Explain about React State :- React State object, How to use the state object

props vs state

props get passed to the component

Function parameters

Props are immutable

props- Functional component

this.props – Class component

state is managed within the component

Variables declared in the function body

State can be changed

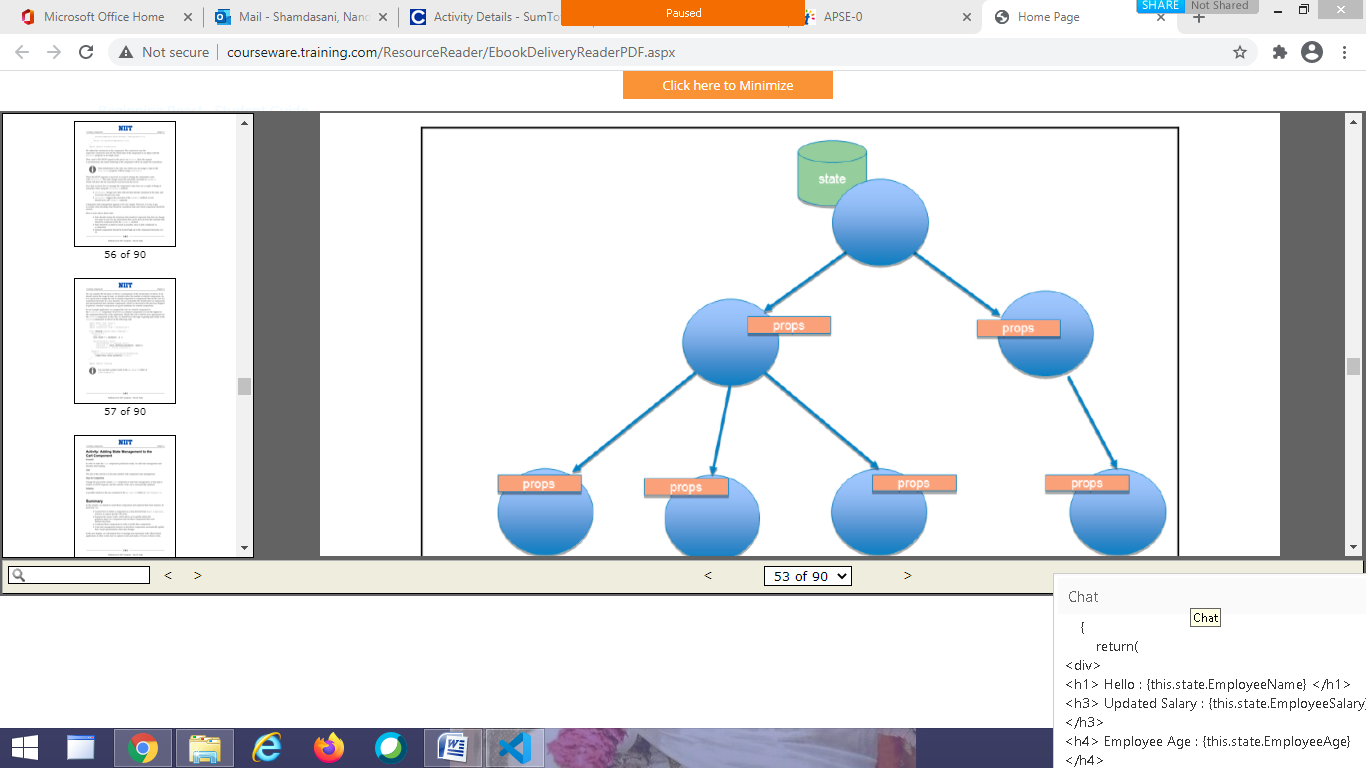
useState Hook – Functional Component

this.state – Class component

Demonstrate on React State and Props

Every react component has a props property. It is to collect data input passed to the component iself.

It is the React property that represents the data that changes over time.



