React-1: Introduction to React: JSX, Props, lists

title: Agenda of the lecture

Prerequistes Required before Starting with React

1. Set up
2. VS Code (recommended)
3. Chrome Browser (recommended)
4. Proper revision of HTML/CSS and Javascript is a must

React has become the most capable tool in today's era in frontend Development for creating beautiful looking smooth and fast UI hence enhancing the UX as well

React has been Built and maintained by Facebook and Big Tech giants like Netflix , Uber and many more are using it in their Org

https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/

Agenda of The Lecture

1. Build your first Hello World program using just HTML and JS
2. Ingesting React with CDN to build the same Hello World
3. Create an Element

4. Create nested React Elements
5. Use root.render
6. Functional Component
7. JSX -> Javascript and XML
8. How JSX is read (Babel Transpiler)
9. Advantages of JSX

Build your first Hello World program using just HTML and JS

Before Moving to react ,Let's just do a simple exercise of Creating a heading element with the Text 'Hello from JS' We will be using plain HTML and JS for now

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Document</title>
 </head>

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

 <script>
   const headingElement = document.createElement("h1");
   headingElement.innerHTML = "Hello from JS";

   const root = document.getElementById("root");
   root.appendChild(headingElement);
 </script>
</html>
```

The JavaScript code in the provided HTML document does the following:

1. Create an HTML Element: It creates a new <h1> element and assigns it to the variable headingElement.
2. Set Element Content: It sets the inner HTML of the headingElement to the text "Hello from JS". This means the <h1> element will display "Hello from JS" as its content when rendered on the webpage.
3. Access an Existing Element: It retrieves the element with the ID "root" from the HTML document and assigns it to the variable root.
4. Append Element to DOM: It appends the newly created headingElement as a child to the root element. This effectively inserts the <h1> element within the element in the HTML structure.
