Unit -4

React JS

React JS - Introduction

- Angular, Vue, and React are some of the most-used Front-end frameworks.
- A Facebook software engineer, Jordan Walke, created React. In 2011, React was launched on Facebook's newsfeed, later in Instagram as well. but it was released to the public in the month of May 2013.
- ReactJS is an Open-Source JavaScript library used to create user interfaces for single-page applications.
- It's responsible for the view layer in web and mobile apps. We can also make reusable UI components with React.
- React is a framework for building client-side dynamic web applications.
- React uses dynamic data binding and a virtual DOM to extend HTML syntax and to eliminate the need for code that keeps the user interface (UI) elements synchronized with the application state.
- ReactJS is a JavaScript library designed for crafting dynamic and interactive applications, elevating UI/UX for web and mobile platforms. Operating as an open-source, component-based front-end library, React is dedicated to UI design and streamlines code debugging by employing a component-oriented approach.

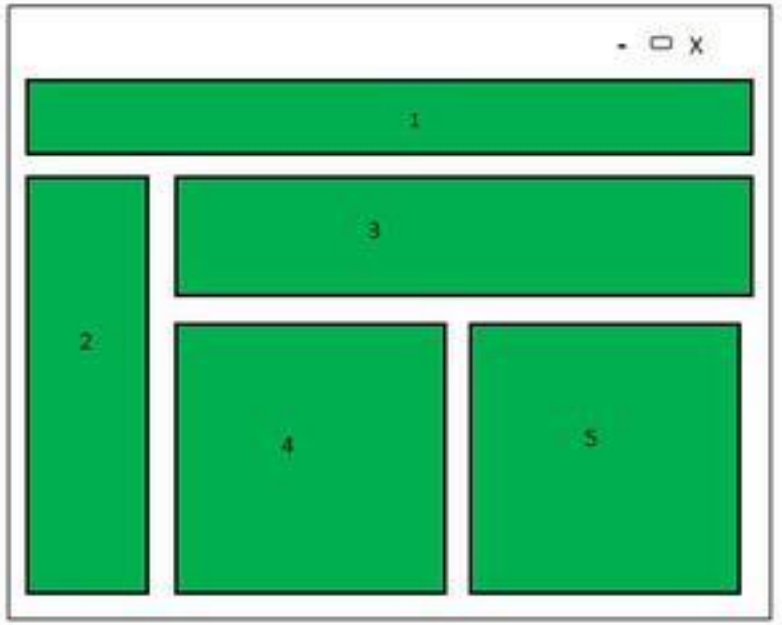
Basic Features of React

1. Components
2. JSX (JavaScript Syntax Extension)
3. Virtual DOM
4. Unidirectional Data Flow
5. State and Props
6. One-Way Data Binding
7. Lifecycle Methods
8. React Hooks
9. Performance
10. Extension
11. Simplicity

Basic Features of React

1. Components

   Everything in React is treated as a component. This means that the development of a web interface or application in React is made up of several components. These components are reusable, which in turn helps in code maintenance when working on larger projects. ReactJS divides the web page into multiple components. Components allow you to break down complex UIs into smaller, more manageable pieces, making it easier to maintain and scale your code. Each component is a part of the UI design which has its own logic and design as shown in the image. So the component logic which is written in JavaScript makes it easy and run faster and can be reusable.



Basic Features of React

2. JSX (JavaScript Syntax Extension)

JSX is one of the best features of React as it makes it extremely simple for developers to write the building blocks. JSX is a combination of HTML and JavaScript. You can embed JavaScript objects inside the HTML elements. This makes it easier to create and modify the structure of your UI directly from your JavaScript code. JSX is not supported by the browsers, as a result, Babel compiler transcompile the code into JavaScript code. JSX makes codes easy and understandable. It is easy to learn if you know HTML and JavaScript.

- A transcompiler, also called a source-to-source compiler or transpiler, is a special type of compiler that converts a program's source code into another language. It can also process a program written in an older version of a programming language, converting it to a newer version of the same language.

