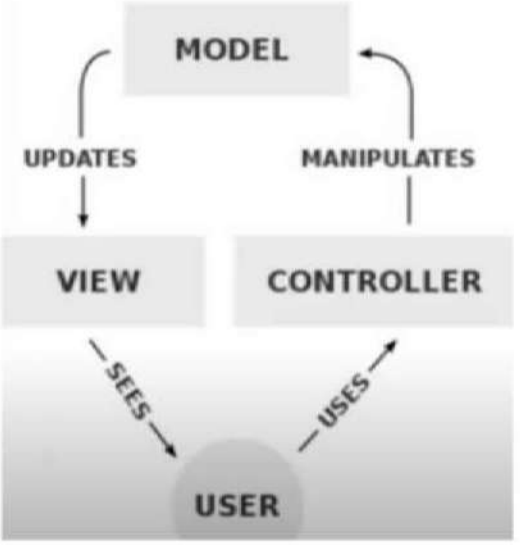
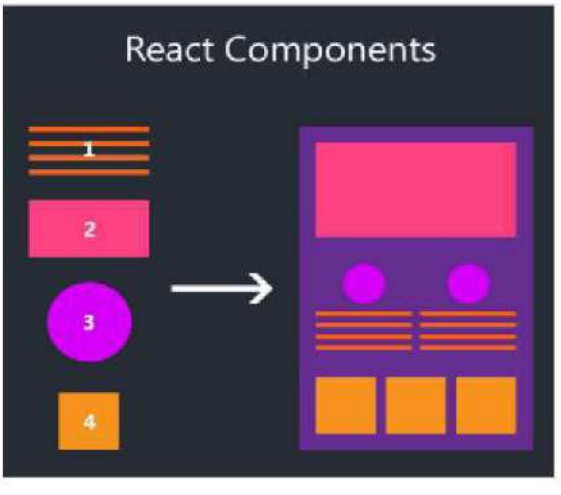
React Introduction

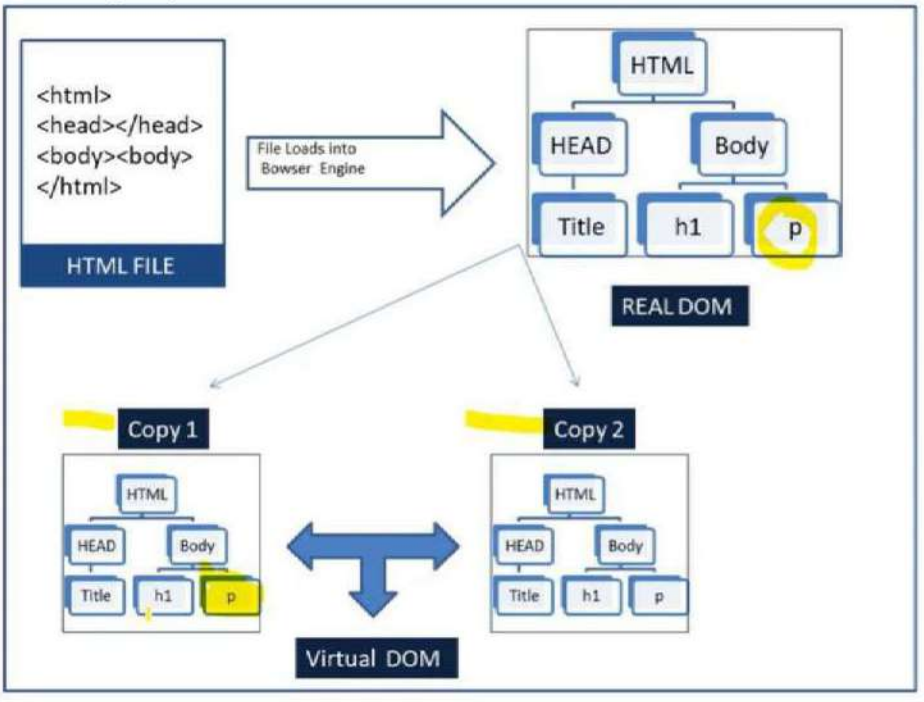
- For developing user interface.
- JavaScript library for building reusable User Interface components.
- It is an open-source, component-based frontend library responsible only for the view layer of the application.
- Most of the websites are built using MVC (model view controller) architecture.



- In MVC architecture, React is the 'V' which stands for view, whereas the architecture is provided by the Redux or Flux.
- A ReactJS application is made up of multiple components, each component responsible for outputting a small, reusable piece of HTML code. The components are the heart of all React applications.



- These Components can be nested with other components to allow complex applications to be built of simple building blocks.
- ReactJS uses virtual DOM based mechanism to fill data in HTML DOM.



- The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time.
- To create React app, we write React components that correspond to various elements. We organize these components inside higher level components which define the application structure.

Why we use ReactJS?

- To improves the speed of the apps.
- To improve the performance of the app.
- To improve readability and to maintain larger apps.

React Features

Currently, ReactJS gaining quick popularity as the best JavaScript framework among web developers. It is playing an essential role in the front-end ecosystem. The important features of ReactJS are as following.

JSX

JSX stands for JavaScript XML. It is a JavaScript syntax extension. Its an XML or HTML like syntax used by ReactJS. It extends the ES6 so that HTML like text can co-exist with JavaScript react code.

Components

ReactJS is all about components. ReactJS application is made up of multiple components, and each component has its own logic and controls. These components can be reusable which help you to maintain the code when working on larger scale projects.

One-way Data Binding

ReactJS is designed in such a manner that follows unidirectional data flow or one-way data binding. The benefits of one-way data binding give you better control throughout the application. If the data flow is in another direction, then it requires additional features.

One way Data Binding means the data can be passed from Parent component to Child component. But to pass the data from Child to Parent, we have to use some additional feature.

Virtual DOM

A virtual DOM object is a representation of the original DOM object. It works like a one-way data binding. Whenever any modifications happen in the web application, the entire UI is re-rendered in virtual DOM representation. Then it checks the difference between the previous DOM representation and new DOM. Once it has done, the real DOM will update only the things that have actually changed. This makes the application faster, and there is no wastage of memory.

Simplicity

ReactJS uses JSX file which makes the application simple and to code as well as understand. We know that ReactJS is a component-based approach which makes the code reusable as your need. This makes it simple to use and learn.

Performance

ReactJS is known to be a great performer. This feature makes it much better than other frameworks out there today. The reason behind this is that it manages a virtual DOM. The DOM is a cross-platform and programming API which deals with HTML, XML or XHTML. The DOM exists entirely in memory. Due to this, when we create a component, we did not write directly to the DOM. Instead, we are writing virtual components that will turn into the DOM leading to smoother and faster performance.

Advantage of ReactJS

1. Easy to Learn and USe

ReactJS is much easier to learn and use. Any developer who comes from a JavaScript background can easily understand and start creating web apps using React in a few days. It is the V(view part) in the MVC (Model-View-Controller) model, and referred to as ?one of the JavaScript frameworks.? It is not fully featured but has the advantage of open-source JavaScript User Interface(UI) library, which helps to execute the task in a better manner.

2. Creating Dynamic Web Applications Becomes Easier

To create a dynamic web application specifically with HTML strings was tricky because it requires a complex coding, but React JS solved that issue and makes it easier. It provides less coding and gives more functionality. It makes use of the JSX(JavaScript Extension), which is a particular syntax letting HTML quotes and HTML tag syntax to render particular subcomponents. It also supports the building of machine-readable codes.

3. Reusable Components

A ReactJS web application is made up of multiple components, and each component has its own logic and controls. These components are responsible for outputting a small, reusable piece of HTML code which can be reused wherever you need them. The reusable code helps to make your apps easier to develop and maintain. These Components can be nested with other components to allow complex applications to be built of simple building blocks. ReactJS uses virtual DOM based mechanism to fill

data in HTML DOM. The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time.

4. Performance Enhancement

ReactJS improves performance due to virtual DOM. The DOM is a cross-platform and programming API which deals with HTML, XML or XHTML. Most of the developers faced the problem when the DOM was updated, which slowed down the performance of the application. ReactJS solved this problem by introducing virtual DOM. The React Virtual DOM exists entirely in memory and is a representation of the web browser's DOM. Due to this, when we write a React component, we did not write directly to the DOM. Instead, we are writing virtual components that react will turn into the DOM, leading to smoother and faster performance.

5. The Support of Handy Tools

React JS has also gained popularity due to the presence of a handy set of tools. These tools make the task of the developers understandable and easier. The React Developer Tools have been designed as Chrome and Firefox dev extension and allow you to inspect the React component hierarchies in the virtual DOM. It also allows you to select particular components and examine and edit their current props and state.

6. Known to be SEO Friendly

Traditional JavaScript frameworks have an issue in dealing with SEO. The search engines generally having trouble in reading JavaScript-heavy applications.

ReactJS overcomes this problem that helps developers to be easily navigated on various search engines. It is because React.js applications can run on the server, and the virtual DOM will be rendering and returning to the browser as a regular web page.

7. The Benefit of Having JavaScript Library

Today, ReactJS is choosing by most of the web developers. It is because it is offering a very rich JavaScript library. The JavaScript library provides more flexibility to the web developers to choose the way they want.

8. Scope for Testing the Codes

ReactJS applications are extremely easy to test. It offers a scope where the developer can test and debug their codes with the help of native tools.

Disadvantage of ReactJS

1. The high pace of development

The high pace of development has an advantage and disadvantage both. In case of disadvantage, since the environment continually changes so fast, some of the developers not feeling comfortable to relearn the new ways of doing things regularly. It may be hard for them to adopt all these changes with all the continuous updates. They need to be always updated with their skills and learn new ways of doing things.

