***React JS BasicConcepts:***

***CLASS Components in React:***

***SNIPPET:***

*import { Component } from 'react';*

*class App extends Component {*

*render(){*

*return (*

*<div className="App">*

*Rohit*

*</div>*

*);*

*}*

*}*

*export default App;*

***STATE in Javascript.***

***SNIPPET:***

*class App extends Component {*

*constructor(){*

*super()* ***// Calls the constructor of parent/Component class.***

*this.state={ /****/ state to be used for creating dynamic content.***

*'name':'Rohit',*

*}*

*}*

*render(){*

*return (*

*<div className="App">*

*Hello I am {this.state.name}*

*</div>*

*);*

*}*

*}*

***Changing state variable in react***

***SNIPPET:***

*class App extends Component {*

*constructor(){*

*super()*

*this.state={*

*'name':'Rohit',*

*}*

*}*

*render(){*

*return (*

*<div className="App">*

*Hello I am {this.state.name}*

*<button onClick={this.state.name='Ghost'}>Change</button>*

***// This is not going to work as we reference to different memory location here so***

***react does not re-renders its DOM here.***

*<button onClick={()=>{this.setState({'name':'Ghost'})}}>Change name</button>*

***// This is going to work because here by using setstate we are shallow merging to the***

***states memory by* *checking the same keys that state has and update it eventually.***

*</div>*

*);*

*}*

*}*

***Q. Why console.log() prints the old state even after updating the state??***

*ans.*

***#1SNIPPET:***

*<button onClick={()=>{*

*this.setState({'name':'Ghost'});*

*// console.log(this.state.name);****// Prints Rohit the old value because this setState is a***

***Asynchronous function so and console.log() is***

***synchronous by nature.So, it takes time to perform***

***operations and allows other stuffs that is synchronous***

***to execute in meantime****.*

*}*

*}>Change name</button>*

***#2SNIPPET****:* ***If we want to see the console.log correctly with change in state.***

*<button onClick={()=>{*

*this.setState(*

*()=>{*

*return{'name':'Ghost'}*

*},*

*()=>{*

*console.log(this.state.name);*

*}*

*);*

*}}>Change name</button>*

***MAPS in React:***

***Maps in Javascript iterate over each element of an array.***

***SNIPPET:***

*class App extends Component {*

*constructor(){*

*super()*

*this.state={*

*'monsters':[*

*{*

*name:'Ron',*

*id:'1'*

*},*

*{*

*name:'vlood',*

*id:'2'*

*},*

*{*

*name:'rainy',*

*id:'3'*

*}*

*]*

*}*

*}*

*render(){*

*return (*

*<div className="App">*

*{*

*this.state.monsters.map((element)=>{*

*return(*

*<div key={element.id}>*

*<h1>{element.name}</h1>*

*</div>*

*)*

*})*

*}*

*</div>*

*);*

*}*

*}*

***Key atttribute: It is needed to be added in top level html attribute as it helps the react to***

***identify,update and renrender that particular state when changes are***

***applied to it. Basically ids are only kept as keyy attributes.***

***LifeCycle methods of React: ComponentDIdMount().***

1. *When react renders a component on to a page that is called as Mounting.Happens only once in a lifecycle.*

*2. Used to make API requests.*

1. *SNIPPET:*

*componentDidMount(){*

*fetch('https://jsonplaceholder.typicode.com/users')*

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

*.then((users)=>{*

*this.setState({'monsters':users});*

*})*

*}*

***Flow of control in react.***

*1. Constructor runs and sets the default state.*

*2. The the render() runs and render the component on to the page.*

*3. ComponentDidMount() runs after the component mounts/renders for first time and if it has got*

*any Api request that changes the the states it goes along and completes its task.*

*4. As soon as the state changes the component rerenders itself.*

***Filter() in react:***

*Used to filter the array of elements and create a new array.*

***SNIPPET:***

*render() {*

*const filtermonster = this.state.monsters.filter((monsterelement)=>{*

***// Filter Takes an array and filters out the elements based on the return condition to***

***create a new array.***

*return monsterelement.name.toLowerCase().includes(this.state.search\_monsters);*

