Chapter1 – Inception

What is React? Why is `React` known as `React`?

React is a JavaScript Library. React is named React because it quickly reacts to changes without reloading the whole page.

What is a Library?

Library is a collections of prewritten code snippets that can be used and reused to perform certain tasks. A particular JavaScript library code can be plugged into application code which leads to faster development and fewer vulnerabilities to have errors.

Examples - React, jQuery, Underscore

What is a Framework?

Framework provides a basic foundation or structure for a website or an application.

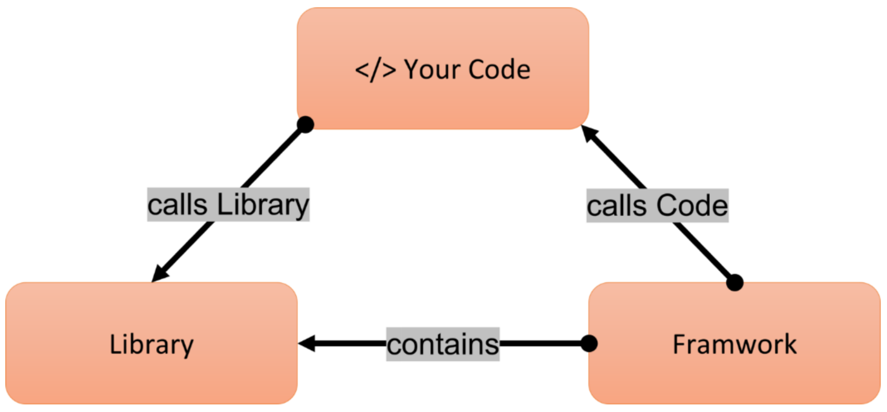
Example - Angular

Similarities between Library and Framework?

Frameworks and libraries are code written by third parties to solve regular/common problems or to optimise performance.

Difference between Library and Framework?

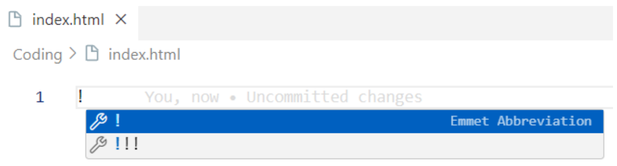
A key difference between the two is Inversion of control. When using a library, the control remains with the developer who tells the application when to call library functions. When using a framework, the control is reversed, which means that the framework tells the developer where code needs to be provided and calls it as it requires.



What is `Emmet`?

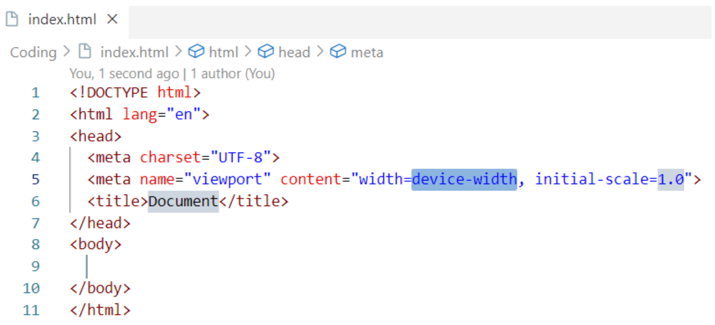
It's a tool for IDE and text editors that help developers like us to generate full fledge boiler plate code snippets from shortcuts.

For example, when we type ! in an html file and press enter –



index.html ×
Coding > index.html

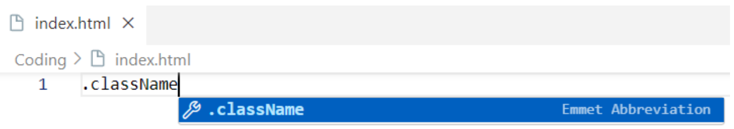
```
!        You, now • Uncommitted changes
  ! Emmet Abbreviation
  !!!
```



index.html ×
Coding > index.html > html > head > meta
You, 1 second ago | 1 author (You)