In a component hierarchy, data propagation is very important. We can pass data from one component to another as a unidirectional data flow, from the parent component toward the child components.

A change in the state causes a data propagation toward the child components through the props property.

import React,{Component} from 'react';

class Employee extends Component

{

constructor(props)

{

super(props);

this.state={

EmployeeName:"Saurabh",

EmployeeSalary:25000,

EmployeeAge:20

};

}

changeSalary=()=>{

this.setState({EmployeeSalary:35000})

}

render()

{

return(

<div>

<h1> Hello : {this.state.EmployeeName} </h1>

<h3> Updated Salary : {this.state.EmployeeSalary} </h3>

<h4> Employee Age : {this.state.EmployeeAge}</h4>

<button onClick={this.changeSalary}> Update Salary </button>

</div>

);

}

}

export default Employee;

State initialization is the only case where you can assign a value to the this.state property without using setState().

When the HTTP response is received, it is used to change the component’s state with setState(). This state change causes the automatic execution of render(), which will show the list of products received from the server.

\*\*\* To remember

setState() merges new data with old data already contained in the state, and overwrites the previous state.

setState() triggers the execution of the render() , so you should never call render() explictly

**React JSX**

**JSX stands for JavaScript XML. It is a syntax extension to JavaScript.**

**JSX is a preprocessor step that adds XML syntax to JavaScript.**

**JSX produces a React “elements”. It is possible to create element without JSX but JSX makes a React a lot more elegant.**

**It is recommended to use JSX with React to describe what the UI should look like.**

**JSX is easier to read and write. Babel transform these expressions into a actual JavaScript code.**

**It also allows React to show more useful error and warning messages.**

**Babel :- is a Javascript compiler that has the ability to compile JSX into pure/regular JavaScript.**

**Examples**

**const el=<h1> Hello </h1> //JSX**

**React.createElement(“h1”,null, “hello”); //babel conversion**

**const el=<h1 className=”bg”> Hello </h1>**

**React.createElement(“h1”,{className:”bg”},”Hello”);**

**const el=<h1> hello {name} </h1>**

**React.createElement(“h1”,null, “hello”,name);**

**const el=<Student/>**

**React.createElement(Student,null);**

**const el=<Student name=”Nandini”/>**

**React.createElement(Student,{name:”Nandini”});**

**JavaScript expressions in JSX**

**We can put any valid JavaScript expression inside the curly brackets in JSX.**

**const el=<h1> {10+20} </h1>**

**React JS - Event Handling**

What is Event?

The actions to which JavaScript can respond are called Events.

Event Handling

Handling Events with React elements is very similar to handling events as DOM elements. There are some syntactic differences:

React events are named using camelCase, rather than lowercase.

With JSX you pass a function as the event handler, rather than a string.

import React from ‘react’;

**function testing(){**

**handleClick(){**

**alert (‘Hi this is HTML event’);**

**}**

**}**

**In HTML**

**<button onclick=”handleClick()” > Click Me </button>**

**In React**

**<button onClick={handleClick}>Click Me </button> //Function component**

**<button onClick={this.handleClick}> Click Me </button> //Class component**

Imrc --- emmet abbreviateion

import React, {Component} from ‘react’;

class Student extends React.Component{

constructor(props){

super(props);

this.state={

name: “Shreehari”,

assctid: this.props.assctid

}

No need of this if using arrow function

this.handleClick=this.handleClick.bind(this);

}

handleClick(){

console.log(“Button Clicked”,this);

}

render(){

return (

<div> <h1> Hello Event {this.state.name} Your Associate Id is {this.state.assctid} </h1>

<button onClick={this.handleClick}> Click Me </button>

</div>

)

}

}

export default Student;

or arrow

handleClick=()=>{

console.log(“Button Clicked”,this);

