Lec 1

Difference between library and framework?

1. Library:
   - A library is a collection of pre-written code that provides specific functionalities or solves particular problems.
   - Libraries do not enforce a specific structure or dictate the overall architecture of an application.
   - Developers have more control over the application flow and architecture, as they selectively use the library components based on their needs.
   - Examples: React

2. Framework:
   - Frameworks usually have a well-defined structure, including rules for organising code, handling dependencies, and managing application flow.
   - It provides a structured environment for building applications, often enforcing specific architectural patterns or design principles.
   - Frameworks have control over the application flow and architecture.
   - Examples: Spring

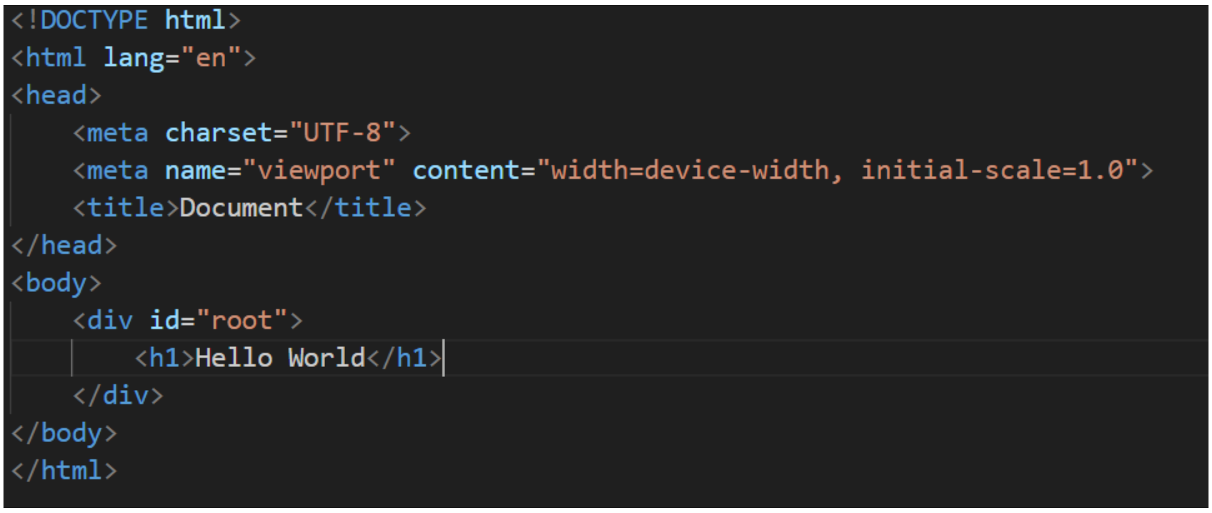
React is highly flexible and can be easily integrated into existing projects or combined with other libraries and frameworks. It can be used alongside libraries like Redux for state management, React Router for routing.

Spring, on the other hand, is regarded as a framework because it provides a comprehensive set of tools and features. Spring has various libraries and components for dependency injection, database access, web development, security, and more.

What is emmet?