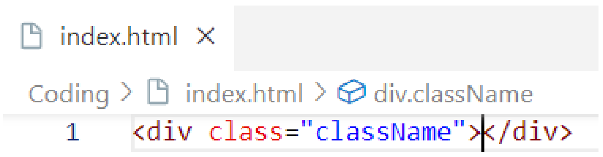
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  
</body>
</html>
```

when we type .<className> in an html file and press enter –



index.html ×
Coding > index.html

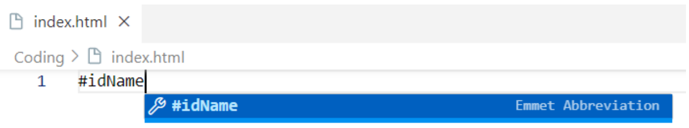
```
.className
  .className Emmet Abbreviation
```



index.html ×
Coding > index.html > div.className

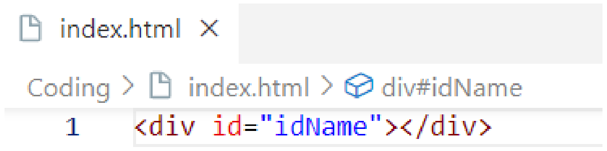
```
<div class="className"></div>
```

when we type #<idName> in an html file and press enter –



index.html ×
Coding > index.html

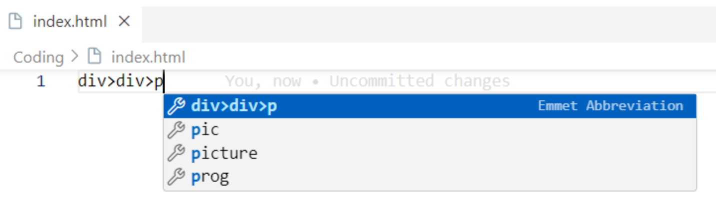
```
#idName
  #idName Emmet Abbreviation
```



index.html ×
Coding > index.html > div#idName

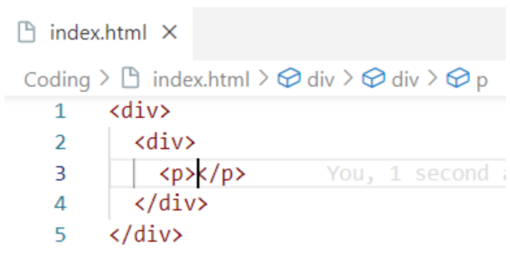
```
<div id="idName"></div>
```

when we want to generate a nested html elements structure we need below syntax.
Parent > child 1 > child's child > and so on …



index.html ×
Coding > index.html

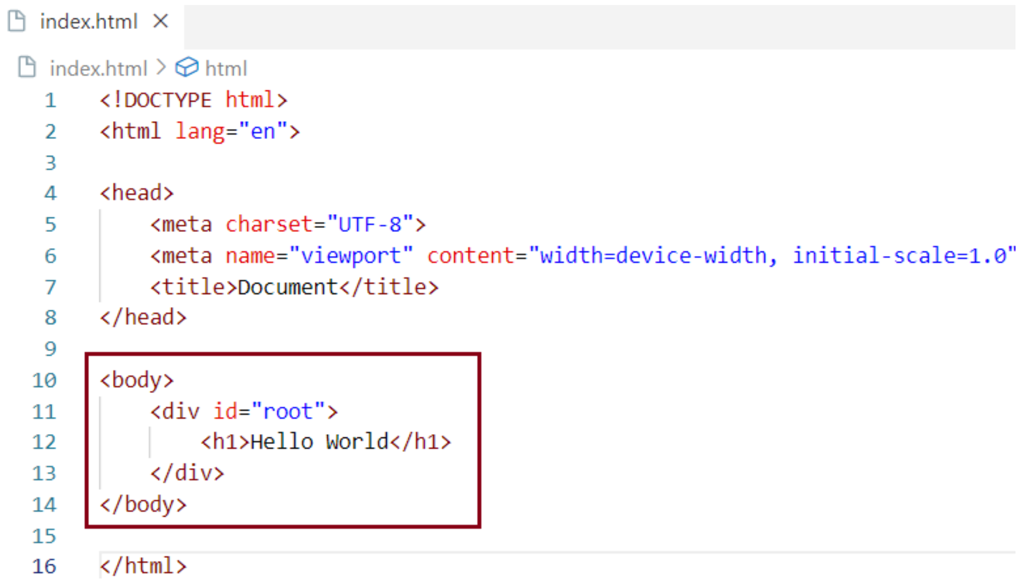
```
div>div>p        You, now • Uncommitted changes
  div>div>p Emmet Abbreviation
  pic
  picture
  prog
```



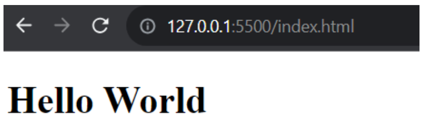
index.html ×
Coding > index.html > div > div > p

