Namaste React

EPISODE-1

creating Hello World using JS

```
<div id="root"> </div>
<script>
  const heading = document.createElement("h1");
  heading.innerHTML = "Hello World!!";
  const root = document.getElementById("root");
  root.appendChild(heading);
</script>
```

creating Hello world program using React

→ In above code, document, createElement, innerHTML, and more are super powers which browser already having in it.

→ Browser have JS Engine in it and it execute this.

→ But Browser do not understand React code.

→ Our project must be configured to use react.

→ inject React into our project by search cdn react.

cdn - content delivery network it is a website where these React is hoisted and we are just pulling this into our project. by CDN Links.

→ CDN is a place where React Library is hoisted.

→ when the CDN links are injected into our project or paste it, Browser can understand React.

crossorigin: crossorigin attribute in the script tag enables crossorigin resource sharing(CROS) for loading external JS files. From different origin that the hosting web page. This allows the script such as making HTTP requests or accessing data.

what representing the two files or links of react?

There are two files. First file is react.development.js.

-> This is the core of React Frame Algorithm.

-> second one is react-DOM. It is react library used for DOM operations. or) This is the react which we need to modify the DOM.

i.e., react-dom.development.js.

-> It is like a bridge b/w the react and Browsers. connected to a DOM.

// program

```
<div id="root"> </div>

<script>
    const heading = React.createElement("h1", {}, "hello world from React!!");

    const root = ReactDOM.createRoot(document.getElementById("root"));

    root.render(heading);
</script>
```

creating object now.

O/P: Hello world from React!!

→ React comes with a philosophy of writing or manipulating DOM using javascript (or) using React.

Why we use {} object in above program?

* This will use to give Attributes in our created element. , { id: "heading" },
* For class, id's, our own creation Attributes.

→ In below, heading when we print heading, it will give us the object i.e., Reacts element

→ React element is nothing but normal JS object.

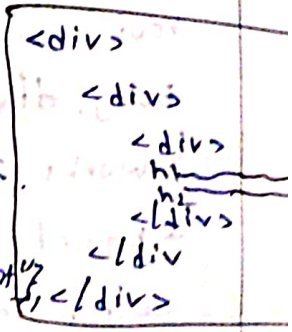
→ Props are the children and attributes that we are passing.

"h1", { id:"heading", xyz:"abc" }, ← Attributes
"Hello word From React"); ← children

* Render :- render function job is basically to take this object, create that h1 tag which browser understands and put it inside that root element or put it upon the DOM.

How do we create Nested Elements using React?

→ For creating this we need to pass the 3rd Agument of react creating object.




```
const parent = React.createElement("div", { id: "parent" },
  React.createElement("div", { id: "child" }, [
    React.createElement("h1", { id: "heading" }, "I am h1"),
    React.createElement("p", { id: "pg" }, "I am p tag"),
  ])
);
```

Will order in HTML Really Matter?

Order of files are always in sequence. Always you need to keep React links before your script.js file.

1. React Links
2. <script src= script.js

→ when you do a root.render() it will replaced everything inside root with whatever I passed from React. React has taken control over the div now.

* HTML Executes line by Line.
* other portions of React works same. Just replaced whatever id or class matches.

A key difference between the two is inversion of control. When using a library, the control remains with the developer who tells the application when to call Library functions.

When using a Framework, the control is reversed which means that the framework tells the developer where code needs to be provided and calls as it requires.