2. Poor Documentation

It is another cons which are common for constantly updating technologies. React technologies updating and accelerating so fast that there is no time to make proper documentation. To overcome this, developers write instructions on their own with the evolving of new releases and tools in their current projects.

3. View Part

ReactJS Covers only the UI Layers of the app and nothing else. So you still need to choose some other technologies to get a complete tooling set for development in the project.

4. JSX as a barrier

ReactJS uses JSX. It's a syntax extension that allows HTML with JavaScript mixed together. This approach has its own benefits, but some members of the development community consider JSX as a barrier, especially for new developers. Developers complain about its complexity in the learning curve.

React render() function

- React renders HTML in web page.
- It is done using ReactDOM.render() function.
- *render()* is the function where the UI gets updated and rendered. *render()* is the required lifecycle method in React.
- *render()* function is the point of entry where the tree of React elements are created.
- *Syntax:*
  import ReactDom from "react-dom";
  ReactDom.render(HTML_code, document.getElementById('root'));

- When a *state* or *prop* within the component is updated, the *render()* will return a different tree of React elements.
- If you use *setState()* within the component, React immediately detects the state change and re-renders the component.
- The virtual DOM concept is possible using this render(). It passes only those elements to the Real DOM which are updated.

Using React with HTML

Installation:

1. Install nodeJS
2. Open Command Prompt and check "node".
3. Check npm and npx commands

NPM: (Node Package Manager)

- First need to install node's library and then we can use it.
- More local storage requires.
- Global packages can be used.

NPX: (Node Package Execute)

- Direct node's library can be used.
- Less storage requires as compare to NPM.
- Local packages can be used.
- It always uses the latest versions.

Example:

- If we want to create react app then we can use :
  npx create-react-app student
- If we want to create react app using npm then it requires to first install create-react-app and then we can use this command.
  npm install -g create-react-app (to install)
  create-react-app student (to create react app- student)

Creating React App:

1. Create one folder where you want to store app related files.

2. Open above created folder in Command prompt.
3. Run "npx create-react-app app_name"
   Ex: npx create-react-app student
4. After completion of process, the app can be accessed through Editor (VSCode)

Working with React App:

1. Open student folder through VSCode
2. Click TERMINAL + NEW TERMINAL option to access terminal through VS Code.
3. Write "npm start" at terminal to run application.
   a. It creates local mini server which is accessed through our web-browser (http://localhost:3000).
   b. The default react app can be accessed through this localhost.
   c. The changes which we will make in app can be implemented through this localhost.
   d. The localhost needs to open first to run app.
   e. Local server detects the changes applied by us and do necessary updates in Web-site.
   f. To close the LocalServer : Press "ctrl+c" and press "Y".

File Structure with necessary files:

Folders:

- **Node_modules:** It includes all modules of NODE which can be used during development of app.
- **Public:** It includes "index.html" file which is important for creating React app.. React creates **Single Page Application**. It means only single web-page is there (i.e. index.html file).
- **Src:** It includes multiple files. It includes one "index.js" file. It passes component related information to the "index.html" file. This file is also includes render(). Multiple other files are created under this folder which are used for creating components. Per component – 1. .js file and 2. .css file can be created under this folder.

Write a program to Display Hello:

index.html file:

```
<!DOCTYPE html>
<html lang="en">
```

```
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"/>
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
   <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

index.js file:

```
import ReactDom from "react-dom";
import './index.css';
ReactDom.render(<h2> Hello </h2>, document.getElementById('root'));
```

index.css file:

```
body{
    background-color: rgb(238, 238, 162);
    color: rgb(239, 58, 8);
}
```

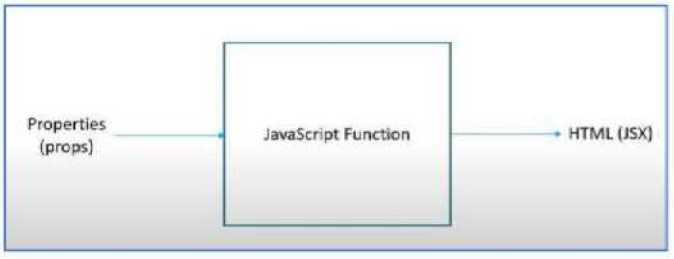
Components:

- The entire application is divided into a small logical group of code, which is known as components.
- A Component is considered as the core building blocks of a React application.
- Each component exists in the same space, but they work independently from one another and merge all in a parent component, which will be the final UI of your application.
- Every React component have their own structure, methods as well as APIs.
- They can be reusable as per your need.
- For better understanding, consider the entire UI as a tree. Here, the root is the Parent component, and each of the other components becomes branches, which are further divided into sub-branches.

Types of Components:

1. Functional Component:

- In React, function components are a way to write components that only contain a render method and don't have their own state.
- It is called stateless component because it never holds or manage states.
- They are simply JavaScript functions that may or may not receive data as parameters.
- We can create a function that takes props(properties) as input and returns what should be rendered (Means HTML block).
- render() is not used within functional component.
- It allows props as argument. It doesn't allow states.



- Syntax:

```
function component_name(props){
return(HTML-JSX block);
}
export default Component_name;
```

Write a program to Display Hello using parent component:

index.js file:

```
import ReactDom from "react-dom";
import App from "./App";

import './index.css';
ReactDom.render(<App />, document.getElementById('root'));
```

App.js file: (Parent Component)

```
import './App.css'
function App(){
    return(<h2 className='app'>Parent Component</h2>
);
}
export default App;
```

App.css file:

```
.app{
    background-color: aquamarine;
    color: brown;
}
```

- One component can be written within another component. (First component access through App component.)

Write a program to Display "Functional Component" using Child component:
App -> Fcomponent

App.js file: (Parent Component)

```
import './App.css'
import Fcomponent from './components/Fcomponent';
function App(){
    return (
        <div>
            <Fcomponent />
         </div>);
}
export default App;
```

Fcomponent Component: (Child Functional Component)

```
import './Fcomponent.css'
function Fcomponent() {
    return(<h3 className='fcomponent'>FunctionalComponent</h3>);
}
export default Fcomponent;
```

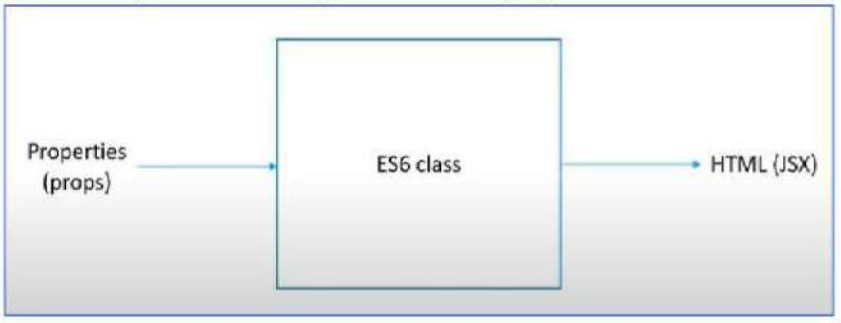
Fcomponent.css file:

```
.fcomponent {
    background-color:aqua;
    color:black;
}
```

2. Class Component:

- Class components are more complex than functional components.
- It requires you to extend from React Component and create a render function which returns a React element.
- It is also known as Stateful component. It can hold and manage states.
- You can pass data from one class to other class components.

- You can create a class by defining a class that extends Component and has a render function.
- Class component have options to use prop as well as to set states.



- Syntax:

```
class component_name extends React.Component{
        render() {
                return (HTML/JSX block);
                }
        }
```

Write a program to Display "Class Component" using Child Class component:
App ->Ccomponent

App.js file: (Parent Component)

```
import './App.css'
import Ccomponent from './components/Ccomponent';

function App(){
    return (
        <div>
           <Ccomponent />
        </div>);

}
export default App;
```

Ccomponent Component: (Child Class Component)

```
import React from 'react';
import './Ccomponent.css'
class Ccomponent extends React.Component{
    render(){
        return <h3 className='ccomponent'>Class Component</h3>
    }
}
```

```
export default Ccomponent;
```

Ccomponent.css file:

```
.ccomponent{
    background-color: black;
    color: aqua;
}
```

Multiple components can also be displayed:

Write a program to Display multiple components:
App -> Fcomponent & App -> Ccomponent

App.js file: (Parent Component)

```
import './App.css'
import Fcomponent from './components/Fcomponent';
import Ccomponent from './components/Ccomponent';

function App(){
    return (
        <div>
            <div className='app'>
                <h2> Parent Component</h2>
            </div>
            <div>
                <Fcomponent />
            </div>
            <div>
                <Ccomponent />
            </div>
        </div>);

}
export default App;
```

Nested components can also be allowed in React:

Write a program to display nested component:
App -> Fcomponent ->Ncomponent

App.js file: (Parent Component)

```
import './App.css'
import Fcomponent from './components/Fcomponent';

function App(){
```

```
    return (<Fcomponent/>);

}
export default App;
```

Fcomponent.js file:

```
import './Fcomponent.css'
import Ncomponent from './Ncomponent';
function Fcomponent() {
    return(<Ncomponent/>);
}
export default Fcomponent;
```

Ncomponent.js file:

```
import './Ncomponent.css'
function Ncomponent(){
    let rno=1
    let name='Het';
    let gender='Male';
    return (
        <div className='nbox'>
            <div className='nrno'>
                Rollno: {rno}
            </div>
            <div className='nname'>
                Name: {name}
            </div>
            <div className='ngender'>
                Gender: {gender}
            </div>
        </div>
    );
}
export default Ncomponent;
```

Ncomponent.css file:

```
.nrno{
    background-color: brown;
    color: aquamarine;
}
.nname{
    background-color: blue;
    color:aliceblue;
}
```

```
.ngender{
    background-color: aqua;
    color: black;
}
.nbox{
    padding:5px;
    background-color:rgb(217, 235, 235);
    margin: 2rem auto;
    width: 30rem;
    border: radius 12px;
    box-shadow:0 1px 8px rgba(0, 0, 0, 0.25);
}
```

Difference between Class and Functional Components

Functional Component	Class Component
• Functional Components is a plain JavaScript, you do not have a choice to set the state in functional component. • There is no render function we are using in functional components. • Functional components only accept the props as an argument. • Functional components are sometimes called stateless components.	• Class components we have a feature to set the set state in component. • In class components, we have a render function which is use to return the react elements. • In class components, we have both options use the props and set the state also. • Class components are sometimes called stateful components.

Passing data through Props

- Props means "**Properties**."
- Props are read only.
- It is an object which stores the value of attributes of a tag and work similar to the HTML attributes.
- It gives a way to pass data from one component to other components.
- It is similar to function arguments. Props are passed to the component in the same way as arguments passed in a function.
- Props are **immutable** so we cannot modify the props from inside the component.

Passing props through Functional Component:

- Under Functional component, props can be passed as attributes. And it can be accessed like function arguments.

Write a program to pass props through function components:
App -> Propsfunction

App.js file: (Parent Component)

```
import './App.css'
import Propsfunction from './components/Propsfunction';

function App(){
    return (
    <div>
        <div><Propsfunction rno={1} name1={'Het'} gender={'Male'}/></div>
        <div><Propsfunction rno={2} name1={'Heri'} gender={'Male'}/></div>
        <div><Propsfunction rno={3} name1={'Riya'} gender={'Female'}/></div>
    </div>);
}
export default App;
```

Propsfunction.js

```
import './Propsfunction.css'
function Propsfunction(props){
    return (
        <div className='propsfun'>
            <div>Rollno:{props.rno}  </div>
            <div>Name: {props.name1} </div>
            <div>Gender:{props.gender}</div>
        </div>
    );
}
export default Propsfunction;
```

Propsfunction.css

```
.propsfun{
    margin:3%;
    background-color: rgb(58, 125, 103);
    color: beige;
}
```

Passing props through Class Component:

- Under Class component, Props can be passed as attributes. It can be accessed using **this.props**.

Write a program to pass props through class components:
App -> Propsclass

App.js file: (Parent Component)

```
import './App.css'
import Propsclass from './components/Propsclass';
function App(){
    return (
    <div>
        <div><Propsclass eno={'E01'} name1={'Rajesh Patel'} /></div>
        <div><Propsclass eno={'E02'} name1={'Ronit Vaghela'} /></div>

    </div>);
}
export default App;
```

Propsclass.js file

```
import React from 'react';
import './Propsclass.css';
class Propsclass extends React.Component{
    render(){
        return (
            <div className='propsclass'>
                <div>Employee No: {this.props.eno}</div>
                <div>Employee Name: {this.props.name1} </div>
            </div>
        );
    }
}
export default Propsclass;
```

Propsclass.css file

```
.propsclass{
    margin-left: 30%;
    margin-right: 30%;
    margin-top:5px;
    background-color: rgb(48, 179, 231);
    color: rgb(247, 249, 249);
}
```

Conditional statements

IF condition / IF-ELSE condition:

- It is used to check the condition and if the condition is true, it returns TRUE part of IF otherwise else part of IF.
- IF is a statement and under JSX we can only pass expression. We can't pass any statement within JSX code.
- **Syntax:**

```
if(condition){
        return block;
}
else{
        return block;
        }
```

Write a program to display the employee name as per condition
App -> Ifcond

App.js file: (Parent Component)

```
import './App.css'
import Ifcond from './components/conditional/Ifcond';

function App(){
    return (
    <div>
        <div><Ifcond isdisplay={true} ename={'Megha Patel'} /></div>
        <div><Ifcond isdisplay={false} ename={'Jignasha Patel'} /></div>
        <div><Ifcond isdisplay={true} ename={'Mohita Patel'} /></div>

    </div>);
}
export default App;
```

Ifcond.js file

```
function Ifcond(props){
    const c=props.isdisplay;
    if(c){
        return <h2>Employee name is {props.ename}</h2>
    }
    return <h2>Sorry.....</h2>
}
```

```
export default Ifcond;

----------OR----------

function Ifcond(props){
    const c=props.isdisplay;
    if(c){
        return <h2>Employee name is {props.ename}</h2>
    }
    else{
    return <h2>Sorry.....</h2>
    }
}
export default Ifcond;
```

Inline IF condition (Using Logical Operator &&):

- Any expression of JavaScript can be written within JSX code using { }.
- Once we use && operator then it is a part of expression so we can use this within JSX code.
- The component is executed if the respective logical condition is true.

&& operator:

Operand 1 && Operand 2		Result
True	True	True
True	False	False
False	True	False
False	False	False
True	Expression	Expression
False	Expression	False

- Example: (Let a=TRUE and b=FALSE)

Operand 1	Operand 2	Result
a	a	True
a	b	False
b	a	False
b	b	False
a	<Component />	<Component />
b	<Component />	False

Write a program to check Inline IF condition and logical && operator
index -> App -> Inlinecond

Index.js file:

```
import ReactDom from 'react-dom';
import App from './App';
import './index.css';
ReactDom.render(<App logical={true}/>, document.getElementById("root"))
```

App.js file: (Parent Component)

```
import './App.css'
import Inlinecond from './components/conditional/Inlinecond';

function App(props){
    let t=props.logical;
    return (
    <div>
        <h1>Testing of Logical operator and use of Inline condition</h1>
        {t && <Inlinecond/>}
    </div>);
}
export default App;
```

Inlinecond.js file

```
function Inlinecond(){
    return <h2>You are in Inline Component</h2>
}
export default Inlinecond
```

Ternary operator (?:):

- Ternary operator (?:) can also be used for rendering purpose.
- It is also operator so the expression related with this can be written within JSX code.
- Syntax: condition? True_part : False_part
- Ex: If you want to render two different components as per the condition then ternary operator can be useful.

Write a program to check Ternary operator:
App -> Ternarycondtrue or Ternarycondfalse

App.js file:

```
import './App.css'
import Ternarycondfalse from './components/conditional/Ternarycondfalse';
import Ternarycondtrue from './components/conditional/Ternarycondtrue';

function App(){
    const check=true;
    return (
        <>
        <h1>Testing of Ternary operator</h1>
        {check? <Ternarycondtrue/> : <Ternarycondfalse/>}
        </>
    );
}
export default App;
```

Ternarycondtrue.js file:

```
function Ternarycondtrue(){
    return <h2>This is Ternary True Component</h2>;
}
export default Ternarycondtrue;
```

Ternarycondfalse.js file:

```
function Ternarycondfalse(){
    return <h2>This is Ternary False Component</h2>;
}
export default Ternarycondfalse;
```

Arrow function:

```
function fun_name(argument){
    return code;
}
```

Arrow function……..

```
fun_name = (argument) =>{
    return code;
}
```

Example:

```
function test(props) {
      let name=props.name;
      return <h1>Hello {name}</h1>
}
```

Arrow function:----------

```
const test = (props) => {
      let name=props.name;
      return <h1>Hello {name}</h1>
}
```

List:

- Lists are used to display data in an ordered format.
- In React, Lists can be created in a similar way as we create lists in JavaScript.

Write a program to display list:
App -> List

App.js file:

```
import './App.css'
import List from './components/List';
const App=()=>{
    return (<List />);
}
export default App;
```

List.js file:

```
const List = () =>{
    const item = ['TV','Laptop','Mobile'];
```

```
    return (
        <div>
            <h1>List of Products</h1>
            <h2>{item[0]}</h2>
            <h2>{item[1]}</h2>
            <h2>{item[2]}</h2>
        </div>
    );
}
export default List;
```

- map() function is used with list same as JavaScript.
- A map is a data collection type where data is stored in the form of pairs. It contains a unique key. The value stored in the map must be mapped to the key. We cannot store a duplicate pair in the map(). It is because of the uniqueness of each stored key.
- A single function can be passed as argument in map() which separately evaluated with all the values of the list using loop.
- Ex: a=[1,2,3]
  b=a.map(x=>x*x)
  Output: b=[1,4,9]

List.js file:

```
const List = () =>{
    const item = ['TV','Laptop','Mobile'];
    return (
        <div>
            <h1>List of Products</h1>
            {
                item.map(it=><h2>{it}</h2>)
            }
        </div>
    );
}
export default List;
```

- Multiple values related list can also be generated:

Write a program to display list:
App -> Listmultiple

App.js file:

```
import './App.css'
import List from './components/List';
const App=()=>{
    return (<List />);
}
export default App;
```

Listmultiple.js file:

```
const Listmultiple=()=>{
    let exp=[
        {   id:'1',
            title:'Daily Expenses',
            amount:500
        },
        {   id:'2',
            title:'Children Education',
            amount:50000
        },
        {   id:'3',
            title:'Home Loan',
            amount:15000
        }
    ];
    return(
        <div>
            <div>
                <h2>Title: {exp[0].title}</h2>
                <h2>Amount: {exp[0].amount}</h2>
            </div>
            <div>
                <h2>Title: {exp[1].title}</h2>
                <h2>Amount: {exp[1].amount}</h2>
            </div>
            <div>
                <h2>Title: {exp[2].title}</h2>
                <h2>Amount: {exp[2].amount}</h2>
            </div>