```
<div>
  <div>
    <p></p>        You, 1 second
  </div>
</div>
```

Before learning react, let's create a basic hello world program using html.

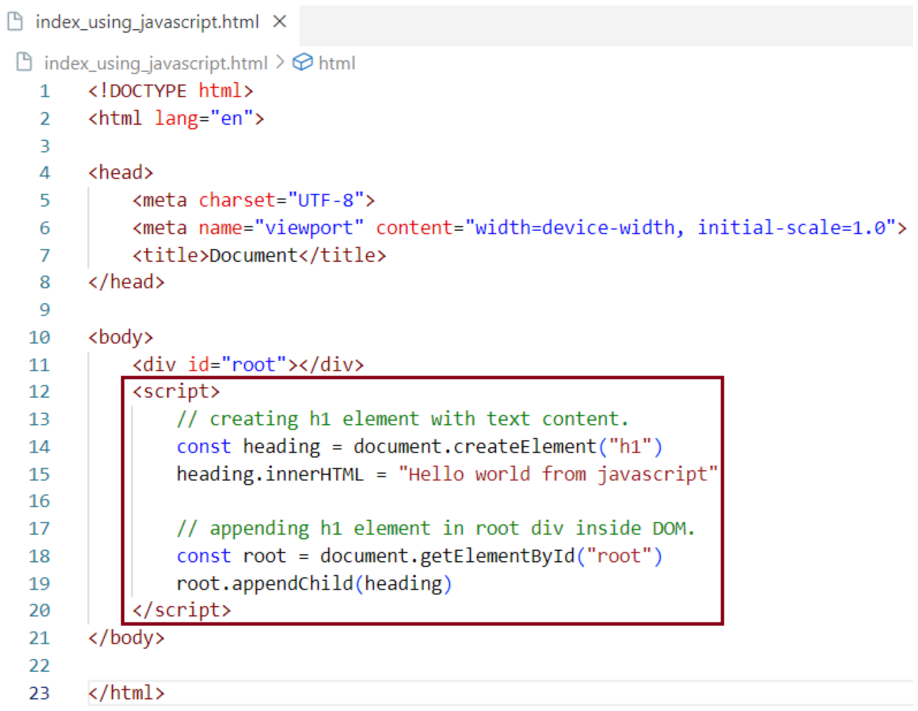


```html
<!DOCTYPE html>
<html lang="en">

<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>
```



Hello World

Now let's create a basic hello world program using JavaScript.



index_using_javascript.html
index_using_javascript.html > 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"></div>
    <script>
        // creating h1 element with text content.
        const heading = document.createElement("h1")
        heading.innerHTML = "Hello world from javascript"

        // appending h1 element in root div inside DOM.
        const root = document.getElementById("root")
        root.appendChild(heading)
    </script>
</body>

</html>
```

Now let's create a basic hello world program using react. Our browser does not know what react is, we need to add react into our project to use all of its superpowers. There are two ways of adding react into our project.