5. As a result, when the HTML document is loaded in a browser, it will display a heading that says "Hello from JS" inside the element with the ID "root".

title: Ingesting react with CDN

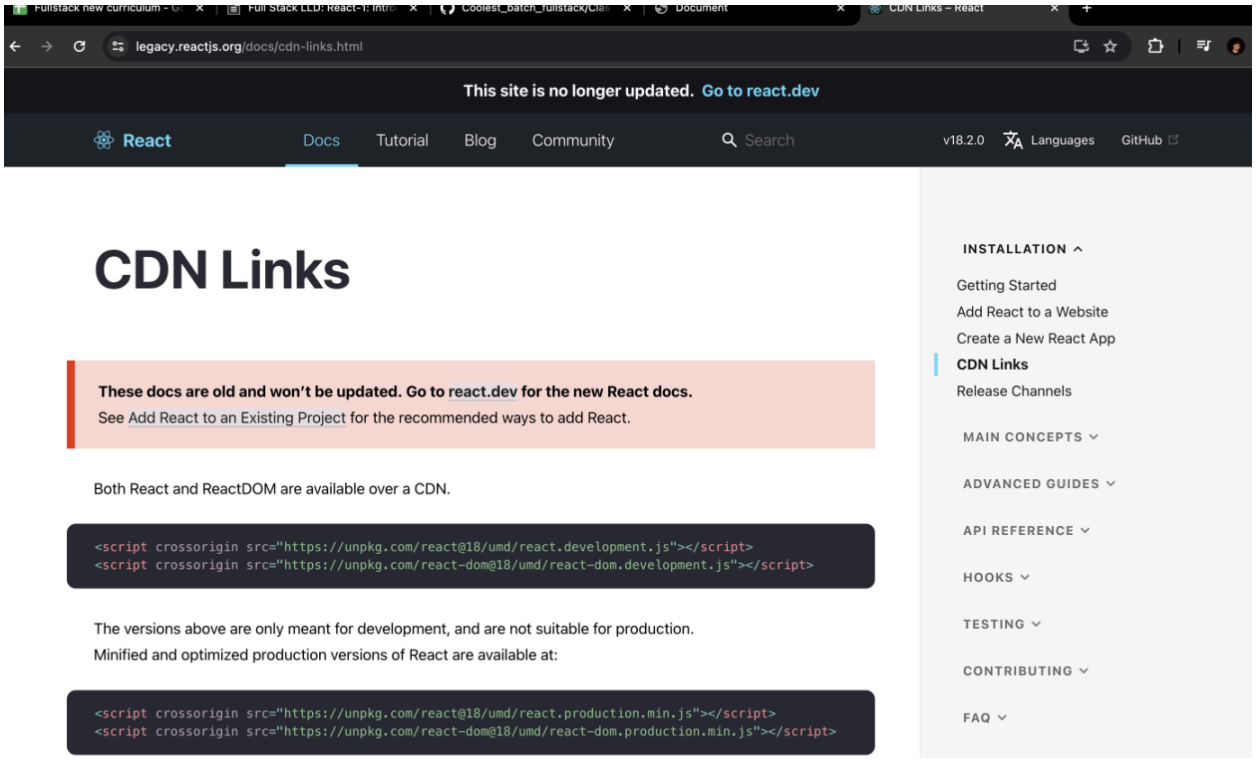
1. Now the First thing that we will do is Get REACT in our Codebase , Now there are various ways of setting up a React app,
2. We can use CDN Links

3. We can use Build tools like (Webpack , Parcel , Vite) (this we will cover Later)

Let's see How to ingest React in our codebase using the CDN

Follow this link - https://legacy.reactjs.org/docs/cdn-links.html

You wil see Something like this when you follow the Link



Now create a Fresh Html file or you can continue in the one you wrote your Hello World program Only!

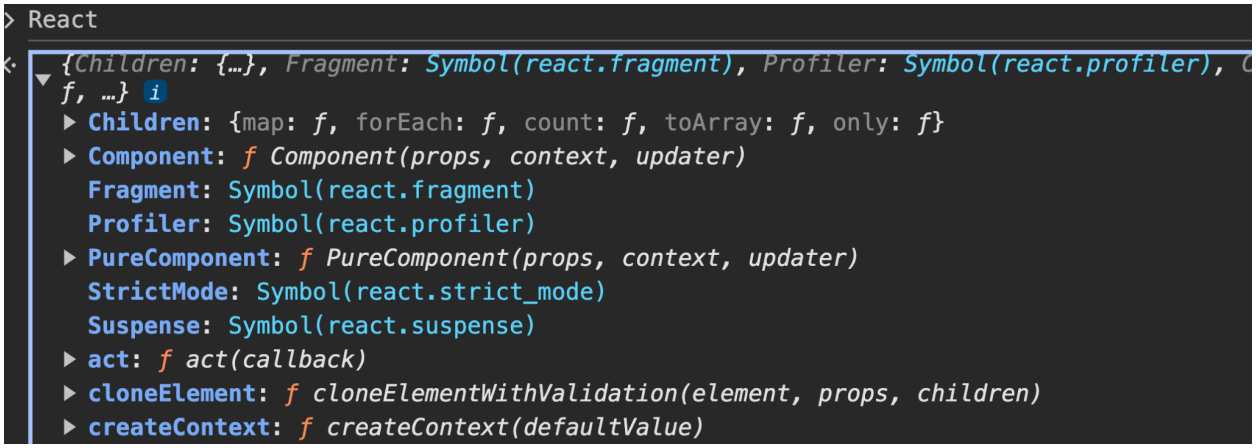
What you have to do is get these CDN links ingested inside the html files like this

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Document</title>
 </head>

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

 <script
   crossorigin
   src="https://unpkg.com/react@18/umd/react.development.js"
 ></script>
 <script
   crossorigin
   src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
 ></script>
</html>
```

Now as You have ingested them , all the Powers of React are with you

Go to the Console and Type React

You will see now React is available to you and you can even open this and see what React Offers , In the future all the methods and properties that you see we will be using them



You can even Open this CDN link in the Browser , if you just copy the CDN link and paste it in the Browser you will see That React is nothing but plain Written Javascript code , This is the Source Code of React! and all React Stuff that we are going to use is coming from Here Only!

ReactDOM

1. Similarly , We have got another CDN link ingested that is for react-dom
2. So the idea is that React and ReactDOM are used together because each library serves a specific purpose in building web applications:
3. React: is a library for creating components and managing Them and It defines the structure and behavior of the components.
4. ReactDOM: Handles rendering of the React components to the DOM (Document Object Model), making them visible on the web browser. It also manages updates to the DOM when React makes some changes.