        </div>
    );
}
export default Listmultiple;
```

- Using map(), the same content can be written like: (Following code display the data in single line because with map() we can only pass single argument.)

Listmultiple.js file:---

```
const Listmultiple=()=>{
    let exp=[
        {   id:'1',
            title:'Daily Expenses',
            amount:500
        },
        {   id:'2',
            title:'Children Education',
            amount:50000
        },
        {   id:'3',
            title:'Home Loan',
            amount:15000
        }
    ];
    return(<>
        {
                exp.map(explist => <h2>Title: {explist.title}  Amount:
{explist.amount}</h2> )
        }
        </>
    );
}
export default Listmultiple;
```

- One more component is created in sale Listmultiple.js file and using mapping the whole object is passed as argument to the Display component.

Listmultiple.js file:----------

```
const Display=({expence}) =>{
    return(
        <>
            <h2>Title: {expence.title}</h2>
            <h2>  Amount: {expence.amount}</h2>
        </>

     );
}

const Listmultiple=()=>{
```

```
    let exp=[
        {   id:'1',
            title:'Daily Expenses',
            amount:500
        },
        {   id:'2',
            title:'Children Education',
            amount:50000
        },
        {   id:'3',
            title:'Home Loan',
            amount:15000
        }
    ];
    return(<>
        {
                exp.map(expence=><Display expence={expence}/>)
        }
        </>
    );
}
export default Listmultiple;
```

React Event:

- An event is an action that could be triggered as a result of the user action or system generated event.
- Event is user's action on which JavaScript can do some activity.
- Ex:
    - Clicking an element
    - Hovering element
    - Submitting form etc.
- Event handing in React is similar to JavaScript but with some syntactic difference:
    - React events are named using camelCase in place of lowercase.
    - With JSX, a function is passed as the event handler instead of a string.
    - Ex:
      HTML: <button onclick="eventHandle()"> Click Here</button>
      React:
        - Functional Component:
          <button onClick={eventHandle}> Click Here</button>
        - Class Component:
          <button onClick={this.eventHandle}> Click Here</button>

Event with Class Component:

- If arrow function is used for event handler then there is no need to bind **this** otherwise with function we have to bind **this** under **constructor()** explicitly.

Write a program to call onclick event:
App -> Eventclass

App.js file:

```
import './App.css'
import Eventclass from './components/Eventclass';
const App=()=>{
    return (<Eventclass />);
}
export default App;
```

Eventclass.js file:

```
import React from 'react';
class Eventclass extends React.Component {
    clickHandle=()=>{
        alert("Button Clicked");
    }
    render(){
        return(
            <div>
                <h2>Event Management</h2>
                <button onClick={this.clickHandle} >Click Here</button>
            </div>

        );
    }
}
export default Eventclass;
```

Event with Functional Component:

Write a program to call onclick event
App -> Eventfunction

App.js file:

```
import './App.css'
import Eventfunction from './components/Eventfunction';
const App=()=>{
    return (<Eventfunction/>);
```

```
}
export default App;
```

Eventfunction.js file:

```
const Eventfunction=(props)=>{
    const clickHandle=()=>{
        alert("You are accessing Google web-site now.")
    }
    return(
        <div>
            <h2>Hello {props.name}</h2>
            <a href="https://www.google.com/" onClick={clickHandle}>Visit
Google Website</a>
        </div>
    );
}
export default Eventfunction;
```

Passing arguments through Event:

Write a program to pass argument through event: Need to create function with onClick:
App -> Eventfunctionpass

App.js file:

```
import './App.css'
import Eventfunctionpass from './components/Eventfunctionpass';
const App=()=>{
    return (<Eventfunctionpass/>);
}
export default App;
```

Eventfunctionpass.js file:

```
const Eventfunctionpass=()=>{
    const clickHandle=(a)=>{
        alert('Thank you '+a)
    }
    return(
        <div>
            <h2>Welcome</h2>
            <button onClick={()=>clickHandle('Het')}>Click Here</button>
        </div>
    );
}
export default Eventfunctionpass;
```

React Event Object:

- Event handlers have access to the React event that triggered the function.
- To access the event which is fired, **type** is used.
- Ex: If the event is fired with onClick event then we get output as Click. If the event is fired with onMouseOver event then we get output as MouseOver.

Write a program to check which event is fired:
App -> Eventobject

App.js file:

```
import './App.css'
import Eventobject from './components/Eventobject';
const App=()=>{
    return (<Eventobject/>);
}
export default App;
```

Eventobject.js file:

```
const Eventobject=()=>{
    const clickHandle=(b)=>{
        alert(b.type);
    }
    return(
        <div>
            <h2>Hello</h2>
            <button onMouseOver={(b)=>clickHandle(b)}>Click</button>
        </div>
    );

}
export default Eventobject;
```

Spread (...) Operator:

- Spread operator is allowed to copy all or part of array or object values into another array or object.
- It is also used in destructuring of array or object.
- Example of array:

OperatorSpreadArray.js file:

```
const OperatorSpreadArray=()=>{
    const indoorgame=['Chess','Carrom'];
    const outdoorgame=['Cricket','Football'];
    const game=[...indoorgame,...outdoorgame];
    const [Destruct1,...Destruct2]=game;
    return(
        <>
            <h2>Indoor games: {indoorgame}</h2>
            <h2>Outdoor games: {outdoorgame}</h2>
            <h2>Games: {game}</h2>
            <h2>Destructuring: First element: {Destruct1}</h2>
            <h2>Destructuring: Other elements: {Destruct2}</h2>
        </>
    );
}
export default OperatorSpreadArray;
```

- Example of object:

OperatorSpreadObject.js file:

```
const OperatorSpreadObject=()=>{
    const stud={
        Name:'Patel Shreya',
        Age: 19,
        Gender:'Female'
    }
    const enrolledStud={
        Rno:1,
        Enrollno:123456,
        ...stud,
        Course: 'BCA'
    }
    console.log(enrolledStud)
    return(
        <>
            <h2>Student's detail</h2>
            <h2>Rollno:{enrolledStud.Rno}</h2>
            <h2>Name:{enrolledStud.Name}</h2>
            <h2>Enrollno:{enrolledStud.Enrollno}</h2>
            <h2>Course:{enrolledStud.Course}</h2>
            <h2>Age:{enrolledStud.Age}</h2>
            <h2>Gender:{enrolledStud.Gender}</h2>
        </>
    );
}
export default OperatorSpreadObject;
```

State in React JS:

- State is also known as data container.
- State is an object which is private and fully controlled by component.
- State can be changed.
- Using let, the variable is created but it cannot be changed. But when we use state, then state identifies the change and re-render the change.
- In state object, property values can be stored of component.
- State can be used with Class component as **this.state**.
- State can be used with Function component as **useState Hook** (Advanced React features allowed with React-16.8)

State object is created in Class Component:

- It allows to modify the properties specified in State.

1. Directly inside the class:

- State object can be written without using Constructor.

Use of State object without constructor:
App->Stateclass

App.js file:

```
import './App.css'
import Stateclass from './components/Stateclass';
const App=()=>{
    return (<Stateclass/>);
}
export default App;
```

Stateclass.js file:

```
import React from "react";
class Stateclass extends React.Component{
    state={
        Rno: 1,
        Name: 'Mohit',
        Gender: 'Male'
    }
    render(){
        return(
            <div>
                <h2>Roll No: {this.state.Rno} </h2>
                <h2>Name: {this.state.Name}</h2>
                <h2>Gender: {this.state.Gender}</h2>
```

```
            </div>
        );
    }
}
export default Stateclass;
```

Props can also be passed to the state:

Use of State object without constructor and using Props:
App->Stateclassprop

App.js file:

```
import './App.css'
import Stateclassprop from './components/Stateclassprop';
const App=()=>{
    return (
    <div>
        <div><Stateclassprop rno={2} name={'Raghav'} gender={'Male'} />
</div>
        <div><Stateclassprop rno={3} name={'Meena'} gender={'Female'} />
</div>
    </div>
    );
}
export default App;
```

Stateclassprop.js file:

```
import React from "react";
class Stateclassprop extends React.Component{
    state={
        Rno: this.props.rno,
        Name: this.props.name,
        Gender: this.props.gender
    }
    render(){
        return(
            <div>
                <h2>Roll No: {this.state.Rno} </h2>
                <h2>Name: {this.state.Name}</h2>
                <h2>Gender: {this.state.Gender}</h2>

            </div>
        );
    }
}
```

```
export default Stateclassprop;
```

2. Inside constructor:

- State object can also be written within constructor of class.
- When the class component is created first of all Constructor() is called.
- The class component is already created so **this** is used to set the properties with **State**.
- Under constructor, there is a need to use super() first to call parent class component.
- When the super(props) is called then these props are available throughout Component using **this.props**.

Use of State object with constructor:
App->Stateclassconstruct

App.js file:

```
import './App.css'
import Stateclassconstruct from './components/Stateclassconstruct';
const App=()=>{
    return (
    <div>
        <div><Stateclassconstruct rno={5} /> </div>

    </div>
    );
}
export default App;
```

Stateclassconstruct.js file:

```
import React from "react";
class Stateclassconstruct extends React.Component{
    constructor(props){
        super(props);
        this.state={
            Rno: this.props.rno,
            Name:'Raj'
        }
    }
    render(){
        return(
            <div>
                <h2>Hello: Your rollno is {this.state.Rno} and name is
{this.state.Name}</h2>
```