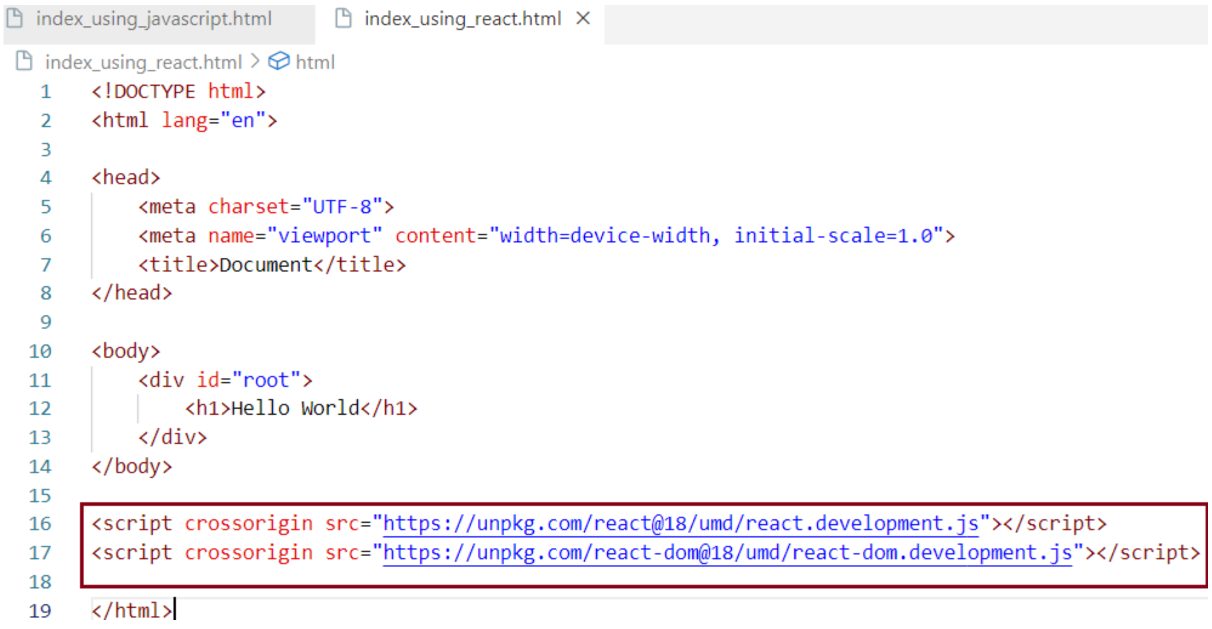
Way1: Using CDNs

Way 2: Install react library via NPM and then import react into your project and use it.

In this chapter we are going discuss about adding react into our project using CDN.

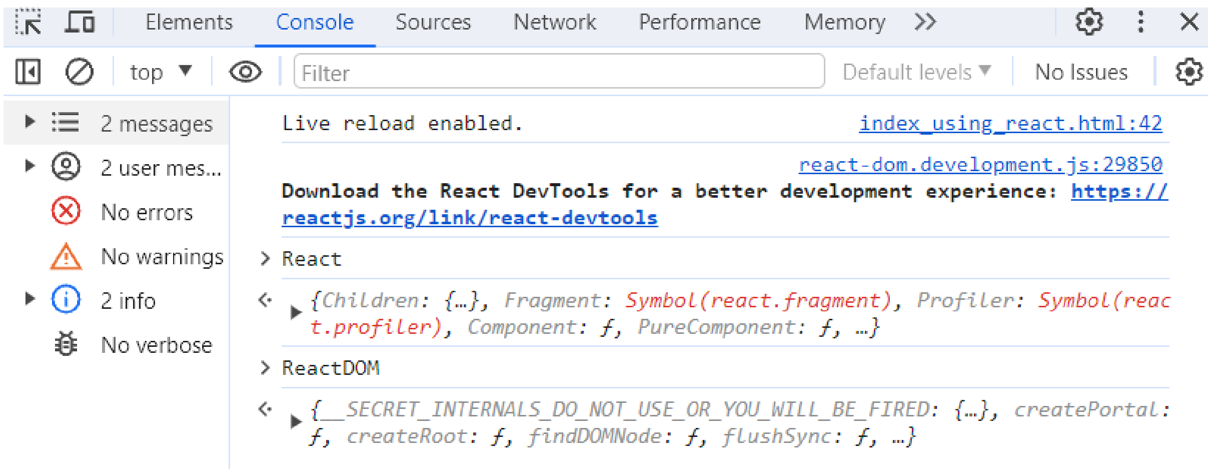
What is `CDN`? Why do we use it?