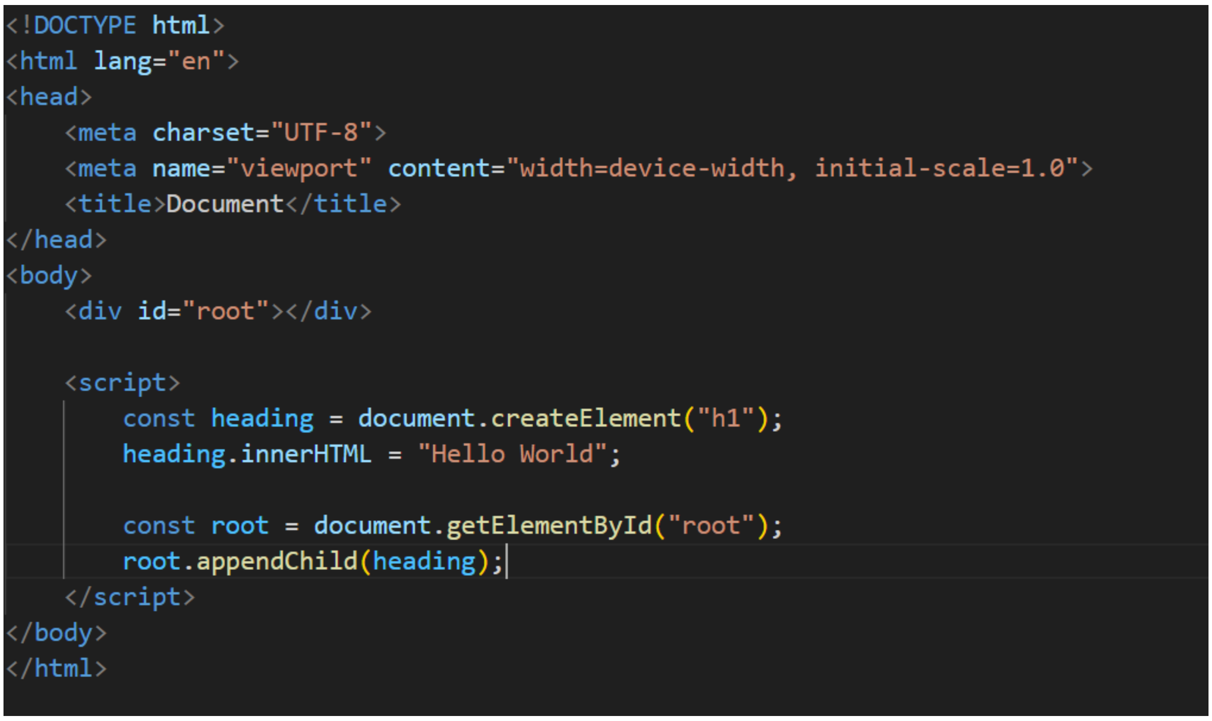
Emmet is a tool that allows developers to write and expand shorthand syntax for HTML, XML, and other languages.

To print something in html?



```html
<!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">
        <h1>Hello World</h1>
    </div>
</body>
</html>
```

To print the same thing in js?

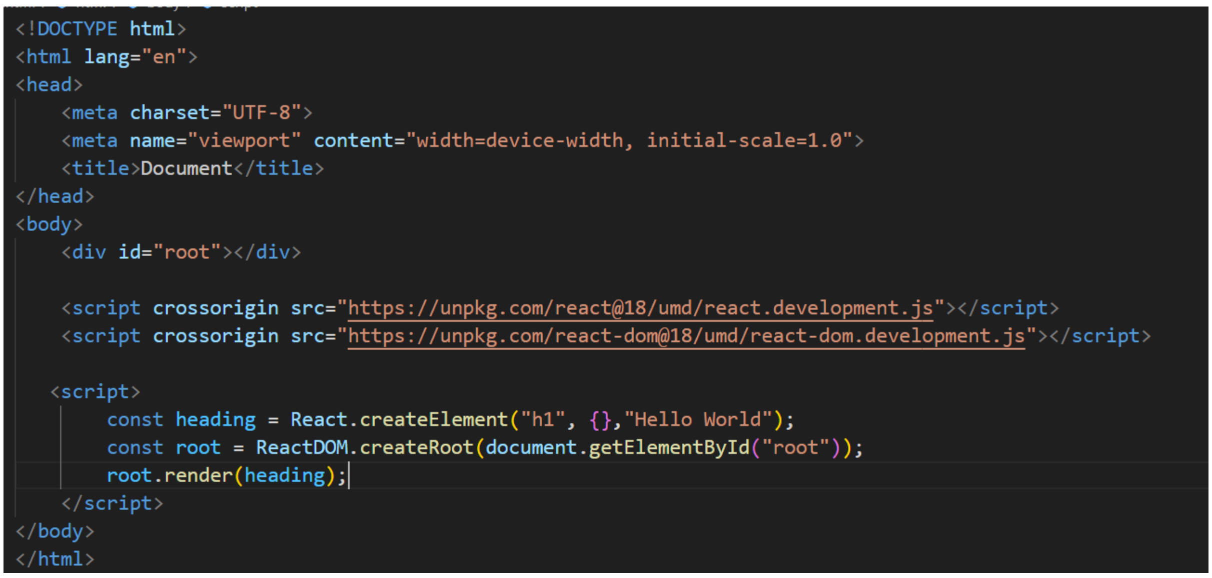


```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="root"></div>

    <script>
        const heading = document.createElement("h1");
        heading.innerHTML = "Hello World";

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

To print the same thing in React?



