* **PROPS :**

**1.** function App(props){

return(

<div>

<h1>Hi my name is {props.name}</h1>

<h2>Age: {props.age}</h2>

</div>

)}

ReactDOM.render(<App name="Prateek" age={24} />, document.getElementById("root"));

**2.** function App({name,age}){

return(

<div>

<h1>Hi my name is {name}</h1>

<h2>Age: {age}</h2>

</div>

)}

ReactDOM.render(<App name="Prateek" age={24} />, document.getElementById("root"));

* **PASSING A LIST/ARRAY** :

const days=["monday","tuesday","wednesday","thursday","friday"];

function AllDays(props){

return(

<div>

<ul>

{days.map(day=>(<li>{day}</li>))}

</ul>

</div>

)}

ReactDOM.render(<AllDays />, document.getElementById("root"));

* **ADDING KEYS IN MAPPING ARRAY :**

**1.** const days=["monday","tuesday","wednesday","thursday","friday"];

function AllDays(props){

return(

<div>

<ul>

{props.days.map((day, i)=>(<li key={i}>{day}</li>))}

</ul>

</div>

)}

ReactDOM.render(<AllDays days={day}/>, document.getElementById("root"));

**2.** const days=["monday","tuesday","wednesday","thursday","friday"];

const daysObject=days.map((day, i)=>({

id:i,

item:day

})

);

function AllDays(props){

return(

<div>

<ul>

{props.days.map((day)=>(<li key={day.id}>{day.item}</li>))}

</ul>

</div>

)}

ReactDOM.render(<AllDays days={daysObject}/>, document.getElementById("root"));

**NOTE :**

**De-structuring:** reaching for an object and grabbing it by its key is known as de-structuring.

* **useState :**

import {useState} from "react";

function App(){

const [emotion,setEmotion]=useState("happy");

return(

<div>

<h1>Current Emotion is {emotion}</h1>

<button onClick={()=>setEmotion("sad")}>sad</button>

</div>

)

}

* **useEffect :**

It is used to manage side-effects that are not related to components rendering,

things like logging in console, loading data or animations are often benefitted by the useEffect.

1. import {useEffect} from "react";

useEffect(

()=>{ console.log(`It's ${emotion} right now`);}

,[ ] );

*// passing blank array as second arg.(dependency array) means it will only be called on first render*

*// we can also pass a value to an array as when the effect will be called e.g ,[emotion] .*

function App(){

const [emotion,setEmotion]=useEffect("happy")

return(

<div>

<h1>Current Emotion is {emotion}</h1>

<button onClick={()=>setEmotion("sad")}>sad</button>

</div>

)}

**2.** import { useState } from 'react';

function AppMy(){

const [checked,setChecked]=useState(false);

return(

<div>

<input type="checkbox" value={checked} onChange={()=>setChecked(!checked)} />

<label>{checked?"checked":"not checked"}</label>

</div> )}

* useReducer:

It takes two arguments -

i) function that we use to update our state

ii) initial value of state

import {useReducer} from "react";

function App(){

const [checked,setChecked]=useReducer({(checked)=>!checked},false);

return(

<div>

<input type="checkbox" value={checked} onChange={setChecked} />

<label>{checked?"Checked":"Not Checked"}</label>

</div>

)}

* **useRef :** A ref is the way to reach out to individual element and check what its value is.

useRef will not re-render the change.

-------(*uncontrolled form element (useRef*))------------

import { useRef } from 'react';

function AppMy(){

const nameofColor=useRef();

const colorHex=useRef();

const submit=(e)=>{

e.preventDefault();

const name=nameofColor.current.value;

const color=colorHex.current.value;

alert( `${name}, ${color}` );

nameofColor.current.value="";

colorHex.current.value="";

}

return(

<div>

<form onSubmit={submit}>

<input type="text" ref={nameofColor} placeholder='Enter color name'/>

<input type="color" ref={colorHex}/>

<button>Add</button>

</form>

</div>

)

}

--------------(*controlled form element (useState)*)------------------

import { useState } from 'react';

function AppMy(){

const [name,setName]=useState("");

const [color,setColor]=useState("");

const submit=(e)=>{

e.preventDefault();

alert( `${name}, ${color}` );

setName("");

setColor("#fffff");

}

return(

<div>

<form onSubmit={submit}>

<input type="text" placeholder='Enter color name' value={name} onChange={(e)=>setName(e.target.value)}/>

<input type="color" value={color} onChange={(e)=>setColor(e.target.value)}/>

<button>Add</button>

</form>

</div>

)

}

* **Building custom hooks:** always start with the keyword "use"

function userInput(initialValue){

const [value,setValue]=useState(initialValue);

return [

{ value,onChange:(e)=>setValue(e.target.value)},

()=>setValue(initialValue)

];

}

function App(){

const [state,setState]=userInput("");

}

* **Fetching data from hooks:**

useState : to handle the data

useEffect : to make api call

import {useState, useEffect} from "react";

function GitHubUser({name,location}){

return(

<div>

<h1>{name}</h1>

<p>{location}</p>

</div>

);

}

function App(){

const [data,setData]=useState(null);

useEffect(

()=>{

fetch('https://api.github.com/users/prateek')

.then((response)=>response.json())

.then(data=>setData(data))

},[]

)

if(data)

return(

<GitHubUser name={data.name} location={data.location} />

)

}

* **Handling loading states:**

When we fetch data from an external API, our data can be in one of a few different states.

