React JS (class 1)

Classroom
Page No.
Date / / /

→ React JS is an open source Javascript library developed by Fdcebook. It is used for building interactive user interfaces and web applications quickly and efficiently with significantly less code.

→ React is a Javascript library, and its sole purpose is to create user interfaces.

→ It is commonly used for creating interactive and dynamic web pages.

→ React allows developers to efficiently update and render components of a web page in response to user interactions, making it a powerful tool for building modern web applications.

* What is Library?

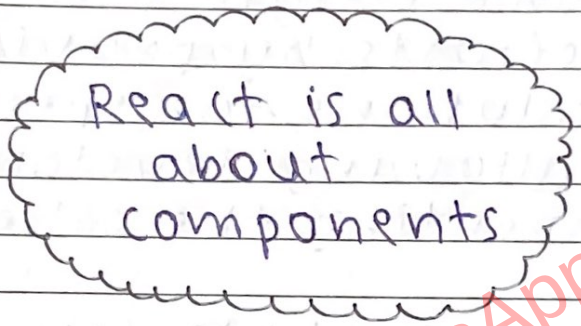
In the context of software deve-lopment, a library is a collection of pre-written code or functions that can be reused by developers to perform common tasks or functions in their applications. Libraries are designed to save time and efforts by providing pre-built solutions for specific problems, allowing developers to avoid reinventing the wheel.

Libraries can include functions, classes, and modules that encap-sulate certain functionalities, making it easier for developers to integrate these features into their own software without having to write the code from scratch. This promotes code reusa-bility, maintainability, and efficiency in software developm-ent.

React, as mentioned earlier, is a library for building user interfac-es in web application's. It provides a set of reusable components and

functions that simplify the process of creating interactive web interfaces, but it doesn't dictate the entire structure of your application, allowing developers to use it alongside other libraries and tools as needed.

— X — X — X — X — X — X —



React is a JS library that revolves around the concept of components.

→ In React, we deal with the components only, and majority of time we create component in our code.

→ components are the building blocks of a React application, & the entire user interface is typically broken down into

hierarchy of reusable component.

* what is component?

In the context of software develo-pment, a component is a self-contained, reusable building block that encapsulates a speci-fic piece of functionality or user interface (UI) element. components are a fundamental concept in many programming frameworks and libraries, inclu-ding React.

** At the end, component ek function hi hai jisko hum zrurat ke time baar baar use krte hai, jaise baaki programming languages me function hote hai, bss React me function ko component bolte hai **

In Hindi, component ek reusable piece of code hai, jisko hum again and again use krte hai, so that copy paste na krna pde, normal programming languages me functi-ons create krte hai, React me components create krte hai.

X — X — X — X — X —

Ques =. while we can do all things (that React do) with only Javascript, then why there is a need of React Library?

While its true that we can build web applications using only Javascript, React and similar libraries or frameworks exist to simplify & enhance the process in several ways.

→ Basically, React follow component based approach and Javascript follow Imperative Approach.

* Imperative Approach :-

- In an imperative approach, software is typically built using procedures or step by step instructions that explicitly specify how to achieve a certain task. It focuses on describing the sequence of actions.

- Imperative code often involves managing the state of the program explicitly, tracking changes in variables, and direct

-ly manipulating the programs state.

- Imperative code tends to be less modular and reusable because it can be tightly coupled and hard to decouple into smaller, independent units.

* Component Based Approach :-

- In a component based approach, the software is built by creati -ng self-contained, reusable components. Each component encapsulates a specific piece of functionality or user inter -face element.

- components are modular and can be composed to create complex systems. This promote code reusability and maintai -nability, as components can be reused across different parts of an application or even in different applications.

→ In normal Javascript, we use imperative approach, means humko ek ek step ko describe krna pdta hai that, kis element ko kaha se uthana, uske ander kya changes krne hai, uske ander kya text place krna hai & uske ander koi new element place krna hai and kis location pe krna hai, ye sb humko explicitly btana pdta hai.

→ But in React, we use component based approach, means isme bss humko end state define krni hoti hai and thats it, baaki React apne aap smbhaal lega, end state means hum bss itna bolenge that UI pe ek element bnao uske background me grey color dedo, uske upar 4 boxes bnao & 4 boxes me color Red, Green, Blue and Yellow krdo, thats our end state.

X — X — X — X — X — X —

→ In React, generally we use SPA (single Page Application) approach

→ In the context of React, a SPA is a web application that loads a single HTML page & dynamically updates its content as the user interacts with the app, without requiring a full page reload from the server. React is the popular library for building SPA's.

