Create React App:

* npx create-react-app my-app
* cd my-app
* npm start

Rendering to DOM:

* Typically, we render one component to DOM
* Rest all components are embedded in to root components

// index.js

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

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

// App.js

import React from 'react';

class App extends React.Component {

render() {

return (

<div className="app">

<h1>App Component</h1>

</div>

);

// return [<p>Hello World</p>, <p>I am user</p>]; ALLOWED

}

}

export default App;

* React calls render() method to show HTML on DOM

What is JSX?

* JSX is a syntactical sugar helps us to write JavaScript and HTML together
* Files you see having (.js) or (.jsx) as extension
* Alternate way of using JSX

return React.createElement('div', {className: 'app'}, React.createElement('h1', null, 'App Component'));

JSX Restrictions:

* CSS “class” can be written as “className”
* JSX must have one root element, all elements must be wrapped inside on <div></div>

Components & JSX Cheat Sheet:

* Components are the core building block of React apps. Actually, React really is just a library for creating components in its core.
* A typical React app therefore could be depicted as a component tree - having one root component ("App") and then a potentially infinite amount of nested child components.
* Each component needs to return/ render some JSX code - it defines which HTML code React should render to the real DOM in the end.
* JSX is NOT HTML but it looks a lot like it. Differences can be seen when looking closely though (for example className in JSX vs class in "normal HTML"). JSX is just syntactic sugar for JavaScript, allowing you to write HTMLish code instead of nested React.createElement(...) calls.
* When creating components, you have the choice between two different ways:
  + Functional components (also referred to as "presentational", "dumb" or "stateless" components - more about this later in the course)
  + class-based components (also referred to as "containers", "smart" or "stateful" components)

Class Components:

* State can be achieved in Class based components

// App.js

import React from 'react';

class App extends React.Component {

render() {

return (

<div className="app">

<h1>App Component</h1>

</div>

);

}

}

export default App;

Functional Components:

// App.js

import React from 'react';

import User from './User/User'

class App extends React.Component {

render() {

return (

<div>

<h1>App Component</h1>

<User />

</div>

);

}

}

export default App;

// User.js

import React from 'react';

const user = (props) => {

return (

<div>

<p>I am User</p>

</div>

);

}

export default user;

Props & State:

* props and state are CORE concepts of React. Actually, only changes in props and/ or state trigger React to re-render your components and potentially update the DOM in the browser

Working with Props:

* To output a dynamic value in HTML use {VALUE}
* It allows you to pass data from a parent (wrapping) component to a child (embedded) component.
* Props are used to pass data in to a component, where we can catch data an can use it

// Parent Component

<User name="Sagar" age="29"/>

// User.js

import React from 'react';

const user = (props) => {

return (

<div>

<p>I am {props.name}, My age is {props.age} </p>

</div>

);

}

export default user;

* Note: While using in Class Components use as follow 🡪 this.props.name
* To access data passed in between tags of a component can be accessed are as follow

<User name="Sagar" age="29">Hello User</User>

<p>I am {props.name}, My age is {props.age}</p>

<p>{props.children}</p>

* Instead of simple text you can pass complex HTML as well

Understanding & Using State:

* Whilst props allow you to pass data down the component tree (and hence trigger an UI update), state is used to change the component, well, state from within. Changes to state also trigger an UI update.
* We can use “State” only in “Class Components” Below 16.8 V
* In 16.8V by introducing “HOOKS” we can use state in “Functional Components” also
* **Note:** If state changes it leads react to re render DOM
* state simply is a property of the component class, you have to call it state though - the name is not optional. You can then access it via this.state in your class JSX code (which you return in the required render() method).
* Whenever state changes (taught over the next lectures), the component will re-render and reflect the new state. The difference to props is, that this happens within one and the same component - you don't receive new data (props ) from outside!

// App.js

import React from 'react';

import User from './User/User'

class App extends React.Component {

state = {

name: "Sagar",

age: 29

}

render() {

return (

<div>

<h1>App Component</h1>

<User name={this.state.name} age={this.state.age}>Hello User</User>

</div>

);

}

}

export default App;

Manipulating State:

* You should not directly update the “state”, react provides a method setState() to update
* If you update value of a key in your state, setState() wont disturb other Key Values

class App extends React.Component {

state = {

name: "Sagar",

age: 29

}

updateUserHandler = () =>{

this.setState({

age: 30

});

}

render() {

return (

<div>

<button onClick={this.updateUserHandler}>Update User</button>

<User name={this.state.name} age={this.state.age}/>

</div>

);

}

}

Handling Events with Methods:

* In JSX event handlers are “CamelCase” 🡪 onClick=”Method\_Name”
* IMP: Don’t add “()” to a method while calling from any event handler, if you add it will execute immediately.
* If you want to use “()” then 🡪 onClick={ () => this.timeLogingHandler()}. Advantages are you can pass params in to functions (Use only for small data refresh)
* For large data refresh 🡪 onClick={this.loggingHandler.bind(this, ‘VALUE’)}
* Use only “Arrow Functions” for Event Handlers

// App.js

import React from 'react';

class App extends React.Component {

timeLoggongHandler = () =>{

console.log(new Date())

}

render() {

return (

<div>

<button onClick={this.timeLoggongHandler}>Log Time</button>

</div>

);

}

}

export default App;

* Note: Refer following URL to get all events by react 🡪 <https://reactjs.org/docs/events.html#supported-events>

Hooks to Manipulate Function State:

* Import useState from react
* In useState you pass your initial state
* useState returns array with exactly two elements always two elements
* 1st element is current state
* 2nd element is a function used to update the state,
* **Note:** Here while updating state (2nd element) it does not merge value to a key it will replace old state with new state value you pass in to it.
* Using useState() multiple time for each and every is a good practice

// App.js

import React, { useState } from 'react';

import User from './User/User'

const App = (props) => {

const [getState, setState] = useState({

user: {

name: "Sagar",

age: 29

}

});

const updateUserHandler = () => {

setState({

user: {

name: "new",

age: 29

}

});

}

return (

<div>

<button onClick={updateUserHandler}>Update User</button>

<User name={getState.user.name} age={getState.user.age}/>

</div>

);

}

export default App;

Stateless & Stateful:

* The component which maintains state is called Stateful component
* Which doesn’t is called Stateless
* Note: Use as many functional stateless components as possible
* Don’t use state in every component

Passing Method Ref Between Components:

* We can pass parent method as props to child component

// App.js

class App extends React.Component {

state = {

name: "Sagar",

age: 29

}

updateUserHandler = () =>{

this.setState({

age: 30

});

}

render() {

return (

<div>

<button onClick={this.updateUserHandler}>Update User</button>

<User

name={this.state.name}

age={this.state.age}

updateUserProp={this.updateUserHandler}>Click Me</User>

</div>

);

}

}

// User.js

const User = (props) =>{

return (

<div>

<p>I am {props.name}, My age is {props.age}</p>

<p onClick={props.updateUserProp}>{props.children}</p>

</div>

);

}

Skip DIV:

* If you like to skip div wrapper tag use an in-built tag given by react
* <React.Fragment> HTML </React.Fragment>

Useful Resources & Links:

* create-react-app: <https://github.com/facebookincubator/create-react-app>
* Introducing JSX: <https://reactjs.org/docs/introducing-jsx.html>
* Rendering Elements: <https://reactjs.org/docs/rendering-elements.html>
* Components & Props: <https://reactjs.org/docs/components-and-props.html>
* Listenable Events: <https://reactjs.org/docs/events.html>