CDN stands for Content Delivery Network which refers to a geographically distributed group of servers that work together to provide fast delivery of Internet content. The main use of a CDN is to deliver content through a network of servers in a secure and efficient way. CDN has 100% uptime.



```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>

<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>
```

Using these two CDN links we made react available to the browser. This is the shortest react program. why? because If we go to the browser and type React and ReactDOM we will get objects like -



```
Live reload enabled.                                    index_using_react.html:42
                                                   react-dom.development.js:29850
Download the React DevTools for a better development experience: https://
reactjs.org/link/react-devtools
> React
< ▸ {Children: {…}, Fragment: Symbol(react.fragment), Profiler: Symbol(reac
  t.profiler), Component: ƒ, PureComponent: ƒ, …}
> ReactDOM
< ▸ {__SECRET_INTERNALS_DO_NOT_USE_OR_YOU_WILL_BE_FIRED: {…}, createPortal:
  ƒ, createRoot: ƒ, findDOMNode: ƒ, flushSync: ƒ, …}
```

These are coming from the CDNs that we have just imported.
The first CDN link https://unpkg.com/react@18/umd/react.development.js is the core file of react.

The second CDN link https://unpkg.com/react-dom@18/umd/react-dom.development.js is the file which we need for DOM Operations.

What is difference between react and react-dom/client packages?

The react package contains React.createElement(), React.Component, React.Children, and other helpers related to elements and component classes. We can use these helper APIs to create react components or elements. The react-dom/client package contains ReactDOM.render() which takes a react element as argument and it defines where the react element has to be rendered inside DOM.

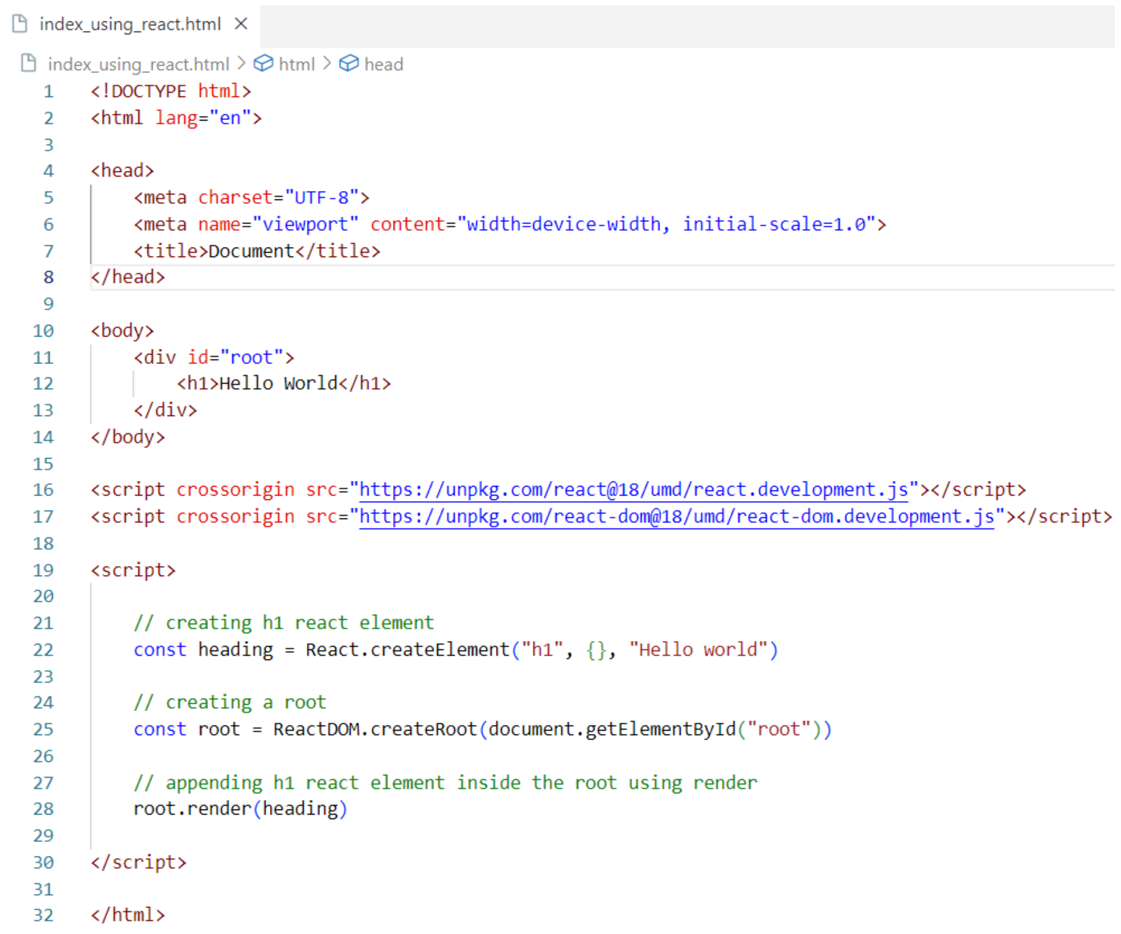
What is crossorigin in script tag?