```
            </div>
        );
    }
}
export default Stateclassconstruct;
```

Updating state in Class Component:

- State properties can be modified directly under the class component.
- It is possible using "setState" method. It can be called through event-handler from where you want to modify the state properties.

Updating Message using class component
App->Stateclassupdate

App.js file:

```
import './App.css'
import Stateclassupdate from './components/Stateclassupdate';
const App=()=>{
    return (
    <div>
        <div><Stateclassupdate/> </div>

    </div>
    );
}
export default App;
```

Stateclassupdate.js file:

```
import React from "react";
class Stateclassupdate extends React.Component{
    constructor(){
        super();
        this.state={
            Message:"Testing - State Class Component Update"
        }
    }
    clickHandle()
    {
        this.setState({Message:'Successfully Message Updated'})
    }
    render(){
        return(
            <>
                <h2>{this.state.Message}</h2>
```

```
                <button onClick={()=>this.clickHandle()} >Click
Here</button>
            </>
        );
    }
}
export default Stateclassupdate;
```

Use of State in Functional Component:

- State can be used with Function component using **Hook feature.** (Hooks are added as feature in 16.8 version of React JS)
- Before this version, the state is used only using Class component. And the properties are modified using this state. But after getting the concept of Hook, now it is possible to modify the properties through Functional components also.
- useState() hook is used for serving this purpose. This useState() function is available in react module.
- Modifying state properties through Functional Component:

Updating Message using Functional component
App->Statefunctionalupdate

App.js file:

```
import './App.css'
import Statefunctionupdate from './components/Statefunctionupdate';
const App=()=>{
    return (
    <div>
        <div><Statefunctionupdate/> </div>

    </div>
    );
}
export default App;
```

Statefunctionupdate.js file:

```
import { useState } from "react";
const Statefunctionupdate= () =>{
    let [message, setValue]=useState('Testing of State modification using
Functional component');
    const clickHandle=()=>{
```

```
        message='Message successfully updated';
        setValue(message);
    }
    return(
        <div>
            <h2>Hello</h2>
            <h2>{message}</h2>
            <button onClick={clickHandle}>Click here</button>
        </div>
        );
}
export default Statefunctionupdate;
```

Display multiplication table of 3 as per user's click.
App->Statefunctionaltable

App.js file:

```
import './App.css'
import Statefunctiontable from './components/Statefunctiontable';
const App=()=>{
    return (
    <div>
        <div><Statefunctiontable/> </div>

    </div>
    );
}
export default App;
```

Statefunctiontable.js file:

```
import { useState } from "react";
const Statefunctiontable = ()=>{
    let [value, setValue]=useState(1);
    const clickHandle=()=>{
        setValue(value+1);
    }
    return (
        <div>
            <h2>Multiplication of Table 3</h2>
            <h2>3 X {value} = {value*3}</h2>
            <button onClick={clickHandle}>Multiply by 3</button>
        </div>
    );
}
```

```
export default Statefunctiontable;
```

Difference between Prop and State:

SN	Props	State
1.	Props are read-only.	State changes can be asynchronous.
2.	Props are immutable.	State is mutable.
3.	Props allow you to pass data from one component to other components as an argument.	State holds information about the components.
4.	Props can be accessed by the child component.	State cannot be accessed by child components.
5.	Props are used to communicate between components.	States can be used for rendering dynamic changes with the component.
6.	Stateless component can have Props.	Stateless components cannot have State.
7.	Props make components reusable.	State cannot make components reusable.
8.	Props are external and controlled by whatever renders the component.	The State is internal and controlled by the React Component itself.

Forms in React JS:

- Forms are an integral part of any modern web application.
- It allows the users to interact with the application as well as gather information from the users.
- A form can contain text fields, buttons, checkbox, radio button, etc.
- React offers a stateful, reactive approach to build a form.
- The component rather than the DOM usually handles the React form.
- Form element internally keep some state. The changes are implemented using setState() method. But in <HTML>, elements like <input>, <button>etc. themselves have state and the values are updated as per user input.
- In React, the form is usually implemented by using controlled components.
- Two types of form inputs:
    - **Controlled Component:**
    - **Uncontrolled Component**

Adding form:

- Same like HTML, form element can be added.
- Its behaviour is same like HTML.
- With react, the default behaviour (Page load) can be prevented and the overall control can be handled by event handler.

Form creation in React:
App->Addform

App.js file:

```
import './App.css'
import Addform from './components/Forms/Addform';
const App=()=>{
    return (
    <div>
        <div><Addform/> </div>

    </div>
    );
}
export default App;
```

Addform.js file:

```
import './Addform.css'
const Addform =()=>{
   return(
      <form className='formmargin'>
         <fieldset>
         <legend >Student's personal information</legend>  <br/>
        <label>Name: (Surname first)</label><br/>
        <input type="text"/><br/>
         <label>Enter your password</label><br/>
         <input type="password"/><br/> <br/>
         <label>Enter your email</label><br/>
         <input type="email"/><br/>
         Enter your Address:<br/>
         <textarea></textarea><br/> <br/>
         <label>Enter your gender</label><br/>
         <input type="radio" value="male"/>Male
         <input type="radio" value="female"/>Female
         <input type="radio" value="others"/>others <br/>

         <label>Select couse: </label>
         <select>
            <option value="BCA">BCA</option>
```

```
            <option value="BBA">BBA</option>
            <option value="B.Com">B.Com</option>
          </select><br/><br/>
          <input type="submit" value="sign-up"/>
        </fieldset>
      </form>
  );
  }
  export default Addform;
```

Addform.css file:

```
.formmargin{
    background-color: rgb(187, 185, 223);
    margin: 10%;
    margin-left: 40%;
    margin-right: 40%;
}
```

Handling & Submitting form:

- Handling forms means handling of data through component. It is applicable when the data is changed or respective button is pressed.
- First of all, inputted data are stored in local state of component.
- The changes can be controlled using event handlers of onChange or onClick.
- Using class component-state or using functions component-useState Hook, these values can be modified after insertion and then it is used for further use.
- Submitting form means we can control the action by adding event handler on onSubmit attribute of form.
- event is used to fetch which event is currently used.
- event.target.value: It is used to see the value of currently applied event.
- event.target.name: It is used to see the name of the currently applied event.

Controlled Component:

In the controlled component, the input form element is handled by the component rather than the DOM. Here, the mutable state is kept in the state property and will be updated only with setState() method.

The form data is controlled by React component.

The inputted data is first of all accessed through states and required modifications can also be applied through JavaScript code before final submission. (Overall control of input data is with controlled component.)

Take single input and display the data:

Take single input from user and display the message:
App-> Singleinputformsubmit

App.js file:

```
import './App.css'
import Singleinputformsubmit from
'./components/Forms/Singleinputformsubmit';
const App=()=>{
    return (
    <div>
        <div><Singleinputformsubmit/> </div>

    </div>
    );
}
export default App;
```

Singleinputformsubmit.js file:

```
import { useState } from 'react';
import './Addform.css'

const Singleinputformsubmit=()=>{
    let [msg,setMsg]=useState("");
    const changeHandler=(event)=>{
        setMsg(event.target.value);
    }
    const submitHandler=(event)=>{
        event.preventDefault();
        alert(msg);
        setMsg('');
    }

    return (
        <form className='formmargin' onSubmit={submitHandler}>
            <label>Enter message:</label><br/>
            <input type='text' value={msg}
onChange={changeHandler}/><br/><br/>
            <input type='submit' value='Submit Message'/>

        </form>
```

```
    );
}
export default Singleinputformsubmit;
```

Take single input from user and display the message in another text box:
App->Singleinputformsubmit

Singleinputformsubmit.js file:

```
import { useState } from 'react';
import './Addform.css'

const Singleinputformsubmit=()=>{
    let [msg,setMsg]=useState("");
    let [dispmsg,setDispmsg]=useState('');
    const changeHandler=(event)=>{
        setMsg(event.target.value);
    }
    const submitHandler=(event)=>{
        event.preventDefault();
        setDispmsg(msg)
        setMsg('');
    }

    return (
        <form className='formmargin' onSubmit={submitHandler}>
            <label>Enter message:</label><br/>
            <input type='text' value={msg} onChange={changeHandler}/>
            <br/><br/>
            <input type='submit' value='Submit Message'/><br/><br/>
            <label >Your message is </label><br/>
            <input type='text'value={dispmsg}/>

        </form>
    );
}
export default Singleinputformsubmit;
```

Take multiple inputs and display the details:

Take multiple inputs from user and display the data:
App->Multiinputformsubmit

App.js file:

```
import './App.css'
import Multiinputformsubmit from './components/Forms/Multiinputformsubmit';
const App=()=>{
    return (
    <div>
        <div><Multiinputformsubmit/> </div>

    </div>
    );
}
export default App;
```

Multiinputformsubmit.js file:

```
import { useState } from 'react';
import './Addform.css'
const Multiinputformsubmit =()=>{
    let [stud, setStud]=useState({
        stname:'',
        stpass:'',
        stemail:'',
        stadd:'',
        stgender:'',
        stcourse:''
    });
    const [studdetail,setstuddetail]=useState({

        stname:'',
        stpass:'',
        stemail:'',
        stadd:'',
        stgender:'',
        stcourse:''
    });
    const changeHandler=(event)=>{
        const name= event.target.name;
        const value=event.target.value;
        setStud({...stud,[name]:value})
    }
    const submitHandler=(event)=>{
        event.preventDefault();
        setstuddetail(stud);
        console.log(studdetail)
    }

   return(
```

```
    <>
      <form className='formmargin' onSubmit={submitHandler}>
         <fieldset>
         <legend >Student's personal information</legend>  <br/>
        <label>Name: (Surname first)</label><br/>
        <input type="text" name='stname' value={stud.stname}
onChange={changeHandler}/><br/>
         <label>Enter your password</label><br/>
         <input type="text" name='stpass' value={stud.stpass}
onChange={changeHandler}/><br/> <br/>
         <label>Enter your email</label><br/>
         <input type="email" name='stemail' value={stud.stemail}
onChange={changeHandler}/><br/>
         Enter your Address:<br/>
         <textarea  name='stadd' value={stud.stadd}
onChange={changeHandler}></textarea><br/> <br/>
         <label>Enter your gender</label><br/>
         <input type="radio" name="stgender" value="male"
onChange={changeHandler}/>Male
         <input type="radio" name="stgender" value="female"
onChange={changeHandler}/>Female
         <input type="radio" name="stgender" value="others"
onChange={changeHandler}/>others <br/>

         <label>Select couse: </label>
         <select name='stcourse' value={stud.stcourse}
onChange={changeHandler}>
            <option value="BCA">BCA</option>
            <option value="BBA">BBA</option>
            <option value="B.Com">B.Com</option>
         </select><br/><br/>
         <input type="submit" value="sign-up"/>

        </fieldset>
      </form>
      <div>
            Name:{studdetail.stname}<br/>
            Password:{studdetail.stpass}<br/>
            Email: {studdetail.stemail}<br/>
            Address:{studdetail.stadd}<br/>
            Gender: {studdetail.stgender}<br/>
            Course: {studdetail.stcourse}<br/>
    </div>
     </>
  );
  }
  export default Multiinputformsubmit;
```

Display multiple inserted students' detail.
Multiinputformsubmit.js file----------

```
import { useState } from 'react';
import './Addform.css'
const Multiinputformsubmit =()=>{
    let [stud, setStud]=useState({
        stname:'',
        stpass:'',
        stemail:'',
        stadd:'',
        stgender:'',
        stcourse:''
    });
    const [studdetail,setstuddetail]=useState([]);
    const changeHandler=(event)=>{
        const name= event.target.name;
        const value=event.target.value;
        setStud({...stud,[name]:value})
    }
    const submitHandler=(event)=>{
        event.preventDefault();
        setstuddetail([...studdetail,stud]);
        console.log(studdetail)
    }

   return(
    <>

      <form className='formmargin' onSubmit={submitHandler}>
         <fieldset>
         <legend >Student's personal information</legend>  <br/>
        <label>Name: (Surname first)</label><br/>
        <input type="text" name='stname' value={stud.stname}
onChange={changeHandler}/><br/>
         <label>Enter your password</label><br/>
         <input type="text" name='stpass' value={stud.stpass}
onChange={changeHandler}/><br/> <br/>
         <label>Enter your email</label><br/>
         <input type="email" name='stemail' value={stud.stemail}
onChange={changeHandler}/><br/>
         Enter your Address:<br/>
         <textarea  name='stadd' value={stud.stadd}
onChange={changeHandler}></textarea><br/> <br/>
         <label>Enter your gender</label><br/>
         <input type="radio" name="stgender" value="male"
onChange={changeHandler}/>Male
```

```
        <input type="radio" name="stgender" value="female"
onChange={changeHandler}/>Female
        <input type="radio" name="stgender" value="others"
onChange={changeHandler}/>others <br/>

        <label>Select couse: </label>
        <select name='stcourse' value={stud.stcourse}
onChange={changeHandler}>
           <option value="BCA">BCA</option>
           <option value="BBA">BBA</option>
           <option value="B.Com">B.Com</option>
        </select><br/><br/>
        <input type="submit" value="sign-up"/>

       </fieldset>
     </form>
     <div>
     {studdetail.map((temp)=>{
           return(
           <div>
               <p>Name:{temp.stname}
               Password:{temp.stpass}
               Email: {temp.stemail}
               Address:{temp.stadd}
               Gender: {temp.stgender}
               Course: {temp.stcourse}<br/></p>
           </div>
           );
       })}
    </div>

   </>
 );
 }
 export default Multiinputformsubmit;
```

Uncontrolled Component:

In the uncontrolled component, the input form element is handled by DOM. In place of writing event handler for state update, ref is used to get form values from DOM.

When you don't want to do modification in user's inputted data, then uncontrolled component is used.

Take single input and display the data:

Display the message using useRef() Hook:

UnConditionalForm.js:----------

```
import { useRef,useState } from "react";

const UnConditionalForm =()=>{
    const msg=useRef(null);
    const [message,setMessage]=useState('');
    const submitHandler=(e)=>{
        e.preventDefault();
        console.log(msg.current);
        setMessage(msg.current.value);
    }
    return(
        <>
            <form onSubmit={submitHandler}>
                <input type='text' ref={msg}/>
                <button>Submit</button>
            </form>
            <div>Inserted message is {message}</div>
        </>
    );
}
export default UnConditionalForm;
```

React Memo

- Using react memo will cause React to skip rendering a component if its props have not changed and it just reuse the last rendered output.
- React memo is used to skip rendering a component if its props are not changed. It just reuses the last rendered output for the child component in place of rendering again.
- It is used to enhance performance.
- The child component is wrapped using React.memo.
- It is used with Functional Component.

App.js file:----------

```
import { useState } from 'react';
import './App.css'
import Memofunc from './components/Memofunc';
```

```
const App=()=>{
    console.warn('Changable Component')
    const [count,setCount]=useState(0);
    const clickHandle=(e)=>{
        e.preventDefault();
        setCount(count+1);
    }
    return (

    <div>
        <div><h3>Counter: {count} </h3></div>
        <div><Memofunc Name={'Het Patel'} /> </div>
        <button onClick={clickHandle}>Click</button>

    </div>
    );
}
export default App;
```

Memofunc.js file:----------

```
import React from "react";
const Memofunc=(props)=>{
    console.warn('Fixed prop component');
    return (<h2>Name: {props.Name}</h2>);
}
export default React.memo(Memofunc);
```

Hooks in React JS:

- Hooks are the new feature introduced in the React 16.8 version.
- It allows to use state and other React features without writing a class.
- Hooks are the functions which "hook into" React state and lifecycle features from function components.
- The idea of using hooks makes it is possible to create full-fledged functional components while using all the React features without need of class component.
- Hooks provides direct API to the react which helps in using react features in better way.
- Hooks are similar to JavaScript functions.
- Hooks can also be called from Custom Hooks.
- Hooks need to import from 'react' before use.

- Hook rules:
  - Hooks can only work at lop level of a component
  - Hooks can only be called inside Functional Component.
  - Hooks cannot be called inside conditional statements, loops and nested functions.

Hook State (useState):

- It is new way of declaring state in Functional Component.
- It uses useState functional component for setting and retrieving state.
- It accepts an argument which is the initial value of the state property and returns the current value of the state property and the method which has capability to modify the state property.
- Syntax:
  const [current_value, method]=useState(initial_value);
- The useState Hook can be used to keep track of strings, numbers, booleans, arrays, objects, and any combination of them.
- Multiple State Hook can also be created within functional component.
- Single State Hook can hold multiple state properties in form of object.
- Single State Hook can also hold array values.

Create multiple states and apply changes to these states:
StateMulti.js file:----------

```
import { useState } from "react";
const StateMulti=()=>{
    const [eno,setEno]=useState(1);
    const [ename, setEname]=useState('Rohit');
    const clickHandler=(e)=>{
        e.preventDefault();
        setEno(2);
        setEname('Manthan');
    }

    return(
        <div>
            <h2>Employee Detail</h2>
            <h2>Employee No: {eno}</h2>
            <h2>Employee Name: {ename}</h2>
            <button onClick={clickHandler}>Change values</button>
        </div>
```

```
    );
}
export default StateMulti;
```

Creating single object of state and apply changes to these values:
StateObject.js file:----------

```
import { useState } from "react";
const StateObject=()=>{
    const [emp, setEmp]=useState({
        eno:1,
        ename:'Rohit'
    });
    return (
        <>
            <h2>Employee Detail</h2>
            <input type='text' value={emp.eno}
onChange={(e)=>setEmp({...emp,eno:e.target.value})}/>
            <input type='text' value={emp.ename}
onChange={(e)=>setEmp({...emp, ename:e.target.value})}/>
            <h2>Employee No: {emp.eno}</h2>
            <h2>Employee Name: {emp.ename}</h2>

        </>
    );
}
export default StateObject;
```

Creating single Array of state and apply changes to these values:
StateArray.js file:----------

```
import { useState } from "react";
const StateArray=()=>{
    const [temp,setTemp]=useState(0);
    const [marks, setMarks]=useState([]);
    const clickHandlerAdd=(e)=>{
        setMarks([...marks,
        {id:marks.length,
        value:temp}])
    }
    return(
        <div>
            <input type='text' value={temp}
```

```
onChange={e=>(setTemp(e.target.value))} />
            <button onClick={clickHandlerAdd}>Add</button>
            <button>Reset</button>
            <div>
                Added values are:
                {marks.map((t)=>{
                    return(
                        <div>
                            {t.id} - {t.value}
                        </div>);
                    })
                }
            </div>
        </div>

