React.js

Why do we need react when we already have JS?

Ans. Because writing applications is much more easier with React.js rather than with JS. JS follows imperative programming model where we define the steps of how to achieve something, whereas React follows Declarative model where we define the end goals and react figures out the best way to achieve it.

**Components**

Components are just custom attributes which you can define in JSX code (which looks like HTML).

Your components must start with upper case.

React handles the uppercase and lower case components differently. Upper case are called as functions.

**JSX**

We use JSX to write React components. It allows to define target HTML code inside of components/ JS code.

JSX code leads to a tree-like code structure that "tells" React how the different components are related and how the UI should look like.

React then executes appropriate commands to manipulate the real DOM to reflect that target structure / code.

function Header() {

return (

<header>

<h1>React Essentials</h1>

<p>This is a react tutorial</p>

</header>

);

}

function App() {

return (

<div>

<Header />

<main>

<h2>Get started!</h2>

</main>

</div>

);

}

**Proper way to set Image source in React**

Consider this way first:

function Test() {

return (

<img src="src/assets/abc.png" />

);

}

The way src is set here is not optimal. It may get lost during the optimization process during build.

This is how to do it:

import reactImg from './assets/abc.png';

function Test() {

return (

<img src={reactImg} />

);

}

- Here, reactImg is a javascript variable pointing to the path of the image.

But it will include an automatically generated path, that will also work when you deploy react app to a server.

- Note that there are no quotes around src attribute value.

**Props**

React allows you to pass data to components via a concept called props.

Let's say when you call one component in another, you also want to pass some data.

You can define custom attributes to your custom components, which the child component then receive.

For eg:

function CoreConcept() {

return (

<li>

<img src='...' alt='...' />

<h3>Title</h3>

<p>Description</p>

</li>

);

}

function App() {

return (

<div>

<CoreConcept />

<CoreConcept />

<CoreConcept />

);

}

Now, let's say we want to display different titles and description.

We can pass these using props. The custom attributes we will be passing is basically props only.

import componentImg from './assets/component-img.png'

function CoreConcept(props) {

return (

<li>

<img src={props.image} alt={props.title} />

<h3>{props.title}</h3>

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

</li>

);

}

function App() {

return (

<div>

<CoreConcept title="Test Value" description="This is a test value" image={componentImg} />

<CoreConcept />

<CoreConcept />

);

}

In normal JS function, we define same number of arguments in a method which are passed.

But in JSX, all the parameters are received in one parameter only, which we typically call as props. But we can give it a custom name.

props parameter will be set by React as it will execute the function.

Remember, we are not calling these component function ourselves, instead we are using them as HTML elements and React is calling them.

So, React converts all the values we pass to an object, which will be received by the child component.

**Get Rid of props keyword**

You can destructure the props parameter received in CoreConcept component, as long as you use the same names.

You then don't need to use props keyword everytime.

This is how the CoreConcept would look like:

function CoreConcept({image, description, title}) {

return (

<li>

<img src={image} alt={title} />

<h3>{title}</h3>

<p>{description}</p>

</li>

);

}

**Children Props**

You have a choice whether to send props as attributes or children. Children props are what is set automatically by React.

The children prop is a special prop that's automatically provided to every component function. It contains the wrapped content as a value.

export default function App() {

return (

<section id="examples">

<h2>Examples</h2>

<menu>

<TabButton>Components</TabButton>

<TabButton>JS</TabButton>

</menu>

</section>

)

}

export default function TabButton({children}) {

return <li><button>{children}</button></li>

}

This method of having one component in another is called as component composition.

Note - while destructuring, you can only use "children" name in the method parameter.

All other props can be named whatever we want (but it should match what we are passing in).

**Reacting to Events/Event Handlers**

React provides us with Prop Functions which usually start with "on" keyword. Also called as event handlers.

They determine what action should happen when an event is fired. Could be a button click or change in text.

Event propagation in react follows the same two phase process as native DOM events: the capturing phase and the bubbling phase. (In order to understand about events, refer to another note on DOM and Events).

**onClick**

There are several ways you can handle this event in differnet languages like HTML, JS, jQuery and React.

Although there are syntactical differences, the underlying concept of handling click events remains consistent.

This is the the TabButton component would look like with an event handler:

export default function TabButton({children}) {

function handleClick() {

alert("clicked");

}

return (

<li><button onClick={handleClick}>Component</button></li>

)

}

Note - You have to pass a function pointer within curly braces.

**Synthetic Events in React**

A synthetic event is a cross-browser wrapper around the browser's native event.

React implements synthetic events that brings consistency and high performance to React apps and interfaces.