Basic Features of React

3. Virtual DOM

It is the most important part of the web as it divides into modules and executes the code. Usually, JavaScript Frameworks updates the whole DOM at once, which makes the web application slow. React JS uses a virtual DOM to minimize the number of direct manipulations to the actual DOM. The virtual DOM is a lightweight in-memory representation of the actual DOM. When a component's state changes, React calculates the difference between the new virtual DOM and the old one and will only update the changed elements. This process, known as "reconciliation," leads to faster and more efficient updates.

Basic Features of React

4. Unidirectional Data Flow

React enforces a one-way data flow. Data flows from parent to child components through a props system (short for properties). This unidirectional flow makes it easier to understand how changes in data affect the UI, reducing the chances of unexpected side effects and making debugging more straightforward.

The Flux JS application's architecture component aids in the unidirectional flow of data. This increases the application's flexibility, which leads to increased efficiency.

5. State and Props

React components can have their own internal state, which is an object that holds the values of various properties that can change over time. State is managed within the component, and when it changes, the component re-renders. Props, on the other hand, are immutable and are passed from parent components to child components.

Basic Features of React

6. One-Way Data Binding

   React enforces a one-way data flow, which means that data is passed from parent components to child components through props. This makes it easier to debug the application, as data flows in a single direction.

7. Lifecycle Methods

   React components have several lifecycle methods that allow you to run code at specific points in a component's life. Some common lifecycle methods include componentDidMount, componentDidUpdate, and componentWillUnmount. These methods can be used to perform actions like making API calls, updating state, or cleaning up resources.

8. React Hooks

   React Hooks are a relatively new addition to React that allows you to use state and other React features in functional components. Hooks like useState, useEffect, and useContext make it easier to write more concise and readable code.

Basic Features of React

9. Performance

   React uses virtual DOM and updates only the modified parts which makes the DOM to run faster. DOM executes in memory so we can create separate components which makes the DOM run faster.

10. Extension

    React has many extensions that we can use to create full-fledged UI applications. It supports mobile app development and provides server-side rendering. React is extended with Flux, Redux, React Native, etc. which helps us to create good-looking UI.

11. Simplicity

    React is a component-based which makes the code reusable and It uses JSX which is a combination of HTML and JavaScript. This makes code easy to understand and easy to debug and has less code.

Factor	AngularJS	React JS
Author	Google	Facebook
Developer	Misko Hevery	Jordan Walke
First Release	Oct 2010	May 2013
Language	JavaScript, HTML	JSX
Type	Open Source MVC Framework	Open-Source JS Framework
Rendering	Client-Side	Server-Side
Packaging	Weak	Strong
Data-Binding	Bi-Directional	Uni-Directional
DOM	Regular DOM	Virtual DOM
Dependencies	Manage Automatically	Need additional tools to manage dependencies
App Architecture	MVC (Model View Controller)	Flux
Routing	It requires a template or controller to its router configuration, which has to be managed manually.	It doesn't handle routing but has a lot of modules for routing, eg., react-router.
Performance	Slow	Fast, due to virtual DOM.
Best For	Single page applications that update a single view at a time.	Single page applications that update multiple views at a time.

Factors	ReactJS	Vue
Definition	React is a declarative, efficient, flexible, open-source JavaScript library for building reusable UI components.	Vue is an open-source JavaScript library for building reusable user interfaces and single-page applications.
History	It was created by Jordan Walke, a software engineer at Facebook. It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram. Facebook developed React in 2011 for the newsfeed section, but it was released to the public on May 2013.	Vue was created by Evan You, a former employee of Google worked on many Angular projects. He wanted to make a better version of Angular, just extracting the part which he liked about Angular and making it lighter. The first release of Vue was introduced in February 2014.
Learning Curve	React is not a complete framework, and the more advanced framework must be looked for the use of third-party libraries. It makes the learning of the core framework not so easy. It adds some complexity to the learning curve as it differs based on the choices you take with additional functionality.	Vue provides higher customizability, which makes it easier to learn than Angular or React. Vue shares some concepts with Angular and React in their functionality. Hence, the transition to Vue from Angular and React is an easy option. Also, the official documentation is well written and covers everything the developer needs to build a Vue app.
Preferred Language	JavaScript/JavaScript XML	HTML/JavaScript
Size	The size of the React library is 100 kilobytes (approx.).	The size of the Vue library is 60 kilobytes (approx.).