The crossorigin attribute sets the mode of the request to an HTTP CORS Request. Web pages often make requests to load resources on other servers. Here is where CORS comes in.

A cross-origin request is a request for a resource (e.g. style sheets, iframes, images, fonts, or scripts) from another domain. CORS is used to manage cross-origin requests.

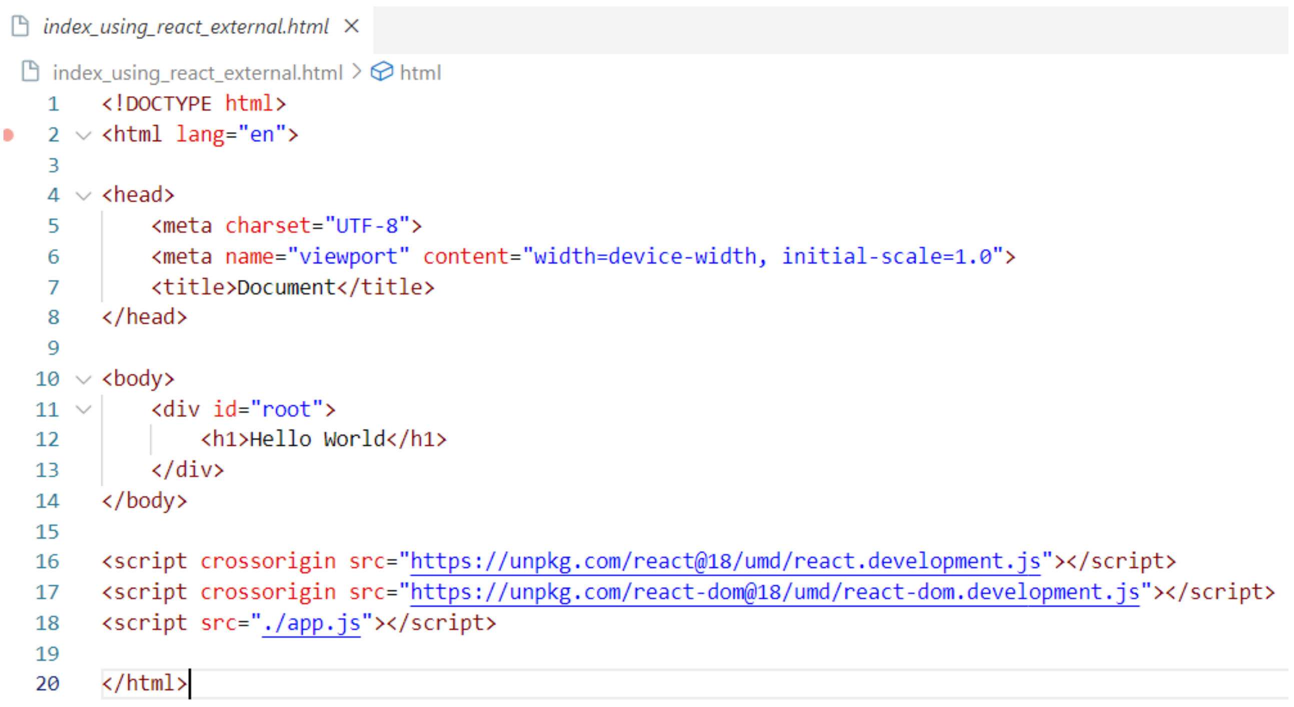
CORS stands for Cross-Origin Resource Sharing, and is a mechanism that allows resources on a web page to be requested from another domain outside their own domain. It defines a way of how a browser and server can interact to determine whether it is safe to allow the cross-origin request. CORS allows servers to specify who can access the assets on the server, among many other things.