5. Essentially, while React creates and manages the components and their internal logic, ReactDOM takes care of rendering these components in the web environment.
6. This separation of concerns allows React to be used not just for web applications but also for other platforms like mobile (through React Native), while ReactDOM specifically deals with web-specific interaction with the DOM.
7. You can even check ReactDOM in the same way and it will show you whatever you will use to handle DOM operations with React

title: Creating Hello World with React

We will use the React and ReactDOM libraries and achieve the Result

HTML Template with React CDN (Without JSX or Components)

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>React App</title>
   <!-- Adding React and ReactDOM from CDN -->
   <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
   <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
 </head>
 <body>
   <!-- Very Important to create a root as all the things you will build will go
inside the root -->
```

```
  <div id="root"></div>
  <script>
    // Get the root element where the React app will be mounted
    const rootElement = document.getElementById("root");

    // Create an element using React.createElement
    // React.createElement takes at least three arguments:
    // 1. The tag name of the HTML element to create
    // 2. The properties of the element (or null if none) Adding null here for now we
will discuss props later
    // 3. The children of this element (or text content)
    const headingElement = React.createElement(
      "h1",
      null,
      "Hello from React"
    );

    // Render the element to the DOM using ReactDOM
    // ReactDOM.render takes two arguments:
    // 1. The React element to render
    // 2. The DOM element where the React element should be mounted
    ReactDOM.render(headingElement, rootElement);
  </script>
 </body>
</html>
```

You can see the Definitions of these methos here

```
> React.createElement
< ƒ createElementWithValidation(type, props, children) {
      var validType = isValidElementType(type); // We warn in this
  creation to
      // succeed and there …
```

Step-by-Step Explanation

1. HTML Setup:

    a. The HTML structure includes loading React and ReactDOM from CDN. These are necessary for using React's APIs without local installations or build tools.
    b. The acts as the mounting point for the React application.
2. Creating the Element:
    a. React.createElement is used to create a React element. This method creates an element which can be rendered into the DOM. Here, it is used to create an <h1> element with the text "Hello from JS".
    b. you manually specify the tag name, properties (if any), and children directly.
3. Rendering the Element:
    a. ReactDOM.render is used to render the React element into the DOM. It takes two arguments: the React element and the DOM container where the element should be mounted.
    b. This call places the <h1> element within the in the HTML.
4. This is How to use React and ReactDOM code to achieve the desired result

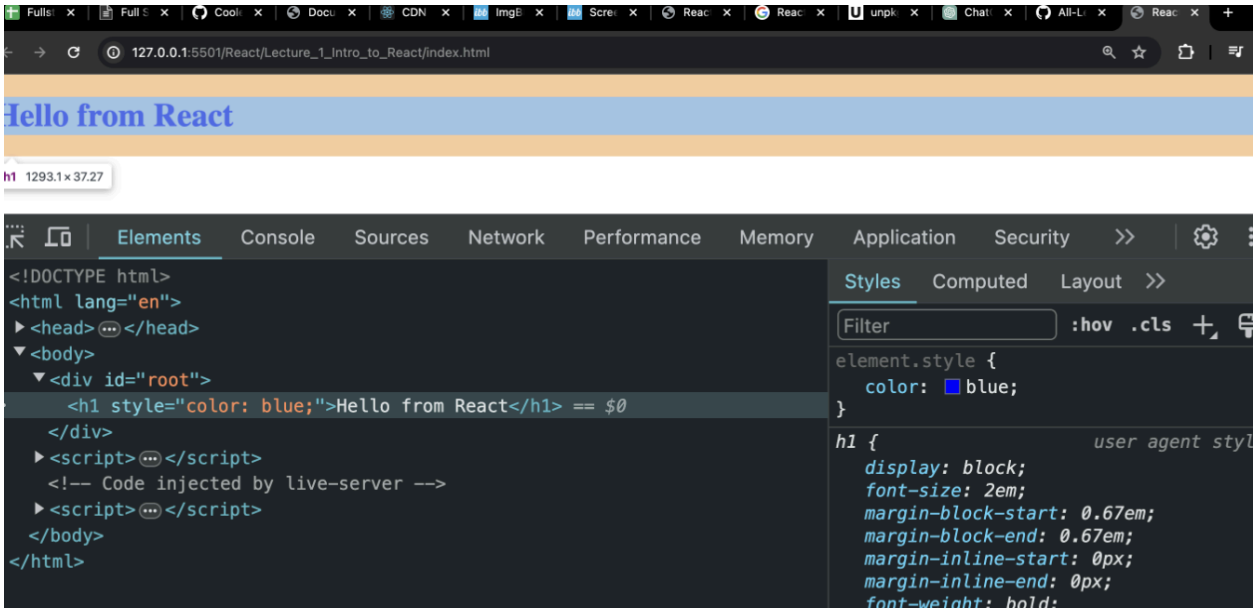
title: Props

1. Analogy: Props as Function Arguments

    a. Imagine you have a function in JavaScript that takes parameters. You can think of props in React as the parameters to a component.
    b. Just as parameters allow functions to accept different values and behave accordingly, props allow components to receive values from their parent component and use them to determine what they render or how they behave.
2. Props (short for “properties”) are a way of passing data from parent to child components in React. They are read-only, which means the child component cannot modify the props it receives.
3. Now if you recall at One Place we have given null in the code , as you pass any attribute in html you cannot directly do that over here in this form of code , so react Provides something known as props , You can make use of props to pass any attribute that you want
4. To add a prop to the createElement method in your existing code, let's introduce a style property that changes the text color of the <h1> element. Properties in React are similar to attributes in HTML but are more powerful due to their dynamic nature, You will witness theri power as we will move forward!
5. Update React.createElement