1. Loading State : fetching the data but it hasn't come back.

2. Success State : data to display

3. Error State : if something goes wrong

import {useEffect, useState} from "react";

function GitHubUser({name, location}){

return(

<h1>{name}</h1>

<p>{location}</p>

)

}

function App(){

const [data,setData]=useState(null);

const [error,setError]=useState(null);

const [loading,setLoading]=useState(false);

useEffect(

()=>{

setLoading(true);

fetch('https://api.github.com/users/prateek')

.then((response)=>response.json())

.then(data=>setData(data))

.then(()=>setLoading(false))

.catch(setError);

},[ ]

);

if (loading) return <h1>Loading ...</h1>

if (error)

return <pre>{JSON.stringify(error)}</pre>;

if (!data) return null;

return(

<GitHubUser name={data.name} location={data.location} />

)

}

* **Fetching data with GraphQL:**

import {useEffect, useState} from "react";

const query=`

query{

allLifts{

name

elevationGain

status

} }`;

const opts={

method:"POST",

headers:{"content-type":"application/json"},

body:JSON.stringify({query})

}

function Lift({name,elevationGain, status}){

return(

<div>

<h1>{name}</h1>

<p>{elevationGain} {status}</p>

</div>

)

}

function App(){

const [data, setData]=useState(null);

const [error, setError]=useState(null);

const [loading, setLoading]=useState(false);

useEffect(

()=>{

setLoading(true);

fetch('https://snowtooth.moonhighway.com/', opts)

.then((response)=>response.json())

.then(setData)

.then(()=>setLoading(false))

.catch(setError);

},[ ]

);

if (loading) return <h1>Loading ...</h1>

if (error)

return <pre>{JSON.stringify(error)}</pre>;

if (!data) return null;

return(

<div>

{data.data.allLifts.map((lift)=>{

<lift name={lift.name} elevationGain={lift.elevationGain} status={lift.status}/>)

}}

</div>

);

}

* **Working with render props:**

const tahoe\_peaks=[

{name:"Freel", elevation:10891},

{name:"Monument", elevation:10067},

{name:"Pyramid", elevation:9983}

]

function List({data,renderItem,renderEmpty}){

return !data.length

? renderEmpty

: (<ul>

{data.map((item)=>(<li key={item.name}>{renderItem}</li> ))}

</ul>)

}

function App(){

return (

<List data={tahoe\_peaks}

renderEmpty={<p>This List is Empty</p>}

renderItem={(item)=>(

<>

{item.name} - {item.elevation} ft.

</>

)}

/>

)}

* **Installing react router v6**

npm install react-router-dom@6

* *In App.js:*

function Home(){

return(

<h1>Home</h1>

)

}

export function About(){

return(

<h1>About</h1>

)

}

export function Contact(){

return(

<h1>Contact</h1>

)

}

function App(){

return(

<Home/>

)

}

* *In Index.js:*

import {Contact,About} from "./App";

import {BrowserRouter, Routes, Route} from "react-router-dom";

ReactDOM.render(

<BrowserRouter>

<Routes>

<Route path="/about" element={<About/>}/>

<Route path="/contact" element={<contact/>} />

</Routes>

</BrowserRouter>,

document.getElementById("root")

)

* **Incorporating the link component:**
* *In App.js*

import {Link} from "react-router-dom";

function Home(){

return(

<div>

<nav>

<Link to="/about">About</Link>

<Link to="/contact">Contact</Link>

</nav>

</div>

)

}

* **Nesting links with React Router :**
* *In App.js*

import {Link,Outlet} from "react-router-dom";

export function History(){

return(

<p>This is out history</p>

)

}

export function About(){

return(

<div>

<nav>

<Link to="/contact">Contact</Link>

<Link to="/">Home</Link>

</nav>

<h1>About</h1>

<Outlet/>

</div>

)

}

* *In Index.js:*

import {Contact,About,History} from "./App";

import {BrowserRouter, Routes, Route} from "react-router-dom";

ReactDOM.render(

<BrowserRouter>

<Routes>

<Route path="/about" element={<About/>}>

<Route path="history" element={<History/>}/>

</Route>

<Route path="/contact" element={<contact/>} />

</Routes>

</BrowserRouter>,

document.getElementById("root")

)

* **To test (JEST)**

npm test -> will run the file .test.js

**1.** *Functions.js*

export function timesTwo(a){

return a\*2;

}

*Functions.test.js*

import {timesTwo} from "./Functions";

test(

"Multiplies By Two",

()=>{

expect(timesTwo(4)).toBe(8);

}

);

1. **Using React Testing Library**

1. *Star.js*

export function Star(){

return <h1>Cool Star</h1>

}

*Star.test.js*

import {render} from "@testing-library/react";

import {Star} from "./Star";

test(

"renders an h1",

()=>{

const {getByText} = render(<Star/>);

const h1 = getByText(/Cool Star/);

expect(h1).toHaveTextContent("Cool Star");

}

)

2. *CheckBox.js*

import {useReducer} from "react";

export function Checkbox(){

const [checked, setChecked]=useReducer((checked)=>!checked,false);

return(

<>

<label htmlFor="checked">

{checked ? "checked" : "not checked"}

</label>

<input id="checked" type="checkbox" value={checked} onChange={setChecked}/>

</>

)

}

*CheckBox.test.js*

import {render,fireEvent} from "@testing-library/react";

import CheckBox from "./CheckBox";

test(

"Selecting checkbox to change value to true",

()=>{

const {getByLabelText}=render(<Checkbox/>);

const checkbox=getByLabelText(/not checked/i);

fireEvent.click(checkbox);

expect(checkbox.checked).toEqual(true);

}

)