***// includes function here checks whether the given string/character is a substring of***

***string on which we have applied our include().***

*})*

*return (*

*<div className="App">*

*<input*

*type="text"*

*placeholder="search monsters"*

*onChange={(event) => {*

***//Here OnChange returns the value wriiten inside the input box by***

***event.target.value.***

*this.setState(()=>{*

*return {search\_monsters:event.target.value.toLowerCase()};*

***//toLowerCase() changes all characters of a string to lowercase****.*

*})*

*}}*

***//Note: Whenever we need to update an array we need to update the entire array***

***else the changes won't get rendered on our page.***

*/>*

*{filtermonster.map((element) => {*

*return (*

*<div key={element.id}>*

*<h1>{element.name}</h1>*

*</div>*

*);*

*})}*

*</div>*

*);*

*}*

***Performance Optimisation in React :***

***SNIPPET:***

*class App extends Component {*

*constructor() {*

*super();*

*this.state = {*

*monsters: [],*

*search\_monsters:''*

*};*

*}*

*componentDidMount() {*

*fetch("https://jsonplaceholder.typicode.com/users")*

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

*.then((users) => {*

*this.setState({ monsters: users });*

*});*

*}*

*onSearchChange=(event) => {*

***// SearchChange() we have removed out from the htmp templates and placed it inside the class in order to optimise our code, as with the re-rendering of the components this function was getting initialised again again which reduces the performance of our code but, by putting this inside the class this function only gets intialised once, thereby incresing the performance of our app.***

*this.setState(()=>{*

*return {search\_monsters:event.target.value.toLowerCase()};*

*})*

*}*

*render() {*

***//Destructing of the class state and function is done here in order to increase the readibility of our code.***

***const {monsters,search\_monsters} = this.state;***

***const {onSearchChange} = this;***

*const filtermonster = monsters.filter((monsterelement)=>{*

*return monsterelement.name.toLowerCase().includes(search\_monsters);*

*})*

*return (*

*<div className="App">*

*<input*

*type="text"*

*placeholder="search monsters"*

*onChange={(event)=>{onSearchChange(event)}}*

*/>*

*{filtermonster.map((element) => {*

*return (*

*<div key={element.id}>*

*<h1>{element.name}</h1>*

*</div>*

*);*

*})}*

*</div>*

*);*

*}*

*}*

***Q. How Components Lie inside React Project:***

* ***Inside Src ->*** *create a components folder****.***
* ***Inside components folder ->*** *Make the component folders and keep the code of each component inside their respective folders.*

***Props in React:***

*Props is an object of properties/values passed from parent component to the child*

*component.*

***SNIPPET:***

***Parent Component***

*class App extends Component {*

*constructor() {*

*super();*

*this.state = {*

*monsters: [],*

*search\_monsters:''*

*};*

*}*

*componentDidMount() {*

*fetch("https://jsonplaceholder.typicode.com/users")*

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

*.then((users) => {*

*this.setState({ monsters: users });*

*});*

*}*

*onSearchChange=(event) => {*

*this.setState(()=>{*

*return {search\_monsters:event.target.value.toLowerCase()};*

*})*

*}*

*render() {*

*const {monsters,search\_monsters} = this.state;*

*const {onSearchChange} = this;*

*const filtermonster = monsters.filter((monsterelement)=>{*

*return monsterelement.name.toLowerCase().includes(search\_monsters);*

*})*

*return (*

*<div className="App">*

*<input*

*type="text"*

*placeholder="search monsters"*

*onChange={(event)=>{onSearchChange(event)}}*

*/>*

*<CardList monsters={filtermonster}/>*

***{/\* monster here is passed to the component as a Prop. Props is an object of***

***properties/values passed from parent component to the child***

***component.  \*/}***

*</div>*

*);*

*}*

*}*

*export default App;*

***Child component:***

*class CardList extends Component{*

***// Always keep the Customized components name in Upper Camelcase convention. It becomes easier for react to identify that the very component is made by the developer itself.***