Factors	ReactJS	Vue
Performance	Its performance is slow as compared to Vue.	Its performance is fast as compared to React.
Flexibility	React provides great flexibility to support third-party libraries.	Vue provides limited flexibility as compared to React.
Coding Style	React uses JSX for writing JavaScript Expression instead of regular JavaScript. JSX is similar to HTML code within the JavaScript expressions. React takes everything as Component, and each component has its own lifecycle methods.	Vue coding style is little similar to Angular. It separates HTML, JS, and CSS as like web developers have been used to the web development scenario for years. But, it also allows using JSX if you prefer. Vue's take of the component lifecycle more intuitive than React's.
Data Binding	React supports one-way data binding.	Vue supports both one-way and two-way data binding.
Tooling	React has great tooling support. It uses third-party CLI tool (create-react-app), which helps to create new apps and components in React Project. It has excellent support for the major IDEs.	Vue provides limited tooling support as compared to React. It has a Vue CLI tool, which is similar to the create-react-app tool. It gives supports for major IDEs but not as good as React.
Current Version	React 16.8.6 on March 27, 2019	Vue 2.6.10 on March 20, 2019.
Long Term Support	It is suitable for long term supports.	It is not suitable for long term support.

JavaScript XML (JSX)

- JSX is a syntax extension designed to run on modern web browsers.
- JSX optimizes performance in web browsers.
- Particularly popular in React applications, JSX optimizes code performance by compiling source code into JavaScript that runs faster than its equivalent code written directly in JavaScript.
- You can use JSX files to render React components, import CSS files, and use React hooks.
- JSX collaborates with standard technologies such as JavaScript, TypeScript, and CSS blocks.
- The JSX code can sometimes look heavy and confusing, causing challenges for developers.

JavaScript XML (JSX)

JSX or JavaScript Syntax Extension or JavaScript XML:

- Extension of JavaScript
- Easier way to create UI components in React
- Produces React “elements”

JSX syntax:

- Is like an XML or HTML-like syntax used by React that extends ECMAScript
- Allows XML or HTML-like text to co-exist with JavaScript or React code
- To be used by preprocessors like transpilers or compiler like Babel, to transform HTML-like text found in JavaScript files into standard JavaScript objects.
- Objects can then be parsed by a JavaScript engine.

JavaScript XML (JSX)

- JSX code
  - Like HTML uses a JavaScript-like variable
  - A syntax extension of regular JavaScript, used to create React elements
  - Elements are rendered to React Document Object Model or DOM
  - Browsers don't understand JSX, so you need to use Babel to compile your JSX code. **create-react-app** command is used for this compilation.
- Commands
  1. npx create-react-app my-react-app

     npx - Node Package eXecute
  2. cd my-react-app
  3. npm start

     npm - Node Package Manager

```
const element1 = <h1>This is a sample JSX code Snippet</h1>
```

JavaScript XML (JSX)

- In the below example, **MyComponent** is a basic React component that renders a simple greeting message.

```
import React from 'react';


function MyComponent() {
  return <div>Hello, World!</div>;
}


export default MyComponent;
```

- The JSX syntax allows for the seamless integration of HTML-like markup within JavaScript code, enhancing the readability and maintainability of React components.

Benefits of JSX

- JSX takes care of the usual output sanitization issues to prevent attacks such as cross-site scripting.
- Easily understood by less technical people.
- You can avoid learning of a templating language.
- It is type-safe, and most of the errors can be found at compilation time.
- It promotes inclusion of inline styles.
- It helps in keeping your code simple and elegant.
- Most people find it helpful like a visual aid.
- It is faster than normal JavaScript.
- it takes care of output sanitization issues.