    );
}
export default StateArray;
```

Hook Effect (useEffect):

- Hook effects allow you to perform side effects in functional component.
- The function passed to this hook will run when the component is rendered.
- useEffect() Hook function is used to serve this purpose.
- Multiple useEffect() can also be used within single component. It helps in performing different tasks with different state or props.
- It does not use components lifecycle methods which are available in class components. In other words, Effects Hooks are equivalent to componentDidMount(), componentDidUpdate(), and componentWillUnmount() lifecycle methods.
- Side effects have common features which the most web applications need to perform, such as:
    - Updating the DOM
    - Fetching and consuming data from a server API
    - Setting up a subscription, etc.
- **Syntax:**

```
useEffect(()=>{
        effect code
        return ()=>{
                cleanup code
                }
        }, [props/state]);
```

Here,

()=>{}: Code block which you want to perform at the time of component rendering.

return ()=> {}: It is optional. Code block which you want to perform for cleaning event.

[props/state]: It is optional. It is used to implement the Hook effect on specific prop or state of component.

- useEffect() is automatically called:
  - When the component is initialized and rendered
  - When the state of component is changed
  - When the props are changed
  - On the bases of mentioned condition in second argument of useEffect().

- In React component, there are two types of side effects:
  - **Effects Without Cleanup**
    It is used in useEffect which does not block the browser from updating the screen. It makes the app more responsive. The most common example of effects which don't require a cleanup are manual DOM mutations, Network requests, Logging, etc.

  - **Effects With Cleanup**
    Some effects require cleanup after DOM updation. For example, if we want to set up a subscription to some external data source, it is important to clean up memory so that we don't introduce a memory leak. React performs the cleanup of memory when the component unmounts. However, as we know that, effects run for every render method and not just once. Therefore, React also cleans up effects from the previous render before running the effects next time.

When the component is initialized and rendered:
EffectComponentInitialize.js file:----------

```
import { useEffect } from "react";
const EffectComponentInitialize=()=>{
    useEffect(()=>{
        alert('Component initialized and rendered');
    });
    return <h2>Hook Effect</h2>
}
```

```
export default EffectComponentInitialize;
```

- useEffect with multiple states using Second argument:

When all states of the component are changed and when specific state is changed.:
EffectStateChange.js

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

const EffectStateChange=()=>{
    const [num, setNum]=useState(5);
    const [salary,setSalary]=useState(15000);
    useEffect(()=>{
        console.log('Common useEffect() is called')
    });
    useEffect(()=>{
        alert('Number is incremented by 2.');
    },[num])
    useEffect(()=>{
        alert('Salary is added.');
    },[salary])
    return(
        <>
            <h2>Hook Effect with State Change</h2>
            <h2>Number is {num}</h2>
            <h2>Salary is {salary}</h2>
            <button onClick={()=>{setNum(num+2)}}>Add number by two</button>
            <button onClick={()=>{setSalary(salary+1000)}}>Add salary by
thousand</button>
        </>
    );
}
export default EffectStateChange;
```

- useEffect with Props:

Apps.js file----------

```
import './App.css'
import EffectPropsChange from './components/Hooks/EffectPropsChange';
import { useState } from 'react';
const App=()=>{
    const [num, setNum]=useState(5);
    const [salary,setSalary]=useState(15000);
    return (
    <>
```

```
    <EffectPropsChange  num={num} salary={salary}/>
    <button onClick={()=>{setNum(num+2)}}>Add number by two</button>
     <button onClick={()=>{setSalary(salary+1000)}}>Add salary by
thousand</button>
     </>
     );
}
export default App;
```

EffectPropsChange.js file:----------

```
import { useEffect } from "react";

const EffectPropsChange=(props)=>{

    useEffect(()=>{
        console.log('Common useEffect() is called')
    },[props.num, props.salary]);
    useEffect(()=>{
        alert('Number is incremented by 2.');
    },[props.num])
    useEffect(()=>{
        alert('Salary is added.');
    },[props.salary])
    return(
        <>
            <h2>Hook Effect with State Change</h2>
            <h2>Number is {props.num}</h2>
            <h2>Salary is {props.salary}</h2>

        </>
    );
}
export default EffectPropsChange;
```

- Hook cleanup effect: Once the useEffect() is called, its effects remains as it is in memory. To clean up the event effect from memory, there is a need to write return(). Within this return function we can specify the reverse effect of specific event for doing related cleaning activity.

EffectCleanUpEvent.js file:----------

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

const EffectCleanUpEvent=()=>{
    const [num, setNum]=useState(5);
    const clickFunction=()=>{
        console.log('Click Event added');
```

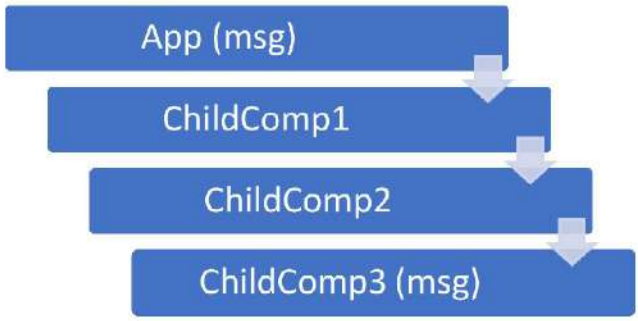
```
    }

    useEffect(()=>{
        window.addEventListener("click",clickFunction);
        return ()=>{
            window.removeEventListener("click",clickFunction);
        }

    });

    return(
        <>
            <h2>Hook Effect with State Change</h2>
            <h2>Number is {num}</h2>
            <button onClick={()=>{setNum(num+2)}}>Add number by two</button>
        </>
    );
}
export default EffectCleanUpEvent;
```

Hook Context (useContext):



- When we want to pass the props from App component to ChildComp3 then there is a need to pass props from
  App->ChildComp1->ChildComp2->ChildComp3
  Means unnecessary, the props are passed through Comp1 and 2.
- Above problem can be solved using Contex.
- Hook context provides a way to pass data through the component tree without passing props down manually through each parent levels.
- Problem can be solved Using Context Provider and useContext Hook.

Context Provider:

Three steps:

- Create Context in App component using React.createContext().
- Provide values through this context which wraps its child component.
- Consume Context value from Child component. (Export Context from App and import it in Child component where you want to use)

Passing props from parent component to child component.
App->ContextComponent1->ContextComponent2

App.js file:----------

```
import React from 'react';
import './App.css'
import ContextComponent1 from './components/Hooks/ContextComponent1';

export const msgContext =React.createContext();
const App=()=>{
    return (
    <>
    <msgContext.Provider value={'Passing message from Parent Component'}>
        <ContextComponent1/>
```

```
    </msgContext.Provider>
     </>
     );
}
export default App;
```

ContextComponent1.js file:----------

```
import ContextComponent2 from "./ContextComponent2";
const ContextComponent1=()=>{
    return(
        <>
            <ContextComponent2/>
        </>
    );
}
export default ContextComponent1;
```

ContextComponent2.js file:----------

```
import { msgContext } from "../../App";
const ContextComponent2=()=>{
    return(
        <>
            <h2>Within Component 2 - Use of ContextProvider</h2>
            <msgContext.Consumer>
                {
                    msg=>{return <h2>Message: {msg} </h2>}
                }
            </msgContext.Consumer>
        </>
    );
}
export default ContextComponent2;
```

useContext() Hook:

- Using useContext(), the consumer side (Child Component) code becomes simpler.
- Syntax:
  const var_name=useContext(context_provider_name);

ContextComponent3.js file:----------

```
import { useContext } from "react";
```

```
import { msgContext } from "../../App";
const ContextComponent3=()=>{
    const msg=useContext(msgContext);
    return(
        <>
            <h2>Within Component 3 - Use of UseContext Hook</h2>
            <h2>Message: {msg} </h2>
        </>
    );
}
export default ContextComponent3;
```

- Multiple contexts can also be passed using this concept.

App->ContextComponent1->ContextComponentMultiple

App.js file:----------

```
import React from 'react';
import './App.css'
import ContextComponent1 from './components/Hooks/ContextComponent1';

export const rnoContext =React.createContext();
export const nameContext=React.createContext();
const App=()=>{
    return (
    <>
    <rnoContext.Provider value={101}>
        <nameContext.Provider value={'Raghav'}>
            <ContextComponent1/>
        </nameContext.Provider>
    </rnoContext.Provider>
     </>
     );
}
export default App;
```

ContextComponentMultiple.js file:----------

```
import { useContext } from "react";
import { rnoContext, nameContext } from "../../App";

const ContextComponentMultiple=()=>{
    const Rno=useContext(rnoContext);
    const Name=useContext(nameContext);
    return (
        <>
            <h2>Multiple values passed through App Component</h2>
            <h2>The rollno of student is {Rno}</h2>
```

```
            <h2>The name of student is {Name}</h2>

        </>
    );
}
export default ContextComponentMultiple;
```

Hook Reference (useRef):

- The useRef Hook allows you to persist values between renders.
- It can be used to store a mutable value that does not cause a re-render when updated.
- Unconditional form can be applied using useRef() Hook.
- It can be used to access a DOM element directly. It permits us to specifically manipulate DOM elements directly.

RefHook.js file: ----------