}

**Index.js**

**Imr**

**Import React from ‘react’;**

**Import ReactDOM from ‘react-dom’;**

**Import Student from ‘./Student’;**

**ReactDOM.render(<Student assctid=”23232”/>,document.getElementById(‘root’));**

**You cannot return false to prevent default behaviour in React. You must call preventDefault explicitly.**

***In HTML***

**<a href=”#” onclick=”console.log(‘Clicked’); return false”>Click me </a>**

***In React***

**function handleClick(e){**

**e.preventDefault();**

**console.log(‘Clicked’);**

**}**

**<a href=”#” onClick={handleClick}>Click Me </a>**

**ES6 arrow functions**

**An arrow function expression has a shorter syntax compared to function expressions. Arrow functions are always anonymous.**

**Syntax:**

**()=>{statements};**

**Example 1**

**function sum(a,b){**

**return a+b**

**}**

**let sum2=(a,b)=>{a+b}**

**Example2**

**function isPositive(number){**

**return number>=0**

**}**

**let positive2=(number)=> number>=0**

**Example3**

**document.addEventListener(‘click’,function(){**

**console.log(‘Click’)**

**})**

**document.addEventListener(‘click’,()=>console.log(‘click’) )**

**React JS - Conditional Rendering**

**In React, we can create distinct components that encapsulates behaviour what we need. We can render only some of the components, depending on the state of the application.**

**Conditional Rendering in React works the same way conditions work in JavaScript.**

**Use JavaScript operators like if or the conditional (ternary) operator to create elements representing the current state, and let React update the UI to match them.**

**Element Variables :- is used to store elements. This helps us in conditionally rendering.**

**If and if-else statements don’t work inside JSX. This is because JSX is just syntatic sugar for function calls and object construction.**