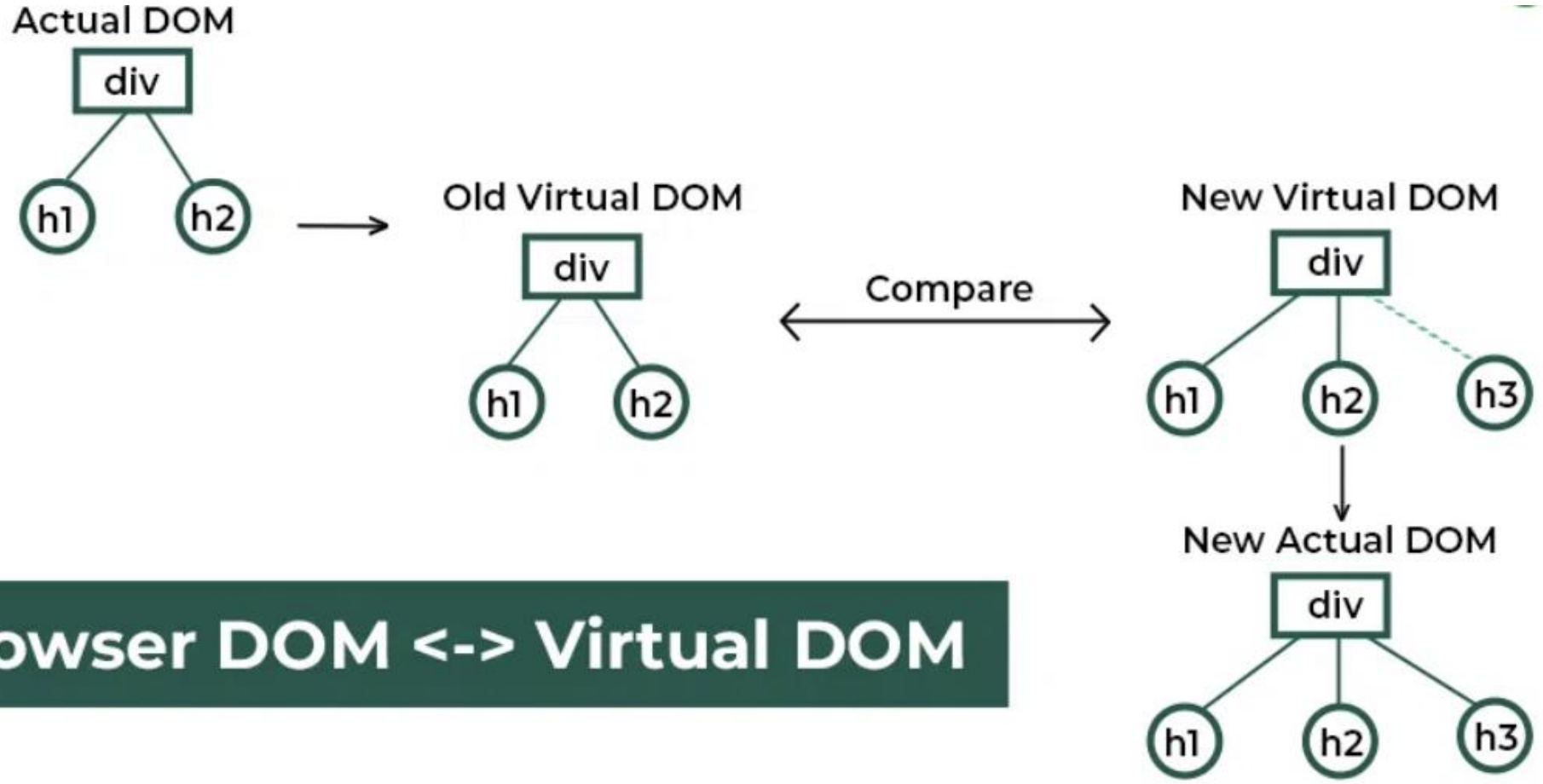
Virtual DOM

- Virtual DOM is the virtual representation of Real DOM.
- Virtual DOM is a lightweight copy of the actual DOM.
- React update the state changes in Virtual DOM first and then it syncs with Real DOM
- Virtual DOM is just like a blueprint of a machine, can do changes in the blueprint but those changes will not directly apply to the machine.
- Virtual DOM is a programming concept where a virtual representation of a UI is kept in memory synced with “Real DOM ” by a library such as ReactDOM and this process is called “Reconciliation”.
- Virtual DOM makes the performance faster, not because the processing itself is done in less time. The reason is the amount of changed information – rather than wasting time on updating the entire page, you can dissect it into small elements and interactions.

Virtual DOM

- For every object that exists in the original DOM, there is an object for that in React Virtual DOM. It is exactly the same, but it does not have the power to directly change the layout of the document.
- Manipulating DOM is slow, but manipulating Virtual DOM is fast as nothing gets drawn on the screen. So, each time there is a change in the state of our application, the virtual DOM gets updated first instead of the real DOM.
- When anything new is added to the application, a virtual DOM is created, and it is represented as a tree.
- Each element in the application is a node in this tree. So, whenever there is a change in the state of any element, a new Virtual DOM tree is created.