```html
<!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>

    <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>

   <script>
       const heading = React.createElement("h1", {},"Hello World");
       const root = ReactDOM.createRoot(document.getElementById("root"));
       root.render(heading);
    </script>
</body>
</html>
```

Putting some nodes in dom, removing it (like show hide modal) is the costly operation and react trying to optimise this using javascript.

React.createElement(tags, [props], [children]) :-

Tags :- a html tag like h1, div etc,
Props :- like class, id of that tag (can be an object or null, where null is same as an empty obj)
Children :- the content of the tag, like what we use in .appendChild()
It is a top-level API of ReactJS which is used to create an element like the document.createElement() in DOM.

Now, in React, we have to create a root such that we can tell the JS Engine that we want to run our React application in that place. To do this, ReactDOM contains APIs to render React components on the client(on the browser).

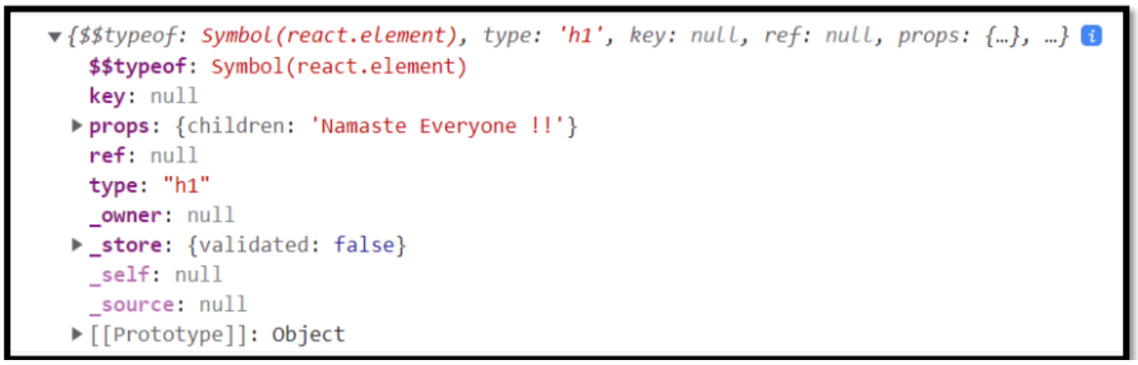
ReactDOM.createRoot(document.getElementById("root")) :-

This API is a Client-side API that lets you create a root to display React components inside a browser DOM mode. It returns an object with 2 functions/methods :- render and unmount

root.render(reactNode) :-

helps to display a piece of JSX("React node") or a React Element(created by react.createElement()) into the React root's browser DOM node.
It returns undefined.

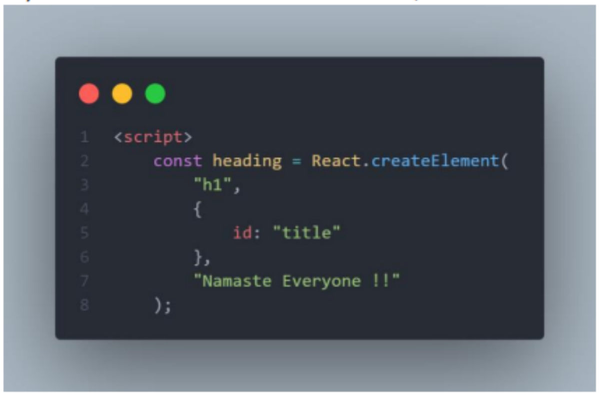
If you do, console.log(heading), it will return an object :-



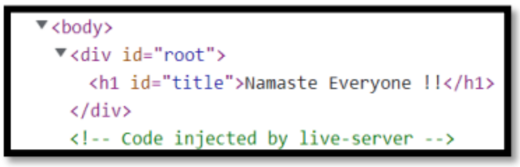
```
▼{$$typeof: Symbol(react.element), type: 'h1', key: null, ref: null, props: {…}, …}
   $$typeof: Symbol(react.element)
   key: null
  ▶props: {children: 'Namaste Everyone !!'}
   ref: null
   type: "h1"
   _owner: null
  ▶_store: {validated: false}
   _self: null
   _source: null
  ▶[[Prototype]]: Object
```

Example 1:-

If you want to insert an element with id, then we do something like this :-