**<div id ={if(condition) {‘msg’}}> Hello </div>**

**React.createElement(“div”, {id:if(condition){‘msg’}},”Hello”);**

**========================= In React**

**if(true){**

**return something;**

**}**

**App.js User.js Guest.js index.html**

**User.js**

**class User extends Component{**

**render(){**

**return(**

**<div>**

**<h1> Welcome Asma </h1>**

**<button> Logout</button>**

**</div>**

**);**

**}**

**}**

**export default User;**

**Guest.js**

**class Guest extends Component{**

**render(){**

**return(**

**<div>**

**<h1> Welcome Guest </h1>**

**<button> Login</button>**

**<button> Signup </button>**

**</div>**

**);**

**}**

**}**

**export default Guest;**

**App.js**

**import User from ‘./User’**

**import Guest from ‘./Guest’**

**class App extends Component{**

**render(){**

**const isRegistered=this.props.consumer;**

**if(isRegistered){**

**return <User/>**

**}**

**else{**

**return <Guest/>**

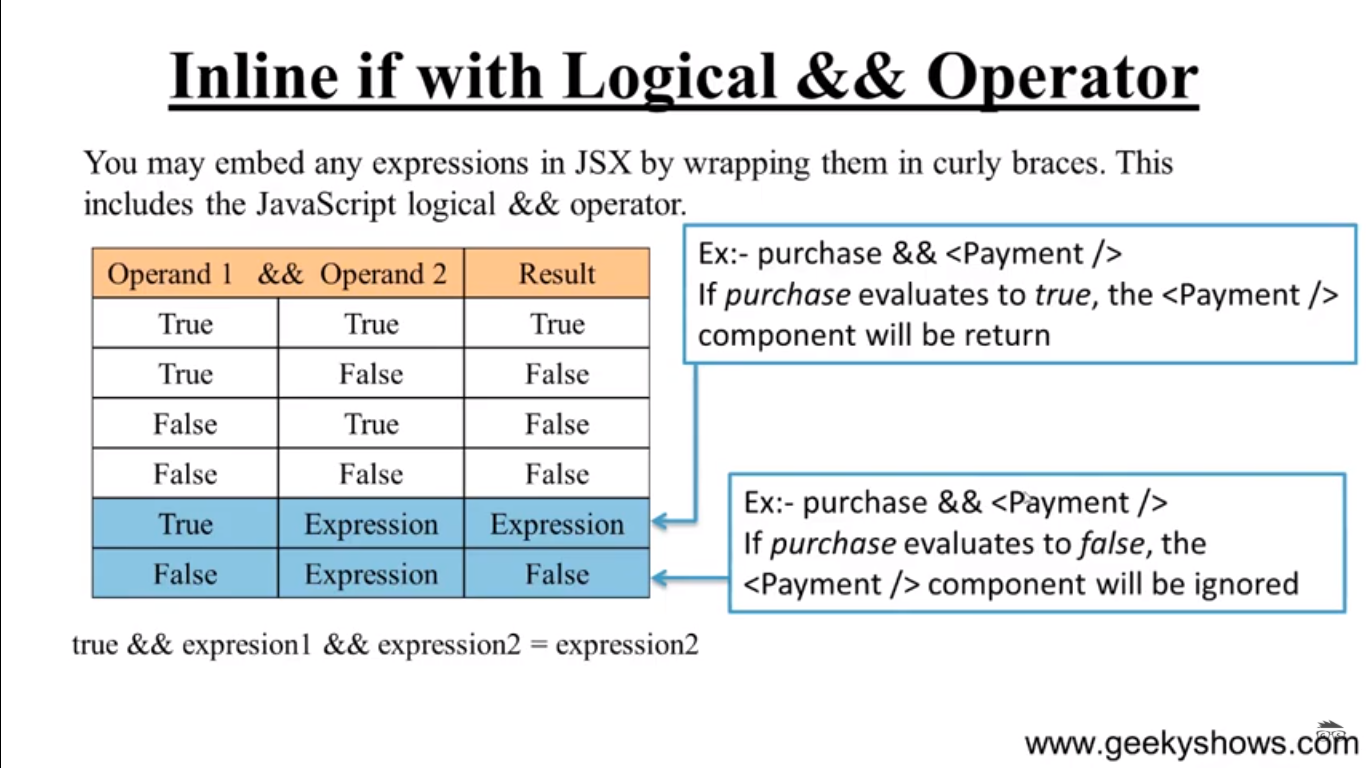
**}**

**}**

**}**

**Index.js**

**ReactDOM.render(<App consumer={true}/>,document.getElementById(“root”));**

**Inline if with Logical && Operator **

App.js User.js index.js index.html

User.js

class User extends Component{

render(){

return <h3> Prime member </h3>

}

}

App.js

import User from ‘./User’

class App extends Component{

render(){

const primeMember=this.props.primeMember

return(

<React.Fragment>

<h1> Welcome User </h1>

{primeMember && <User/>}

</React.Fragment>

}

}

Index.js

Import React from ‘react’;

Import ReactDOM from ‘react-dom’

Import App from ‘./App’;

ReactDOM.render(<App primeMember={true}/>, document.getElementById(“root”));

If else statement

App.js user.js guest.js index.js index.html

User.js

**class User extends Component{**

**render(){**

**return(**

**<React.Fragment>**

**<h1> Welcome Asma | {this.props.name} </h1>**

**<button onClick={this.props.clickData}> Logout</button>**

**</React.Fragment>**

**);**

**}**

**}**

**export default User;**

**Guest.js**

**class Guest extends Component{**

**render(){**

**return(**

**< React.Fragment >**

**<h1> Welcome Guest </h1>**

**<button onClick={this.props.clickData}> Login</button>**

**<button> Signup </button>**

**</ React.Fragment >**

**);**

**}**

**}**

**export default Guest;**

App.js

class App extends Component{

state={

isLoggedIn:false | true

}

clickLogin=()=>{

this.setState({isLoggedIn:true})

}

clickLogout=()=>{

this.setState({isLoggedIn:false})

}

render(){

const isLoggedIn=this.state.isLoggedIn;

let consumer;

if(isLoggedIn){

return | consumer= <User name=”Asma” clickData={this.clickLogout}/>;