*render(){*

*const {monsters} = this.props* ***//Destructuring of PROPs.***

*return(*

*<div>*

*{*

*monsters.map((element)=>{*

*return(*

*<div key={element.id}>*

*<h1>{element.name}</h1>*

*</div>*

*)*

*})*

*}*

*</div>*

*)*

*}*

*}*

*export default CardList;* ***//Used to export our components.***

***Q. How Components re-renders in react:***

***It is based on 2 conditions:***

* ***Either our state changes***
* ***Or, our props changes.***

***Flow of Components in React :*** *TOP-Down -> From Parent to Child component.*

***CSS in React:***

*Can be used by creating a .css file and writing css by targeting classnames or ID.*

***NOTE: CSS file we import in 1 component of our react is accessible to all the components present inside our Tree of components. This sometimes leads to overlapping of styles because of same classname.***

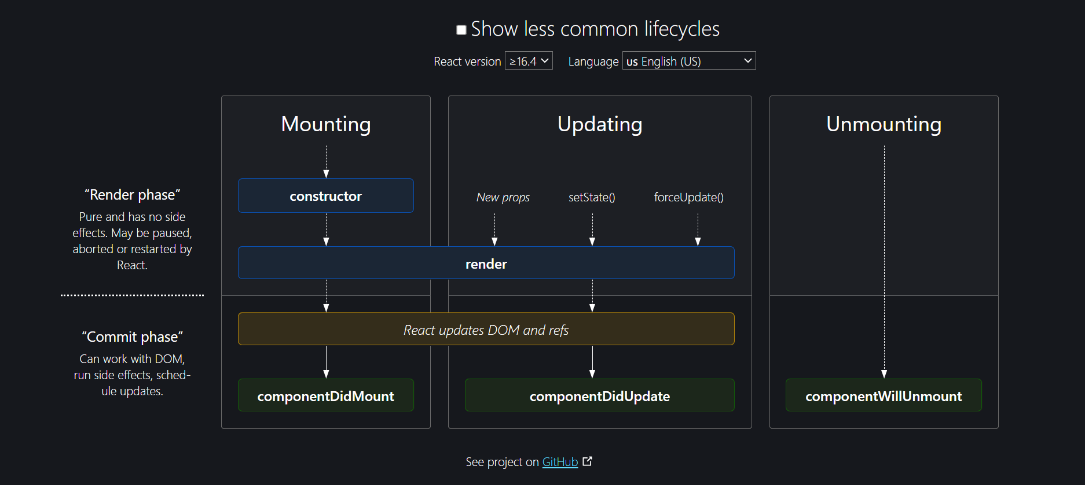
***String Interpolation:***

***Example: `Rohit is a ${this.rohit.personality}`***

*In**String interpolation we can write dynamic variables along with string using*

*backticks.*

***Functional VS Class Based Components:***



***Functional component donot have this lifecycle methods Like componentDidMount(), ComponentDidUpdate() and ComponentWillUnmount() as class components .****But it follows this 3 lifecycles of React given in diagram, i.e.*

***1. Mounting***

***2. Updating***

***3. Unmounting***

***NOTE:***

***ComponentDidUpdate():*** *This method runs when our react component re-renders/updates*

*due to change in state/props/use of forceUpdate() method.*

***ComponentWillUnmount():*** *This method runs at end(i.e. before the component is destroyed or*

*unmounted from the DOM tree.) to perform clean-ups which prevents any*

*type of memory leaks.*

***Functional Components in React:***

* *It does not have any lifecyle like class components.*
* *React display the component as it is from Top to Bottom.*
* ***SNIPPET:***

*const Text = ()=>{*

*return(*

*<h1 className="app-title">Monsters Rolodex</h1>*

*)*

*}*

***Pure and Impure functions in React:***

* ***Pure functions:*** *It return the same output if we use the same input parameters as it only uses the*