React me hmare pass multiple files nhi hoti, ek single HTML file hoti hai and ek baar server me ye HTML file load krdi, uske baad baaki saare changes dynami-cally hote hai. Dynamically changes means jaise weather app me temprature ko hum API call se leke aa rhe the, thats called dynamically changes.

— X — X — X — X — X —

Steps for creating a React App :-

i) NodeJS should be installed.
ii) create a new folder in which we create our React app, suppose its name is "ReactFolio".
iii) change directory to "ReactFolio".
iv) run this command, i.e., "npx create-react-app demoapp"
this is the name of our react app, we can use any name here, but make sure that there should be no space and all alphabets are small, i.e., lower-case.
v) now, change directory to "demoapp"
vi) to run the app, write "npm start", and hit enter ↵

As we have created the React App successfully, we do all the changes in that only.

There are many files in the React structure which are create-d by default in the app, some files plays very crucial role in the React App development, these files are :-

(1) "package.json" :- the 'package.json' file in a React project serves as a central configuration file. It contains important information about the project, such as its name, version, description, and author. This metadata helps identity and manage the project. One of its key roles is to list dependencies are specified in the "dependencies" section. This ensures that all team members use the same versions of these dependencies, preventing compa-tibility issues.
In the "scripts" section of "packa-ge.json", you define custom commands for tasks like starting a development server, running tests, or building the production version of the app. This makes it easy to automate common develo-pment tasks and ensures consist

-ency across the team.
Additionally, the "package.json" file may include a ".gitignore" field, specifying files and directo-ries to be excluded from version control. This is crucial for keeping the repository clean and avoiding unnecessary commits of developme-nt artifacts and build outputs. Overall, the "package.json" file is a central hub for managing the configuration, dependencies and scripts in a React project, making it an essential part of modern web development with React.

(2) "index.js" :- In a React application, the 'index.js' file plays a crucial role as the entry point for your application. Its typically one of the first files that gets executed when the React app starts.

(3) "app.js" :- In a React application, the 'app.js' file is typically the main entry point where you initialize and render your root React component into the HTML document. It imports necessary libraries, like React and ReactDOM, and often imports your top-level

component, such as 'App', which represents the core of your applications user interface. The 'ReactDOM.render' function is used to render this component into a specific HTML element, and any initial configuration or setup can be done in this file. while 'app.js' is a common naming convention, you can choose a different name for this file if you prefer.

— x — x — x — x — x —

- index.js :-

```
import React from 'react';
import ReactDOM from 'react-dom
                        /client';
import './index.css';
import App from './App';

const root = ReactDOM.createRoo
    -t(document.getElementB
    -yId('root'));

root.render(<App/>);
```

- index.html :-

```
<html>
  <head>
    ———
    ———
  </head>
  <body>
    <div id="root">
    </div>
  </body>
</html>
```

- App.js :-

```
import logo from './logo.svg';
import './App.css';

function App()
{
  return
  {
    <div className="App">
      Hello Hello
    </div>
  };
}

export default App;
```

Classroom
Page No.
Date

Now, we will see the previous files working :-

As we know that React App's starting point is "index.js", we first look into that file,

→ In 'index.js' file, hmne React ka root element create kiya hai, and to create that root element hmne ek element fetch kiya hai using the id and if we check the index.html file, fetched id ke corresponding ek div created hai, and hmne uss div ko React ka root element bnaya hai.

→ After that, uss React ke root element me hmne 'App' component ko render kiya hai, and that App component lies in 'App.js' file.

→ If we look at App.js file, the App component is nothing but a function, and App is the first component in React, and in that App component we have returned the HTML like code, its not an HTML code, but it is similar syntax to HTML, this is JSX basically, i.e., Javascript XML, JSX is basically, HTML written inside Javascript.

"src" folder:

- app.js
- index.css
- index.js

} most important files

"public" folder:

- index.html

— X — X — X — X — X —

Let suppose hum Javascript me koi bhi function ya class create krr rhe hai, and we want to use this function or class in some other file, then we have to make sure that jaha bhi humne uss function ya class ko create kiya waha unhe export kiya hona chahiye and jaha bhi hum unhe use krna chahte hai waha unko import kiya hona chahiye.

— X — X — X — X — X — X —

Jaise hi hum React Ka code complete krke "npm start" comm-and run krte hai, to ye comma-nd saare React Ke code KO equivalent HTML and JS Ke code me convert krti hai, jo at the end browser understand krta hai.

