        if (count1 % 2 == 0) {
            return "Even"
        }
        else { return "odd" }
    }

    // with useMemo Hook
    let value = useMemo (() => {
        let i = 0
```

```
    while (i < 2000000000) {
      i++
    }
    if (count1 % 2 === 0) {
      return "Even"
    } else {
      return "odd"
    }
  }, [count1])

  let checkCount2 = () => {
    if (count2 % 2 === 0)
    {
      return "Even"
    }
    else
    {
      return "odd"
    }
  }

return (
  <div>
    <button onClick={() => { setCount(count1+1) }}> count1 : {count1} {value} </button> <br />
    <button onClick={() => { setCount2(count2+1) }}> count2 : {count2} {checkCount2()} </button>
  </div>
)
}
export default UseMemo1
```

useCallBack Hook :-

→ Use Callback Hook is used to memoize functions, preventing unnecessary rerenders of child component.

Syntax

```
const reference-variable = useCallback(() => {
    // some execution
}, [dependency])

reference-variable()
```

→ useCallback Hook is used for performance optimization.

→ It is specially usefull when passing a callback function to child component and to prevent them from re-rendering when the parent component re-renders.

e.g:-

Parent Component

```
import React, { useCallback, useState }
from 'react'
import UseCallback1Child from
'./UseCallback1Child'

Const UseCallback1 = () => {
  Const [count, setCount] = useState(0)
// without useCallback hook
  Const increment = () => {
    setCount (count + 1)
  }
// with useCallback hook
  const increment1 = useCallback(
          () => {
            setCount(count + 1)
          }, [count])

return (
  <div>
    count : {count} <br />
    <UseCallback1Child func =
      {increment} ></UseCallback1Child>

<UseCallback1 child func = {increment1}>
  < UseCallback1 Child>
  </div>
) }
export default UseCallback 1
```

Child Component

```
import React from 'react'
Const UseCallback1Child =(
  { func }) => {

return (
  <div>
    <button onClick ={func}>
      increment </button>
  </div>
)
}
export default UseCallback1Child
```