- This new Virtual DOM tree is then compared with the previous Virtual DOM tree and make a note of the changes. After this, it finds the best possible ways to make these changes to the real DOM. Now only the updated elements will get rendered on the page again.
- React maintains two Virtual DOM at each time, one contains the updated Virtual DOM and one which is just the pre-update version of this updated Virtual DOM. Now it compares the pre-update version with the updated Virtual DOM and figures out what exactly has changed in the DOM like which components have been changed. This process of comparing the current Virtual DOM tree with the previous one is known as ‘diffing’. Once React finds out what exactly has changed then it updates those objects only, on real DOM.

Virtual DOM


Actual DOM
div
h1
h2
Old Virtual DOM
div
h1
h2
Compare
New Virtual DOM
div
h1
h2
h3
New Actual DOM
div
h1
h2
h3
Browser DOM <-> Virtual DOM

Basic React app

Steps to Create React Application:

- Step 1: Install create-react-app using the following command on terminal:
  - npm install -g create-react-app
- Step 2: Create a react application using the following command:
  - npx create-react-app foldername
- Step 3: Change your directory to the newly created application using the following command
  - cd foldername
- Step 4: Inside App.js and write down the following code as shown below:

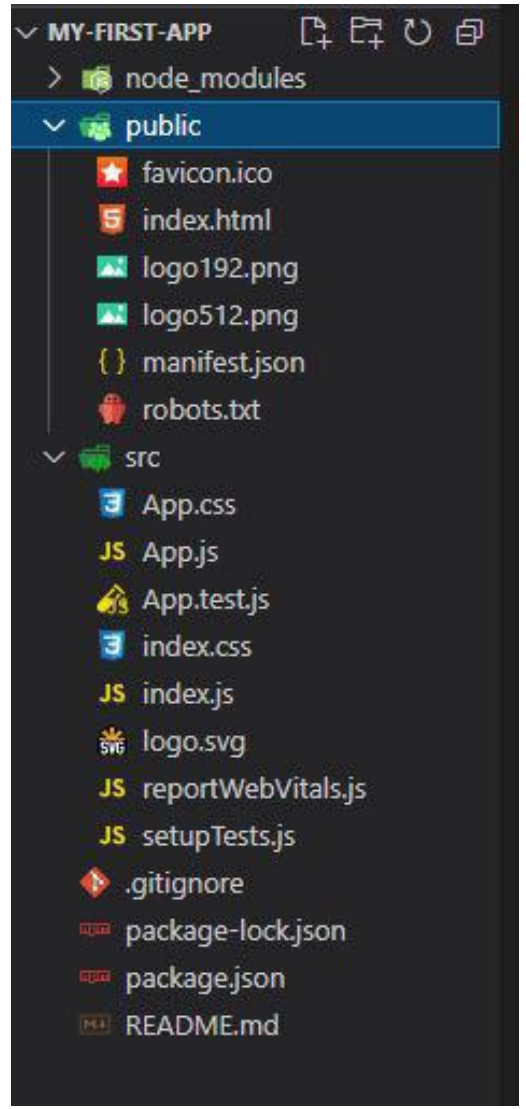
```
import React from 'react';
import './App.css';

function App() {
    return (
        <h1> Hello World! </h1>
    );
}

export default App;
```