X ——— X ——— X ——— X ——— X

creating our own component :-

creating a component in React is not a very difficult task, to create a component we have to just create a js file and its corresponding css file.

** while creating a component, the best practice is that the first alphabet of the name of the component is always in upperca-se. For example :- Item compon-ent, Profile component, etc.

Now, we create the component in the js file, suppose the name of the component is Item, so we create the js file with the name 'Item.js'

And as we know component is nothing but a function, so we create a function that is, inside the js file.

Item.js,

```
// linking Item.css file
import './Item.css';

// defining the function
function Item() {
  return (
    <p className="singer">
      Sidhu Moosewala
    </p>
  );
}

// whenever we create a compone
  -nt, make sure to export it
export default Item;
```

JSX syntax

Item.css,

```
.singer{
   text-align: center;
}
```

Now, if we want to use our own custom created compone -nt in "App.js", we have to import it first.

App.js,

```
// importing App.css file
import './App.css';

// import our custom created
component
import Item from './components/
                    Item';

function App()
{
    return (
        // parent container
        <div>
            // using the Item compon
            -ent
            <Item></Item>
            <div className="App">
                Hello Ji..
            </div>
        </div>
    );
}

export default App;
```

Ques. why we write "className" in React instead of "class" in JS file?

In React, when defining components using JSX (Javascript XML), we use "className" instead of "class" to specify CSS class names for elements. This is because JSX is a syntax extension for Javascript, and it needs to be converted into regular Javascript before it can be executed in the browser.

Here's why we use "className" in React instead of "class":-

i) Avoiding confusion: In JSX, the "class" attribute is already used for defining the class of a Javascript element. However, in HTML, the "class" attribute is used to define CSS classes. To avoid confusion between Javascript classes and CSS classes, React uses "className" for specifying CSS classes.

ii) JSX Resembles HTML :- React's JSX syntax closely resembles HTML, and it was designed this way intentionally to make it more familiar to developers. while HTML uses 'class' for specifying CSS classes, React chose 'className' to stay consistent with Javascript conventions and to avoid conflicts.

iii) Avoid Reserved words :- 'class' is a reserved keyword in Javascript used for defining classes and constructors functions. By using 'className' in JSX, React ensures that there is no collision with this reserved keyword.

As we know that React is all about components, and components are reusable piece of code we can reuse the components whereever we need them, now we try to use previous Item and ItemDate components in App.js file,

App.js,

```
import './App.css';

import Item from './components/
                    Item';
import ItemDate from './compone
                -nts/ItemDate';

function App()
{
  return(
    <div>
      <Item></Item>
      <ItemDate></ItemDate>

      <Item></Item>
      <ItemDate></ItemDate>

      <Item></Item>
      <ItemDate></ItemDate>
```

```
            <div className="App">
                Hello Ji
            </div>
        <div>
    );
}

export default App;
```

Item.js,

```
import './Item.css';

function Item()
{
    return(
        <p className="singer">
            Sidhu Moosewala
        </p>
    );
}

export default Item;
```

ItemDate.js,

```
import './ItemDate.css';

function ItemDate()
{
    return (
        <div className="datediv">
            <span>11</span>
            <span>June</span>
            <span>1993</span>
        </div>
    );
}

export default ItemDate;
```

Now, as we have reused Item and ItemDate components in App.js file, 3 times. But all the time the same content is displaying on the browser. So, there is no such reusability concept left in the code.

So, there is a concept in React called Props.

* **Props**

In React, "props" is short for "properties" and it is a mechanism for passing data from one component to another. Props are a fundamental concept in React and are used to make your components reusable and configurable.

Using Props we can pass data from Parent component to child component only.

Item.js,

```
// linking Item.css file
import './Item.css';

// creating our first component
// to use props we have to pass
'props' keyword as a parameter
function Item(props)
{
    // props pass kyte hue jo attribute
    ka naam hmne App.js me pass
    kiya hai uss parameter/attribu
    -te ka naam hi use krna
    pdegga yaha pe
    const singerName = props.
                        name;

    return (
        <p className="singer">
            {singerName} // jo props
                ne value di hai
                usko aise curly
                braces me pass krke
                use krenge
        </p>
    );
}

export default Item;
```

ItemDate.js,

```
import './Item.css';

function ItemDate(props)
{
    const itemDate = props.date;
    const itemMonth = props.month;
    const itemYear = props.year;

    return (
        <div className="datediv">
            <span>{itemDate}</span>
            <span>{itemMonth}</span>
            <span>{itemYear}</span>
        </div>
    );
}

export default ItemDate;
```

