**ReactJS-HOL 3**

**Questions & Answers –**

* **Explain React components**

**Ans -** In React, components are the fundamental building blocks of user interfaces (UIs). Think of them as independent, reusable pieces of code that encapsulate their own logic and rendering. They allow you to split the UI into independent, reusable pieces, and think about each piece in isolation. Essentially, a component is a JavaScript function or a JavaScript class that optionally accepts inputs called "props" (properties) and returns React elements describing what should appear on the screen.

* **Identify the differences between components and JavaScript functions**

**Ans –**

|  |  |  |
| --- | --- | --- |
| **Feature** | **React Component** | **JavaScript Function (General Purpose)** |
| **Purpose** | Executes a specific task, returns a value. | Describes a piece of UI, returns React elements. |
| **Input** | Arguments | props object (a special object provided by React) |
| **Output** | Any data type (number, string, object, etc.). | React elements (JSX), which React uses to build the DOM |
| **State** | Typically stateless, or state managed externally. | Can manage its own internal state (for class components, or using Hooks for functional components) |
| **Lifecycle** | No inherent lifecycle. | Has a well-defined lifecycle (mounting, updating, unmounting) that React manages |
| **Reusability** | Can be reused for various computational tasks. | Designed for UI reusability, building complex UIs from smaller parts. |
| **Naming** | Typically camelCase | PascalCase (conventionally, especially for functional components) |

* **Identify the types of components**

**Ans –** React had two primary types of components:

1. Class Components
2. Functional Components

With the introduction of Hooks in React 16.8, functional components gained the ability to manage state and side effects, making them the preferred way to write new components today. However, understanding class components is still important for working with older codebases or specific advanced use cases.

* **Explain class component**

**Ans -** A class component is a JavaScript ES6 class that extends React.Component. It uses a render() method to return JSX that describes the component's UI. Class components can also hold and manage their own internal state and have access to lifecycle methods.

**Example:**

import React from 'react';

class Welcome extends React.Component {

constructor(props) {

super(props);

this.state = {

message: "Hello"

};

}

render() {

return (

<div>

<h1>{this.state.message}, {this.props.name}!</h1>

<button onClick={() => this.setState({ message: "Welcome" })}>Change Message</button>

</div>

);

}

}

export default Welcome;

* **Explain function component**

**Ans -** A functional component (also known as a "stateless functional component" before Hooks) is a plain JavaScript function that accepts props as an argument and returns React elements (JSX). With React Hooks (introduced in React 16.8), functional components can now manage state and side effects, making them incredibly powerful and the recommended way to write components.

**Example:**

import React, { useState } from 'react';

function GreetUser(props) {

const [count, setCount] = useState(0); // Using the useState Hook for state

return (

<div>

<h2>Hello, {props.name}!</h2>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>Click me</button>

</div>

);

}

export default GreetUser;

* **Define component constructor**

**Ans -** The constructor in a React class component is a special method called when an instance of the component is created. It's the first method called in the component's lifecycle.

**Example:**

class MyComponent extends React.Component {

constructor(props) {

super(props); // Essential!

this.state = {

counter: 0,

text: "Initial Text"

};

}

}

* **Define render() function**

**Ans -** The render() function is a mandatory method in a React class component. Its primary responsibility is to return the React elements (written in JSX) that describe what should be displayed on the screen.

**Hands-On: (Code)**

*CalculateScore.js –*

import '../Stylesheets/mystyle.css'

const percentToDecimal = (decimal) => {

    return (decimal \* 100).toFixed(2) + '%'

}

const calcScore = (total, goal) => {

    return percentToDecimal(total/goal)

}

export const CalculateScore = ({Name, School, total, goal}) => {

    return(

    <div className="formatstyle">

        <h1><font color="Brown">Student Details:</font></h1>

        <div className="Name">

            <b><span>Name: </span></b>

            <span>{Name}</span>

        </div>

        <div className="School">

            <b><span>School: </span></b>

            <span>{School}</span>

        </div>

        <div className="Total">

            <b><span>Total: </span></b>

            <span>{total}</span>

            <span> Marks</span>

        </div>

        <div className="Score">

            <b>Score:</b>

            <span>

                {

                    calcScore(total, goal)

                }

            </span>

        </div>

    </div>

    )

}

*Mystyle.css –*

.Name{

    font-weight: 300;

    color: red;

}

.School{

    color: aqua;

}

.Total{

    color: blueviolet;

}

.formatstyle{

    text-align: center;

    font-size: large;

}

.Score{

    color: chartreuse;

}

*App.js –*

import './App.css';

import { CalculateScore } from './components/CalculateScore';

function App() {

  return (

    <div>

      <CalculateScore Name={"Naveen"}

      School = {"KL University"}

      total = {592}

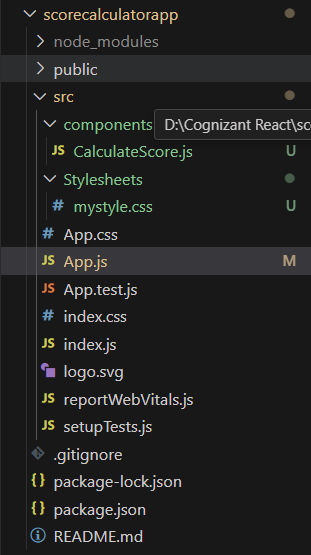
      goal = {600} />

    </div>

  );

}

export default App;



**OutPut:**