```
const headingElement = React.createElement(
       "h1",
       {
            style: { color: 'blue' }
        },
       "Hello from React"
    );
```

6. Adding the style Property:
    a. Inside the React.createElement call, the second argument is used for properties (props). Here, we've added a style prop.
    b. The style prop expects an object where the keys are camelCased CSS property names and the values are the corresponding CSS values. In this case, { color: 'blue' } changes the text color of the <h1> element to blue.
    c. Effect on Rendering:
        i. When the element is rendered to the DOM, the style property applies the CSS directly to the <h1> element, affecting its appearance on the webpage.
7. The Output now has a style attribute which sets the color to Blue



title: JSX and Components

Using React.createElement directly makes the code verbose and harder to read, especially as the application grows.

While it's useful to understand how React works under the hood with React.createElement, for practical application development

Adopting JSX and a component-based architecture is highly recommended.

Let's see What is this JSX and React's Component Based Architechture

Component Characteristics:

1. Independent: Each component is independent, encapsulated with its own functionality and possibly its own state, much like rooms in a house are designed for specific activities.
2. Reusable: Components are designed to be reusable across different parts of an application or even between different applications, much like standardized parts like windows and doors can be used in various buildings.
3. Manageable: Components manage their own state and behavior, which helps in keeping the overall application architecture clean and maintainable.

4. So , In React, a component is independent piece of UI. It's like a JavaScript function that returns elements describing what should appear on the screen. Components can be simple and display static content (like a header or button), or complex and maintain internal state and lifecycle events (like a form or interactive dashboard)
5. They can be classified mainly into two types:
    a. Functional Components: These are simple JavaScript functions that return JSX.
    b. Class Components: These are ES6 classes that extend React.Component and can include more complex logic, lifecycle methods, and state management.
        i. Class Components are now nearly deprecated beacuse of their complexity we will be using Functional Components

What is JSX?

1. JSX is a syntax extension for JavaScript that looks similar to XML or HTML. JSX is used with React to describe what the UI should look like. It lets you write HTML-like code in your JavaScript files.
2. So in place of those verbose React.createElement code, we will write a much simpler way to create components

3. Under the hood, JSX is transformed into React.createElement calls by preprocessors like Babel. This makes it easier to write and understand the UI code.

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>React App</title>
   <!-- Adding React and ReactDOM from CDN -->
   <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
   <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
 </head>
 <body>
   <div id="root"></div>
    <script>
      // Functional Component using JSX
      function App() {
        return <h1>Hello from React</h1>;
      }

      ReactDOM.render(<App />, document.getElementById("root"));
    </script>
 </body>
</html>
```

4. Creating a Functional Component:
    a. function App() { ... }
    b. App is defined as a function, which makes it a functional component in React terms. In React, a functional component is simply a JavaScript function that returns React elements.