} else{

return | consumer=<Guest clickData={this.clickLogin}/>;

}

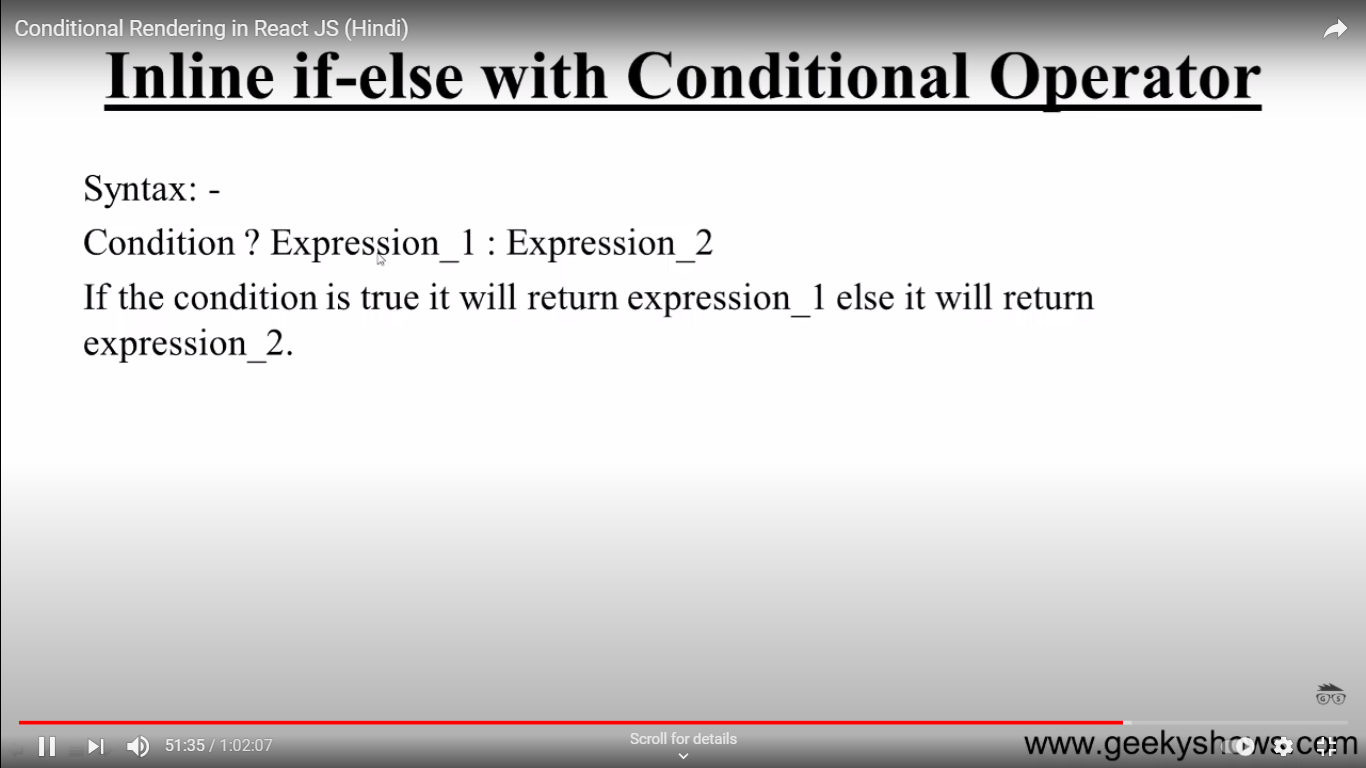
return <div>{consumer} </div>

}

}

Index.js

ReactDOM.render(<App/>,document.getElementById(“root”));



render(){

const isLoggedIn=this.state.isLoggedIn;

return (

<div>

{ isLoggedIn ? (<User clickData={this.clickLogout />) : (<Guest clickData={this.clickLogin}/>)}

</div>

);

}

}