*arguments that lies within its scope.*

* ***Impure functions:*** *It give different outcomes when we pass the same arguments multiple times as*

*they use some parameters/variables which lies outside their scope.*

***UseState Hook in Functional components:***

* *It gives us back an array with 2 values. First is the value we want to store. And the second value is the setvalue() function which we gonna use to update our first value.*
* *It is individual values not like objects in class components.*

***Example:*** *import { Component, useState } from "react";*

*const [searchfield, setsearchfield] = useState("");*

*console.log(searchfield);* ***// ’ ’empty string***

*setsearchfield(‘Rohit’)*

*console.log(searchfield);* ***// Rohit***

***Rendering and Re-Rendering in React Functional Componets:***

*Rendering happens only when:*

* ***Either our prop changes***
* ***Or, our state change***

*But as, in case of class components only the render() used to render on change of props or the state, not the entire component. But, in case of functional component entire component re-renders from Top to Bottom..*

***NOTE: React only re-renders when the value in state changes by a different Values***

***(string,number)/values in different memory refrences(array,objects). And Not***

***on just using setState().***

***Making an API call using Fetch:***

**SNIPPET:**

*fetch("https://jsonplaceholder.typicode.com/users")*

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

*.then((users) => {*

*setmonsters(users);*

*});*

***// Here infinite time rerendering of an components occurswhich basically is due the***

***sideeffect(API call), our state gets changed by an array having different memory***

***references. Can be solved using UseEffect.***

***UseEffect HOOK:***

*Used for writing sideeffects(means values outside the scope of our function). Example: APIS.*

***SNIPPET-1:***

*useEffect(() => {*

*fetch("https://jsonplaceholder.typicode.com/users")*

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

*.then((users) => {*

*setmonsters(users);*

*});*

*}, [])* ***// Would be rendered once as dependency array is left empty.***

*Consists of two things inside it one is the*

* ***callback()*** *method that handles the side effect and,*
* ***dependency array*** *which consists of state/props.*

*So, whenever the state/props present inside our dependency array changes then the useEffect calls it callback function to generate the side effects.*

***SNIPPET-2:***

*useEffect(()=>{*

*let newfiltermonster = monsters.filter((monsterelement)=>{*

*return monsterelement.name.toLowerCase().includes(search\_monsters);*

*})*

*setfiltermonster(newfiltermonster);*

*},[monsters,search\_monsters])*

***//States passed here based on which side effects would be generated through our callback() method.***

***Destructing in Function-Based components.***

***SNIPPET:***

*const Card =({element})=>{* ***//Destructing of props to get elements inside.***

*const {id,name,email}=element;* ***//Further Destructing of prop elements.***

*return (*

*<div className="card-container">*

*<img src={`https://robohash.org/${id}?set=set2&size=180x180`} alt="" />*

*<h2>{name}</h2>*

*<p>{email}</p>*

*</div>*

*)*

*}*

***NOTE****: ReactScripts mode in version 18 and above re-renders our component* ***2 times*** *in order to*

*Catch weird behaviors that occurs due to your side-effects in your functional components.*

***SNIPPETS:***

*const root = ReactDOM.createRoot(document.getElementById('root'));*

*root.render(*

*<React.StrictMode>* ***//Renders the components 2 (twice)***

*<App />*

*</React.StrictMode>*

*);*

***How Does Virtual DOM works in React??***

***Real-Dom:*** *The process of Updating/Changing the Real-dom is very costly.**So, React creates a*

*duplicate copy of Real-dom(i.e. the Vitual-dom)**where it reflects elements in javascript*

*which reflects this changes much quicker.*

*Now, we futher create the copy of this Virtual-dom and use this virtual-dom as a snapshot.*

*And the new copy of Virtual-dom what we create will be the place where we would create*

*Changes. Now, this New Virtual dom copy will be compared with the Virtual dom snapshot and*