5. Components can be named using any valid JavaScript identifier but typically use PascalCase (where each word starts with a capital letter).
6. Returning JSX from the Component:
    a. return <h1>Hello from React</h1>;
    b. Inside the App function, JSX is returned. JSX allows you to write HTML-like syntax directly within JavaScript, which makes the structure of the component's UI clear and readable.
7. JSX Element: The <h1>Hello from React</h1> is a JSX element. JSX compiles into React.createElement() calls behind the scenes, but it's written like HTML for better readability and familiarity.
8. Rendering the Component:
    a. ReactDOM.render(<App />, document.getElementById('root'));
    b. This line of code is where React starts interacting with the DOM.
9. ReactDOM.render: This function is used to render a React element into the DOM in the supplied container and returns a reference to the component (or returns null for stateless components).
    a. First Argument <App />: Here, the App component is instantiated. The JSX tag <App /> translates to a React element representing the App component. This is similar to creating an instance of a class or calling a function.

b. Second Argument document.getElementById('root'): This specifies where the React application should be mounted in the DOM. In this case, it targets a DOM element with the ID root, typically a element in your HTML.
c. This method tells React to wipe out any existing content inside the and replace it with the JSX returned by the App component, which in this case, is <h1>Hello from React</h1>.

10. What is JSX Here?
    a. JSX: In this script, <h1>Hello from React</h1> and <App /> are examples of JSX. JSX lets you expressively layout the component's HTML structure within JavaScript with a syntax that resembles HTML. The JSX expression <h1>Hello from React</h1> represents an HTML h1 element with text content "Hello from React".
11. Now you will notice that this code will not run and will not provide the expected Output!
12. It Gives a Blank page
13. Your Browser can Only understand HTML , CSS and JS code , but if you see JSX is a differnent format altoghther , So Your Browser is not able to understand what this code means and hence not able to understand and render the output.

title: Adding Babel to parse JSX

To use JSX, we need Babel to convert JSX into React calls. Here's how you can include Babel:

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>React App</title>
   <!-- Adding React and ReactDOM from CDN -->
   <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
   <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
   <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
 </head>
 <body>
   <div id="root"></div>
   <script type="text/babel">
     // Functional Component using JSX
     function App() {
       return <h1>Hello from React</h1>;
     }

     ReactDOM.render(<App />, document.getElementById("root"));
   </script>
 </body>
</html>
```

How Babel Reads JSX

1. Babel is a JavaScript compiler that transforms syntax. With the React preset, Babel converts JSX into React.createElement calls.
2. For example, <h1>Hello from React</h1> in JSX is transformed by Babel into React.createElement("h1", null, "Hello from React").

3. This transformation makes the JSX syntax usable in browsers that only understand standard JavaScript.
4. So React’s team developed a user friendly intuitive syntax for development
5. Then they wrote a babel extension to convert jsx to JS
6. Babel’s plugin based architecture played a pivotal role in fast adoption of React

title: Advnatages of JSX

1. JSX looks like HTML: For anyone who has any experience with HTML, JSX feels familiar and easy to understand. It allows you to write HTML-like code directly within your JavaScript files, making it simpler to visualize and design the UI within the context of the JavaScript code that handles logic.
2. Clear layout: By using JSX, the structure of the interface you're building is much clearer. This makes it easier for developers, including newcomers, to see the hierarchy and layout of the application's UI, making it more straightforward to understand and modify.
3. Visual component structure: With JSX, components’ structure is visually evident in the code, much like the layout of elements in a standard HTML page. This helps in visualizing parent-child relationships in the component hierarchy, making component-based development more intuitive.

JSX is not just a syntactic sugar but a powerful tool that integrates seamlessly with the JavaScript language and React's component philosophy, offering both aesthetic and functional advantages.

Its HTML-like syntax coupled with JavaScript's power enhances productivity and maintainability, making it particularly beneficial for beginners in easing the learning curve of React development.

So Now you have Idea about React , React DOM , Component , JSX and Babel

But this setup is ideal for learning purposes. For production or larger projects, more sophisticated build setups is needed

So in the Next class , We will be talking about Build Tools and will see how to set up react with Build Tools.