If we navigate to these CDN links, we get source code of React which is written using JavaScript by Facebook developers. So, react in the end of the day is JavaScript. Now let's continue writing hello world program using react.



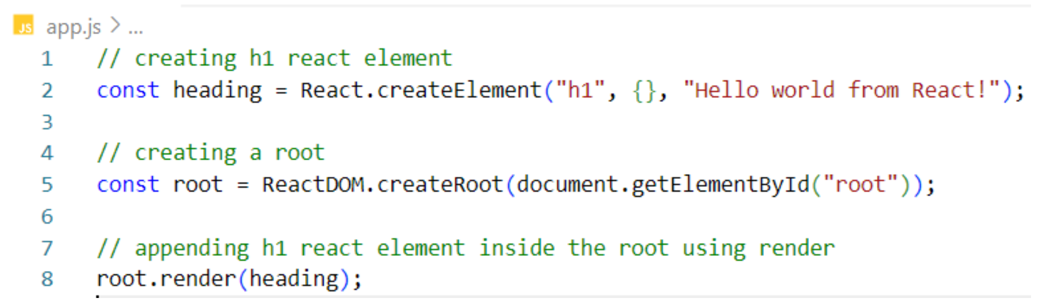
```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <div id="root">
        <h1>Hello World</h1>
    </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>

<script>

    // creating h1 react element
    const heading = React.createElement("h1", {}, "Hello world")

    // creating a root
    const root = ReactDOM.createRoot(document.getElementById("root"))

    // appending h1 react element inside the root using render
    root.render(heading)

</script>

</html>
```

Let’s keep this script code in an external JavaScript file and put a reference to this file in html code.


index_using_react_external.html

index_using_react_external.html > 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>

<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 src="./app.js"></script>

</html>
```


app.js > ...

```javascript
// creating h1 react element
const heading = React.createElement("h1", {}, "Hello world from React!");