```
import { useRef } from "react";

const RefHook =()=>{
    let refeffect=useRef(null);
    const clickHandler=()=>{
        refeffect.current.value='React JS';     //set value
        //refeffect.current.style.backgroundColor='yellow'; //set background
        //refeffect.current.disabled=true;  //Make textbox disable
        //refeffect.current.focus();      //keep focus on text.
        //refeffect.current.select();   //keep text selected
        //refeffect.current.blur();   //remove focus
    }
    return(
        <>
            <input type='text' ref={refeffect}/>
            <button onClick={clickHandler}>useRef Hook</button>
        </>
    );
}
export default RefHook;
```

- If we tried to count how many times our application renders using the useState Hook, we would be caught in an infinite loop since this Hook itself causes a re-render.
- To avoid this, we can use the useRef Hook.

To count numbers of time event occurs:--------- (W3School)

RefCounter.js:----------

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


function RefCounter() {
  const [inputValue, setInputValue] = useState("");
  const count = useRef(0);

  useEffect(() => {
    count.current = count.current + 1;
  });

  return (
    <>
      <input
        type="text"
        value={inputValue}
        onChange={(e) => setInputValue(e.target.value)}
      />
      <h1>Render Count: {count.current}</h1>
    </>
  );
}
export default RefCounter;
```

Hook Reducer (useReducer):

- It is used for complex state management.
- It is an alternative of useState.
- It is also helps in managing Global State in Redux and React Context API.
- useState is built using useReducer.
- If you find yourself keeping track of multiple pieces of state that rely on complex logic, useReducer makes it easy for us.
- Syntax:
  const [state,dispatch]=useReducer(reducer, initialstate);
  Here,
  - State: Current state
  - Dispatch: It is a function which specifies "Which action needs to perform?". It is used to trigger ACTION method of REDUCER. It has TYPE property to perform this activity.
  - Initialstate: To set initial state value

- Reducer: It is a function which takes input (state and action) and returns new state after performing action on state. It is most important function for useReducer Hook.

  Syntax: const reducer=(state,action)={

  .....

  return state;

  }

Apply increment/decrement/reset using useReducer Hook:
ReducerIncDecRes.js file:----------

```
import { useReducer } from "react";
const initialvalue=0;
const reducer=(state, action)=>{

    if(action.type==='inc')
        return state+1;
    if(action.type==='dec')
        return state-1;
    if(action.type==='res')
        return state=initialvalue;
}
const ReducerIncDecRes=()=>{
    const [state, dispatch]=useReducer(reducer,initialvalue);
    return (
        <div>
            <h2>Use of useReducer Hook</h2>
            <h2>{state}</h2>
            <button onClick={()=>dispatch({type:'inc'})}>Increment</button>
            <button onClick={()=>dispatch({type:'dec'})}>Decrement</button>
            <button onClick={()=>dispatch({type:'res'})}>Reset</button>
        </div>
    );
}
export default ReducerIncDecRes;
```

Perform +,-,*,/ and square root operations using useReducer() hook.
ReducerCalculator.js file:----------

```
import { useReducer,useState } from "react";
const initialvalue=0;

const ReducerCalculator=()=>{
    const [num1,setNum1]=useState(0);
    const [num2,setNum2]=useState(0);
```

```
    const reducer=(state, action)=>
    {
        switch(action.type)
        {
            case 'add':return Number(num1)+Number(num2);
            case 'sub':return Number(num1)-Number(num2);
            case 'mul':return Number(num1)*Number(num2);
            case 'div':return Number(num1)/Number(num2);
            case 'sqrt': return Number(num1)*Number(num1);
            default: return 0;
        }
    }
    const [state,dispatch]=useReducer(reducer,initialvalue);

    return(
        <>
            <h2>Calculator</h2>
            First no:<input type='text' name={num1}
onChange={(e)=>setNum1(e.target.value)} /><br/>
            Second no:<input type='text' name={num2}
onChange={(e)=>setNum2(e.target.value)}/><br/>
            <h2>Result: {state}</h2><br/>
            <button onClick={()=>dispatch({type:'add'})}>Add</button>
            <button onClick={()=>dispatch({type:'sub'})}>Subtract</button>
            <button onClick={()=>dispatch({type:'mul'})}>Multiply</button>
            <button onClick={()=>dispatch({type:'div'})}>Divide</button>
            <button onClick={()=>dispatch({type:'sqrt'})}>Square of first
no</button>
        </>
    );
}
export default ReducerCalculator;
```

Hook Callback (useCallback):

- It is used for performance optimization.
- By default, all the components are rendered when any changes occurred. For example, if 50 components are called in parent component, then all these 50 components are called during rendering process. This activity decreases the performance of the system. (Solution is React.Memo)
- With react.memo, only those components are rendered where some changes are applied. When any function is passed as Props to the child component, then because of 'Reference Equality", during re-rendering all functions of the

components are again created. Now this function is considered as Modified by React and so re-rendering of the same is done.

- In this type of situation, Hook Callback is used.
- Hook callback allows changes only if one of the dependencies are changed.
- It is helpful to optimize the re-rendering of child components.
- It prevents child component to re-render till its props are not modified.
- Hook callback is possible using useCallback().
- useCallback returns the memorized function.
- useCallback() runs only when one of its dependencies are updated.
- Syntax:

```
Function_name=useCallback(()=>{function_block},[dependency]);
```

Without using useCallback:----------

CallbackWithout.js file:----------

```
import CallbackTesting from "./CallbackTesting";
import { useState } from "react";

const CallbackWithout=()=>{
    console.log('Inside Without Callback Component');
    const [count,setCount]=useState(0);
    const dispMessage=()=>{
        console.log('Inside Function');
    }
    const clickHandler=()=>{
        setCount(count+1);
    }
    return(
        <>
            <h2>Without applying Callback</h2>
            <h2>Counter: {count}</h2>
            <CallbackTesting msg={dispMessage}/>
            <button onClick={clickHandler}>Increment</button>
        </>
    );
}
export default CallbackWithout;
```

CallbackTesting.js file:----------

```
import React from "react";
const CallbackTesting=()=>{
    console.log('Inside Child Component');
    return(
        <>
```

```
            <h2>Inside Callback Child Component</h2>

        </>
    );
}
export default React.memo(CallbackTesting);
```

Using useCallback:----------

CallbackWith.js file:----------

```
import CallbackTesting from "./CallbackTesting";
import { useState,useCallback } from "react";

const CallbackWithout=()=>{
    console.log('Inside Without Callback Component');
    const [count,setCount]=useState(0);
    const dispMessage=useCallback(()=>{
        console.log('Inside Function');
    },[]);
    const clickHandler=()=>{
        setCount(count+1);
    }
    return(
        <>
            <h2>Without applying Callback</h2>
            <h2>Counter: {count}</h2>
            <CallbackTesting msg={dispMessage}/>
            <button onClick={clickHandler}>Increment</button>
        </>
    );
}
export default CallbackWithout;
```

Hook Memo (useMemo):

- The React useMemo Hook returns a memorized value.
- The useMemo Hook only runs when one of its dependencies update.
- This can improve performance.
- The useMemo and useCallback Hooks are similar. The main difference is that useMemo returns a memorized value and useCallback returns a memorized function.
- Syntax:

const var_name=useMemo(function,[dependency])

Without using useMemo function

MemoTesting.js file:----------

```
import { useState } from "react";
const MemoTesting=()=>{
    const [count,setCount]=useState(0);
    const [countmultiple, setCountmultiple]=useState(1);
    const testing=()=>{
        console.log('Within testing function');
        return count+5;
    }
    return(
        <>
            <h2>Count: {count}</h2>
            <h2>Countmultiple: {countmultiple}</h2>
            <h2>changing count value through testing function:
{testing()}</h2>
            <button onClick={()=>setCount(count+1)}>Add Count</button>
            <button onClick={()=>setCountmultiple(countmultiple*5)}>Add
Countmultiple</button>
        </>
    );
}
export default MemoTesting;
```

Here, when you click on any button, respectively the TESTING() function is called which is not required at the time of COUNTMULTIPLE state. It decreases the performance of the system.

- To solve the above problem, useMemo is used.

MemoTestingUseMemo.js file:----------

```
import { useState, useMemo } from "react";
const MemoTestingUseMemo=()=>{
    const [count,setCount]=useState(0);
    const [countmultiple, setCountmultiple]=useState(1);
    const test_var=useMemo(function testing(){
        console.log('Within testing function');
        return count+5;
    },[count]);
    return(
        <>
            <h2>Count: {count}</h2>
            <h2>Countmultiple: {countmultiple}</h2>
```

```
            <h2>changing count value through testing function: {test_var}</h2>
            <button onClick={()=>setCount(count+1)}>Add Count</button>
            <button onClick={()=>setCountmultiple(countmultiple*5)}>Add
Countmultiple</button>
        </>
    );
}
export default MemoTestingUseMemo;
```

Custom Hook: (Building, advantages and use)

- When you have component logic that needs to be used by multiple components, we can extract that logic to a custom Hook.
- We can create new file (Name start with "use...") which contains the function where respective logic is placed. The same file can be imported to another component and can be used to apply same logic.
- Ex:

useTitle.js file:----------

```
import { useEffect, useState } from "react";
const useTitle=(props)=>{
    useEffect(()=>{document.title='Custom Hook:'+props});
}
export default useTitle;
```

CustomHook.js file:----------

```
import { useState } from "react";
import useTitle from "./useTitle";
const CustomHook=()=>{
    const [count,setCount]=useState(0);
    useTitle(count);
    return(
        <>
            <h2>Use of Custom Hook useTitle</h2>
            <button onClick={()=>{setCount(count+1)}}>Add</button>
        </>
    );
}
export default CustomHook;
```

Advantages:

- Improve performance
- Helps in performing life cycle equivalent operations through functional component.
- Helps in avoiding unnecessary processing of components which are not changed during re-rendering process.
- Helps in cleaning unnecessary load of memory.
- Reduce unnecessary load of props.