- Step 5: Run the application using the following command:
  - npm start

Project Structure:



Components: Functional and Class

- Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.
- The component's name *must* start with an uppercase letter
- Components come in two types: Class components and Function components

Components: Functional and Class

- A class component must include the extends React.Component statement. This statement creates an inheritance to React.Component and gives your component access to React.Component's functions.
- The component also requires a render() method, this method returns HTML.
- Create a Class component called Student

```
class Student extends React.Component {
        render() {
                return <h2>Hi, I am a Student!</h2>;
        }
}
```

Functional Component

- A Function component also returns HTML and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand.
- Create a Function component called Student

```
function Student() {
        return <h2>Hi, I am a Student!</h2>;
}
```

Component

Feature	Functional Component	Class Component
Syntax	Simple function	ES6 class
State Management	useState hook	this.state, setState
Lifecycle Methods	useEffect	componentDidMount, etc.
Readability	Cleaner & concise	Verbose
Performance	Slightly better in hooks	Slightly heavier

Rendering a Component

- Now your React application has a component called Student, which returns an <h2> element.
- To use this component in your application, use similar syntax as normal HTML: <Student />
- Display the Student component in the "root" element:

- ```
  const root =ReactDOM.createRoot(document.getElementById('root'));
  root.render(<Student />);
  ```
  ```

Props and State

- Props stands for properties.
- A prop is an object accessible to all React components. It serves as a means to pass data from a parent component to a child component.
- Props are arguments passed into React components.
- Props are passed to components via HTML attributes.
- Example: Add a “course” attribute to the Student element:

```
const myElement = <Student course=”B.Tech." />;
```

- The component receives the argument as a props object:
- Use the course attribute in the component:

```
function Student(props) {
    return <h2>I am a { props.course } student!</h2>;
}
```

Props and State

- State in React:
  - A state is a variable that exists inside a component, that cannot be accessed and modified outside the component and can only be used inside the component. Works very similarly to a variable that is declared inside a function that cannot be accessed outside the scope of the function in normal javascript. State can be modified using this.setState. The state can be asynchronous. Whenever this.setState is used to change the state class is rerender itself.

PROPS	STATE
The Data is passed from one component to another.	The Data is passed within the component only.
It is Immutable (cannot be modified).	It is Mutable (can be modified).
Props can be used with state and functional components.	The state can be used only with the state components/class component (Before 16.0).
Props are read-only.	The state is both read and write.

Event handling

- Just like HTML DOM events, React can perform actions based on user events.
- React events are click, change, mouseover etc.
- React events are written in camelCase syntax. Ex: onClick instead of onclick.
- React event handlers are written inside curly braces. Ex: onClick={shoot} instead of onclick="shoot()".
- `<button onClick={shoot}>Take the Shot!</button>`

Event handling

- Common Events in React

Event Type	React Event	Triggered When...
Mouse	onClick, onDoubleClick, onMouseEnter	User interacts with mouse
Form	onSubmit, onChange, onInput	Form input actions
Keyboard	onKeyDown, onKeyUp, onKeyPress	Key is pressed/released
Focus/Blur	onFocus, onBlur	Element gains/loses focus

Event handling

Example:

Put the shoot function inside the Football component:

```
function Football() {
  const shoot = () => {
    alert("Great Shot!");
  }

  return (
    <button onClick={shoot}>Take the shot!</button>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

Event handling

- Passing Arguments:
  - To pass an argument to an event handler, use an arrow function.
- Example: Send "Goal!" as a parameter to the shoot function, using arrow function:

```
function Football() {
  const shoot = (a) => {
    alert(a);
  }

  return (
    <button onClick={() => shoot("Goal!")}>Take the shot!</button>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

React Forms

- Just like in HTML, React uses forms to allow users to interact with the web page.
- Ex: Add a form that allows users to enter their name:

```
function MyForm() {
  return (
    <form>
      <label>Enter your name:
        <input type="text" />
      </label>
    </form>
  )
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

React Forms Handling

- Handling forms means how you handle the data when it changes value or gets submitted.
- In HTML, form data is usually handled by the DOM, but in React, form data is usually handled by the components.
- When the data is handled by the components, all the data is stored in the component state. We can control changes by adding event handlers in the onChange attribute.
- We can use the useState Hook to keep track of each inputs value and provide a "single source of truth" for the entire application.

React Forms Handling

- Example: Use the useState Hook to manage the input and Add a submit button and an event handler in the onSubmit attribute:

```
import { useState } from 'react';
import ReactDOM from 'react-dom/client';

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

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`The name you entered was: ${name}`)
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>Enter your name:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
      <input type="submit" />
    </form>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

React Hooks

- *React Hooks* are a new addition in React 16.8. They let you use state and other React features without writing a class.
- Hooks are functions that allow the programmer to use state and other React features without writing a class.
- Hooks allow you to “hook into” React state and lifecycle features from functional components.
- Prior to Hooks, stateful logic in React components was primarily encapsulated in class components using the setState method.
- Hooks provide a more functional approach to state management and enable the use of lifecycle methods, context, and other React features in functional components.
- React Hooks can’t be used inside of class components.

Advantages of React Hooks

- **Simplified Logic:** Hooks eliminate the need for class components, reducing boilerplate code and making components easier to understand and maintain.
- **Reusability:** With Hooks, you can extract stateful logic into custom hooks and reuse it across multiple components, promoting code reuse and modularity.
- **Improved Performance:** Hooks optimize the rendering process by allowing React to memoize the state and only re-render components when necessary.
- **Better Testing:** Functional components with Hooks are easier to test compared to class components, as they are purely based on input and output.

React Hooks

- **Rules of React Hooks**
  - Hooks should be called only at the top level.
  - Don’t call hooks conditionally and inside a loop.
  - Hooks should be called only in a functional component but not through regular JavaScript functions.
- **Types of Hooks**
  - useState
  - useEffect
  - useReducer
  - useLayoutEffect
  - useContext
  - useCallback
  - useMemo
  - CustomHooks

useState React Hook

- It manages state in functional components by providing a state variable and a function to update it, enabling dynamic UI updates.

Syntax :

const [stateVariable, setStateFunction] = useState(initialStateValue)

Implementation of useState Hook

```
import React from 'react';
import {useState} from 'react';
export default function Incrementor()
{
    const [count, setCount]=useState(0);
    const increment=()=>{
        setCount(count+1);
    }
    return (
        <>
        <h1>{count}</h1>
        <button onClick={increment}>increment</button>
        </>
    )
}
```

useEffect React Hook

- It handles side effects like data fetching, subscriptions, or DOM manipulation in functional components after rendering.
- Syntax:

```
useEffect(() => {
        // Effect code
        return () => {
                // Cleanup code
                };
        }, [dependencies]);
```

useReducer React Hook

- useReducer is a Hook in React used for state management.
- It accepts a reducer function and an initial state, returning the current state and a dispatch function.
- The dispatch function is used to trigger state updates by passing an action object to the reducer.
- This pattern is especially useful for managing complex state logic and interactions in functional components.

Syntax:

```
const [state, dispatch] = useReducer(reducer, initialState);
```

useLayoutEffect React Hook

- useLayoutEffect is a Hook in React similar to useEffect, but it synchronously runs after all DOM mutations. It’s useful for operations that need to be performed after the browser has finished painting but before the user sees the updates.
- It is recommended to use “useEffect” over “useLayoutEffect” whenever possible, because “useLayoutEffect” effects the performance of the application.

useContext and useCallback React Hooks

- useContext simplifies data sharing by allowing components to access context values without manual prop drilling. It enables passing data or state through the component tree effortlessly.

Syntax:

```
const value = useContext(MyContext);useCallback :
```

- useCallback memoizes callback functions, preventing unnecessary re-renders of child components when the callback reference remains unchanged. It optimizes performance by avoiding the recreation of callbacks on each render.

Syntax:

```
const memoizedCallback = useCallback(() => {
// Callback logic
}, [dependencies]);
```

useMemo and Custom React Hooks

- useMemo memoizes function values, preventing unnecessary re-renders due to changes in other state variables. It's used to optimize performance by memoizing expensive calculations or derived values.

Syntax:

```
const memoizedValue = useMemo(() => {
// Value computation logic
}, [dependencies]);Custom Hooks
```

- Custom Hooks in React allows you to create your own hook and helps you to reuse that hook's functionality across various functional components. A custom hook can be created by naming a JavaScript function with the prefix "use".

Syntax :

```
function useCustomHook() {
//code to be executed
}
```

Router

- React-Router is a popular React library that is heavily used for client-side routing and offers single-page routing. It provides various Component APIs (like Route, Link, Switch, etc.) that you can use in your React application to render different components based on the URL pathnames on a single page.
- There are two types of router objects.
  1. BrowserRouter
  2. HashRouter
- If we want to handle the dynamic request, then use BrowserRouter and if we want to serve the static request then use HashRouter.
- These are 4 React Router Hooks that can be used in React applications:
  - useHistory
  - useParams
  - useLocation
  - useRouteMatch

App.js

```
1  import React, { Component } from 'react';
2  import { BrowserRouter as Router, Switch, Route, Link } from 'react-router-dom';
3  import Home from './components/Home';
4  import About from './components/About';
5  import Contact from './components/Contact';
6
7  class App extends Component {
8    render() {
9      return (
10     <Router>
11       <div>
12         <h2>Welcome to React Router Tutorial</h2>
13         <nav className="navbar navbar-expand-lg navbar-light bg-light">
14         <ul className="navbar-nav mr-auto">
15           <li><Link to={'/'} className="nav-link"> Home </Link></li>
16           <li><Link to={'/contact'} className="nav-link">Contact</Link></li>
17           <li><Link to={'/about'} className="nav-link">About</Link></li>
18         </ul>
19         </nav>
20         <hr />
21         <Switch>
22             <Route exact path='/' component={Home} />
23             <Route path='/contact' component={Contact} />
24             <Route path='/about' component={About} />
25         </Switch>
26       </div>
27     </Router>
28   );
29   }
30 }
31
32 export default App;
```

Main.js

```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import App from './App';
4
5  ReactDOM.render(<App />, document.getElementById('App'));
```

Home.js

```
1  import React, { Component } from 'react';
2
3  class Home extends Component {
4    render() {
5      return (
6        <div>
7          <h2>Home</h2>
8        </div>
9      );
10   }
11 }
12
13 export default Home;
```

About.js

```
1  import React, { Component } from 'react';
2
3  class About extends Component {
4    render() {
5      return (
6        <div>
7          <h2>About</h2>
8        </div>
9      );
10   }
11 }
12
13 export default About;
```

Contact.js

```
1  import React, { Component } from 'react';
2
3  class Contact extends Component {
4    render() {
5      return (
6        <div>
7          <h2>Contact</h2>
8        </div>
9      );
10   }
11 }
12
13 export default Contact;
```