```
<script>
    const heading = React.createElement(
        "h1",
        {
            id: "title"
        },
        "Namaste Everyone !!"
    );
```



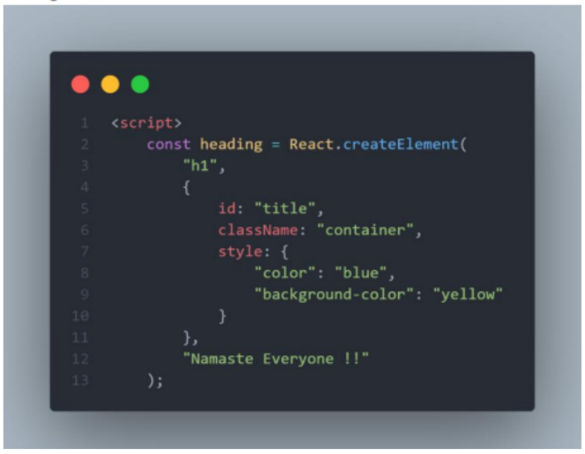
```
▼<body>
  ▼<div id="root">
     <h1 id="title">Namaste Everyone !!</h1>
   </div>
   <!-- Code injected by live-server -->
```

#NOTE:-
We can inject React into any existing projects too, without affecting the other areas. Like if we already have 2 div-s above and below the root div, the React will only affect the root div and the others will be displayed as usual.

If there are already some children present in the root element of React, the React will override it :-

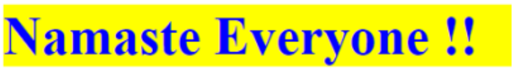
Example 3 :-

React tries to use HTML through JS with the help of JSX and in JSX syntax, class is called className and there are a few other changes.



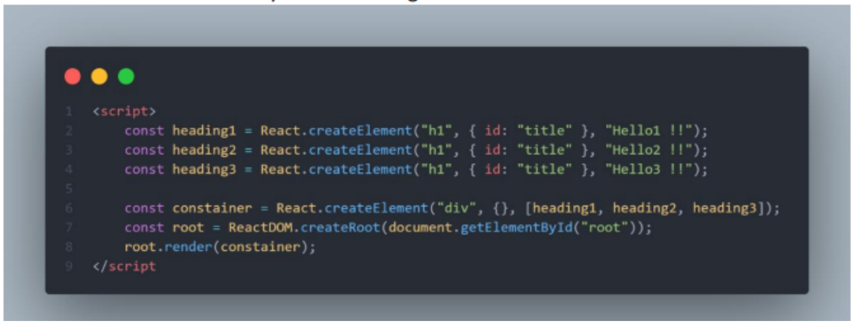
```
<script>
    const heading = React.createElement(
        "h1",
        {
            id: "title",
            className: "container",
            style: {
                "color": "blue",
                "background-color": "yellow"
            }
        },
        "Namaste Everyone !!"
    );
```

Output :-



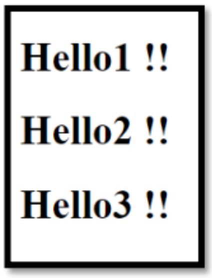
Example-4 :-

We can also insert multiple children elements inside the root element by creating an element which contains all those children elements in an array and rendering that element inside the React root like :-