App.js,

```
import './App.css';

// importing both Item and
ItemDate components
import Item from './components/
                        Item';
import ItemDate from './compone
        -nts/ItemDate';

function App()
{
  const firstSingerYear = "1993";
  const secondSinger = "Diljit";
  const thirdSingerMonth =
                        "January";

  return (
    <div>
      // now we are using Props
      to pass data from App
      component to Item and
      ItemDate component
      <Item name="Sidhu">
      </Item>
      <ItemDate date="11"
        month="June"
        year={firstSingerYear}>
      </ItemDate>
```

passing data from App component to Item and ItemDate component

```
<Item name={secondSinger}>
</Item>
<ItemDate date="06"
  month="January"
  year="1984">
</ItemDate>
```

passing data from App component to Item and ItemDate component

```
<Item name="Karan">
</Item>
<ItemDate date="18"
  month={thirdSingerMonth}
  year="1997">
</Item>
<div className="App">
  Hello Ji
</div>
</div>
);
}

export default App;
```

In previous codes we have seen that, we are passing data from App component to Item and ItemDate component, and we are reusing the components.

But still we have to write too many code while passing data from App component to Item and ItemDate component, as we have only 4 components, but suppose we have 50 components, then we have to write code for 50 components.

→ Suppose, we have called an API, and that API gives us an array in response that contains 3 objects, then how we can fetch the data from the array and pass the data in another component.

App.js,

```
import './App.css';

// importing Item and ItemDate
component
import Item from './components/
                        Item';
import ItemDate from './compone
                    nts/ItemDate';

function App()
{
  // currently we are hard coding
   the array but suppose we are
    getting this array as a respon
   -se from API call
   const response = [
      {
          itemName : "Sidhu",
          itemDate : "11",
          itemMonth : "June",
          itemYear : "1993"
      },
      {
          itemName : "Diljit",
          itemDate : "06",
          itemMonth : "June",
          itemYear : "1984"
      }
```

```
{
    itemName : "Karan",
    itemDate : "18",
    itemMonth : "January",
    itemYear : "1997"
  }
];

// now we will use the data by
fetching it from the array
return (
  <div>
    <Item name={response
      [0].itemName}>
    </Item>
    <ItemDate date={response
      [0].itemDate} month=
      {response[0].itemMonth}
      year={response[0].item
        -Year}>
    </ItemDate>

    <Item name={response[1]
                .itemName}>
    </Item>
    <ItemDate date={response
      [1].itemDate} month=
      {response[1].itemMonth}
      year={response[1].itemY
        -ear}>
    </ItemDate>
```

fetching data from array & passing it to another component

fetching data from array & passing it to another component.

```
<Item name={response[2].itemName}>
</Item>
<ItemDate date={response[2].itemDate}
month={response[2].itemMonth} year={
response[2].itemYear}>
</ItemDate>

<div className="App">
Hello Ji
</div>
</div>
);
}

export default App;
```

* Props children

Till now we are using our components like this :-

```
<Item name={response[0].itemName}>
</Item>
```

→ we are printing the content using "Props".

But, what if we write some content inside the component tag, i.e.,

```
<Item name={response[0].itemName}>
    Printing the first item
</Item>
```

→ Here, we write some text inside the component tag.

But this text will not print on the browser, and this is because it is the default working of the component, that the text or anything written inside the component tag will not be print-ed on the browser.

So, here if we want to show that content we use the concept of "props children".

"In React, "Props children" allow us to pass and render content or component inside a parent component. It lets us insert custom content between the opening and closing tags of a parent component, making it more versatile and reusable. This is useful for creating container components that can wrap and style their children components or content, providi-ng a flexible way to compose complex user interfaces."

If we want to render the content written inside the component tags, we use "props.children" inside components JS file.

App.JS,

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

function App()
{
    const itemName = "sidhu";
    return (
        <div>
            <Item name={itemName}>
                Printing first item
            </Item>
        </div>
    );
}

export default App;
```

Item.js,

```
import './Item.css';

function Item(props)
{
    const singerName = props.
                        name;

    return (
        <div>
            <p className="singer">
                {singerName}
            </p>

            Where we are using
            props children to print
            the content written
            inside the component
            tag in App.js file
            <p className="itemDiv">
                {props.children}
            </p>
        </div>
    );
}

export default Item;
```

Item.css,

```
.singer{
    text-align: center;
    background-color: lightgreen;
}

.itemDiv{
    text-align: center;
    background-color: lightblue;
}
```