*highlight changes needs to done. Then we will update our Real-dom eventually.*

***Q. Using React API with actual HTML templates to understand how react internally works??***

***SNIPPET:***

*<!DOCTYPE html>*

*<html lang="en">*

*<head>*

*<meta charset="UTF-8">*

*<meta http-equiv="X-UA-Compatible" content="IE=edge">*

*<meta name="viewport" content="width=device-width, initial-scale=1.0">*

*<title>Document</title>*

*</head>*

*<body>*

*<div id='root'>I am Rohit, I am sad</div>*

*<script src="https://unpkg.com/react@18.0.0-rc.0/umd/react.development.js"></script>*

***// React API***

*<script src="https://unpkg.com/react-dom@18.0.0-rc.0/umd/react-dom.development.js"></script>*

***//ReactDom API***

*<script>*

***React.createElement(1st parameter, 2nd parameter,3rd parameter):***

1. ***1st parameter: element name(div,h1,span,etc)***
2. ***2nd parameter: properties (such as classname)***
3. ***3rd parameter: child elements(written inside an [ ] and if string written inside “ ”.)***

***Here you can add multiple elements separated by commas.***

*const Person = (props) => {*

*return React.createElement('div',{},[*

*React.createElement('h1',{},props.name),*

*React.createElement('p',{},props.role),*

*])*

*}*

*const App = () => {*

*return React.createElement('div',{class:'App'},[*

*React.createElement('h1',{class:'mood'},"Sunny Day"),*

*React.createElement(Person,{name:'Rohit',role:'DeveOPs'},null),*

*React.createElement(Person,{name:'Rohan',role:'Web3'},null),*

*React.createElement(Person,{name:'Roshan',role:'Developer'},null),*

*])*

*}*

*ReactDOM.render(*

*//* ***ReactDom.render() 2 arguments*** *:*

***1. The element we want to Render.***

***2. The Target element***

*React.createElement(App),*

*document.getElementById('root')*

*)*

*</script>*

*</body>*

*</html>*

***NOTE:***

***1.Whenever we update/make changes to our dom, or when our components get mounted***

***and unmouted then we see reflows/flashes in our page, this is a way we can see when***

***react renders and re-renders a components within a tree of components.***

***2.SSH keys:***

*you can* ***connect to GitHub without supplying your username and personal access token****at*

*each visit. You can also use an SSH key to sign commits. You can access and write data in*

*repositories on GitHub.com using SSH* ***(Secure Shell Protocol).***

***CapStone Project***

***Sass Vs. Css.***

* ***The difference is how we target the child elements in sass and css.***
* ***When we build our react App this Sass is again gets converted into css file, which is understood by our search engines.***
* ***Example: Sass***

*.category-body-container {*

*height: 90px;*

*padding: 0 25px;*

*display: flex;*

*flex-direction: column;*

*align-items: center;*

*justify-content: center;*

*border: 1px solid black;*

*background-color: white;*

*opacity: 0.7;*

*position: absolute;*

*h2 {*

*font-weight: bold;*

*margin: 0 6px 0;*

*font-size: 22px;*

*color: #4a4a4a;*

*}*

*}*

***Example: css***

*.category-body-container {*

*height: 90px;*

*padding: 0 25px;*

*display: flex;*

*flex-direction: column;*

*align-items: center;*

*justify-content: center;*

*border: 1px solid black;*

*background-color: white;*

*opacity: 0.7;*

*position: absolute;*

*}*

*.category-body-container  h2 {*

*font-weight: bold;*

*margin: 0 6px 0;*

*font-size: 22px;*

*color: #4a4a4a;*

*}*

***Note:***

* ***We select the Google fonts and copy it link and place it in the head of our index.html file in public directory.***
* *In react we write the inline css as an object containing key as the css property and value as value to that property.*

***Style={{background: ‘Red’ }}***

* *To create a new branch and checkout at same time we have the command:*

***Git checkout -b < branchname>***

***If we use***

* ***npm => Package-lock.json***
* ***yarn => yarn-lock.json***

***To store our installed dependencies with their version itself/***