```
<script>
    const heading1 = React.createElement("h1", { id: "title" }, "Hello1 !!");
    const heading2 = React.createElement("h1", { id: "title" }, "Hello2 !!");
    const heading3 = React.createElement("h1", { id: "title" }, "Hello3 !!");

    const constainer = React.createElement("div", {}, [heading1, heading2, heading3]);
    const root = ReactDOM.createRoot(document.getElementById("root"));
    root.render(constainer);
</script
```

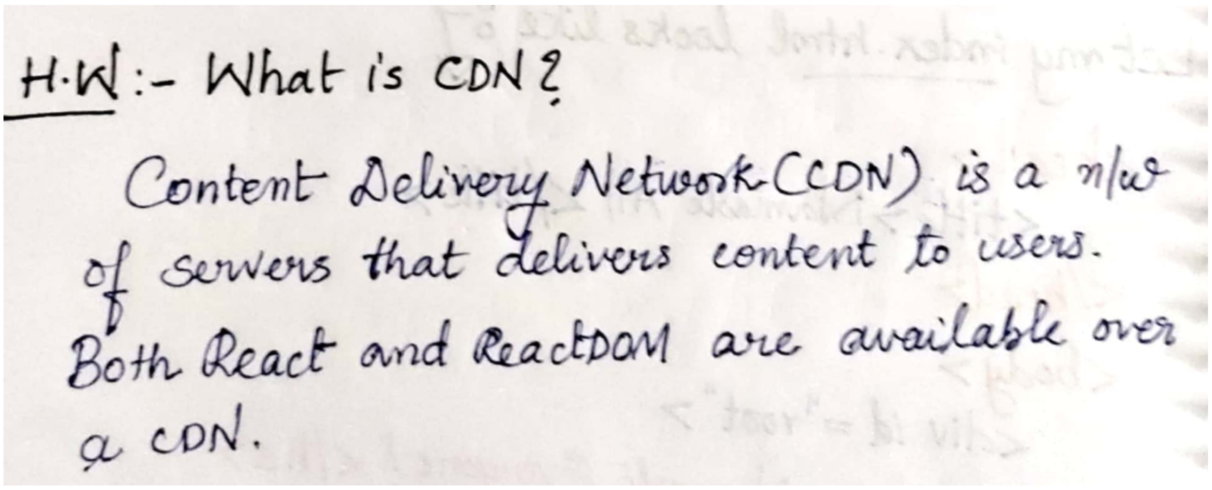


Important points :-

- If we write the React script before the 2 links, then we will get an error in the console "React not defined" and the react elements will not be rendered properly
- Developers write :-

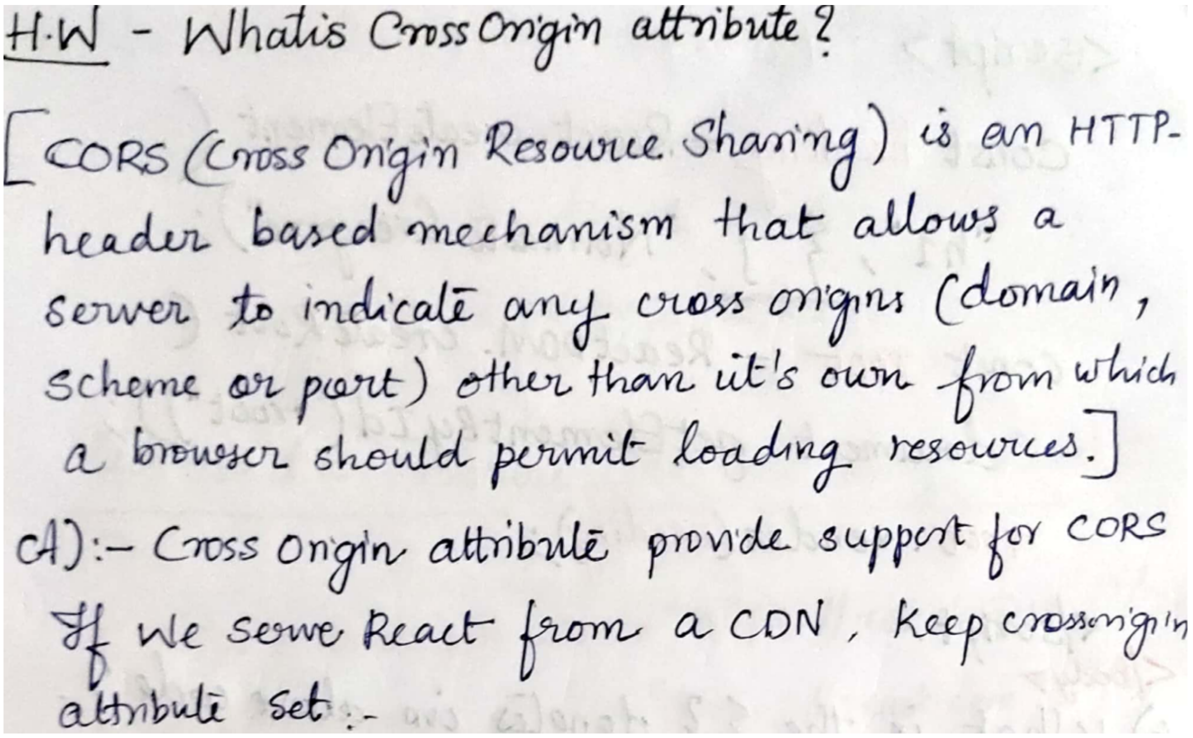
  <div id="root">Not Rendered Yet</div>

  Because, if something happens due to which react is not rendered properly in the root element, that will be easily detected.
- The heading1,2 and 3 in example-4, are all react elements and we know that they are nothing but objects and render function is used to overwrite the contents of root HTML element with those elements.



H·W :- What is CDN ?

Content Delivery Network (CDN) is a n/w of servers that delivers content to users. Both React and ReactDOM are available over a CDN.



H·W - What is Cross Origin attribute ?

[CORS (Cross Origin Resource Sharing) is an HTTP-header based mechanism that allows a server to indicate any cross origins (domain, scheme or port) other than it's own from which a browser should permit loading resources.]

A):- Cross origin attribute provide support for CORS If we serve React from a CDN, keep crossorigin attribute set :-

Why is React known as React? Ans : React is a JavaScript library for building user interfaces that was developed and open-sourced by Facebook in 2013. It is known as React because it was designed to help developers "react" to changes in the state of an application, by efficiently rendering and updating the UI in response to those changes.

What is the difference between React and ReactDOM ?
React is a library for building user interfaces, while ReactDOM is a library for interacting with the DOM and rendering React components to the web page. While they are often used together, they serve different purposes and can be used independently of each other.

What is the difference between react.development.js and react.production.js files via CDN? Ans: react.development.js and react.production.js are two different versions of the React library that are available via CDN (Content Delivery Network). These files contain the same code, but are optimised for different environments and have different features enabled.

The react.development.js file is intended for use during development and includes additional features and debugging tools that can be helpful when building and testing React applications. These features may include additional warning messages, error checking, and other tools that can help identify problems or potential issues with the code.

The react.production.js file, on the other hand, is intended for use in production environments and has been optimised for performance and size. It does not include the additional debugging tools and may have other features disabled in order to reduce the size of the file and improve performance.

What is async and defer? async and defer are attributes that can be used in the script tag to specify the loading behaviour of a script.
The async attribute tells the browser to download the script in the background while the page is still loading, and to execute the script as soon as it is available. This can improve the loading speed of the page, as the script does not block the rendering of the page while it is being downloaded. However, the script may not necessarily be executed in the order it appears in the HTML, as it may be executed as soon as it is available.
The defer attribute tells the browser to download the script in the background while the page is still loading, but to execute the script only after the page has finished loading. This can also improve the loading speed of the page, as the script does not block the rendering of the page. However, unlike async, defer ensures that the script is executed in the order it appears in the HTML.
Both async and defer are used to improve the performance and loading speed of web pages by allowing scripts to be loaded and executed in a non-blocking manner. However, they have different behaviours and may be more suitable for different use cases.