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

// appending h1 react element inside the root using render
root.render(heading);
```

The application behaves the same way it was behaving before.

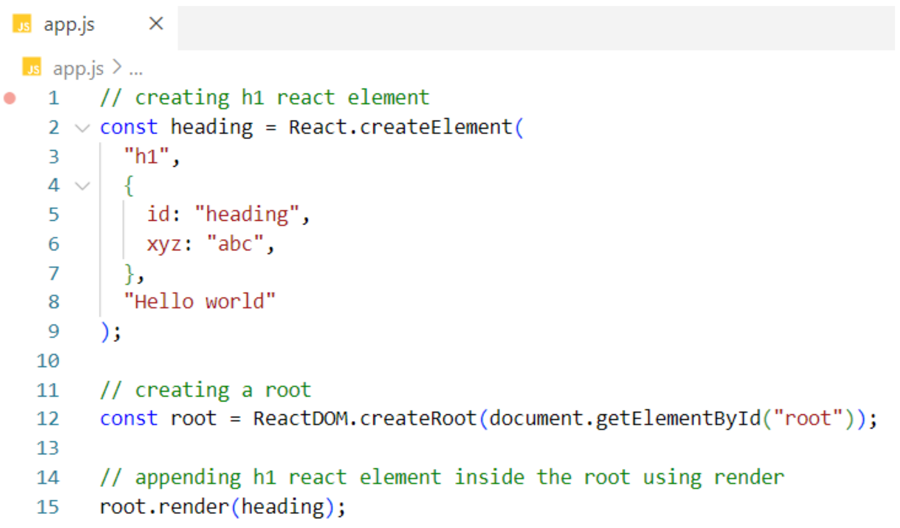
React.createElement

createElement is a top-level API that React uses to create a React element. This API takes three arguments such as tagName, props & children and returns a JavaScript object AKA react element.

Syntax - React.createElement(tagName, {props}, [children])

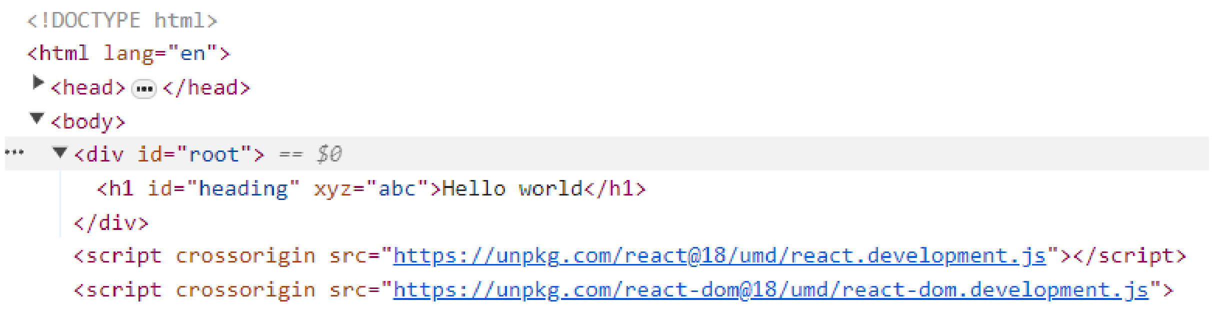
Where tagName represents Name of html tag like h1, div etc, Props represents attribute to html element, props could also be null & children represents the content of html element or an array list of react elements.

In the above example lets add some attributes to the h1 react element. We configure these attributes inside props object.



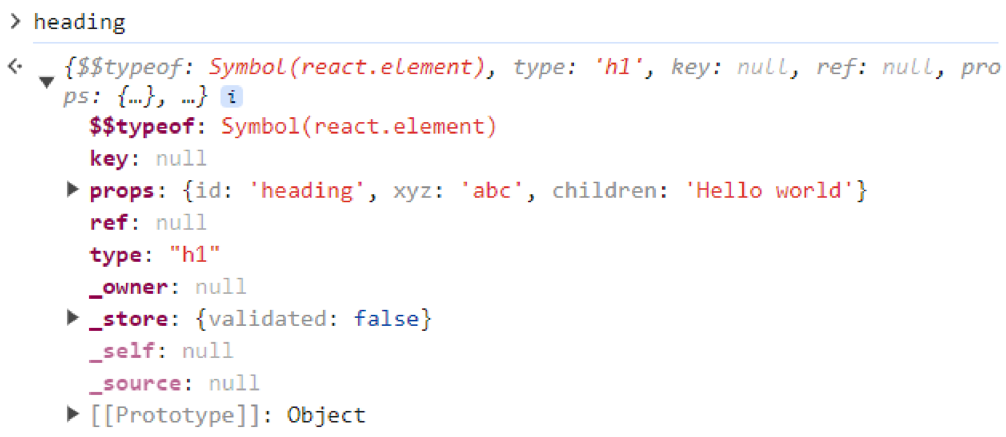
```
// creating h1 react element
const heading = React.createElement(
  "h1",
  {
    id: "heading",
    xyz: "abc",
  },
  "Hello world"
);

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

// appending h1 react element inside the root using render
root.render(heading);
```

We see these attributes got added to the h1 element in the DOM.



```
<!DOCTYPE html>
<html lang="en">
 ▶<head> ··· </head>
 ▼<body>
 ··· ▼<div id="root"> == $0
        <h1 id="heading" xyz="abc">Hello world</h1>
      </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">
```

In above JavaScript code snippet, heading is a plain JavaScript object which is a react element.



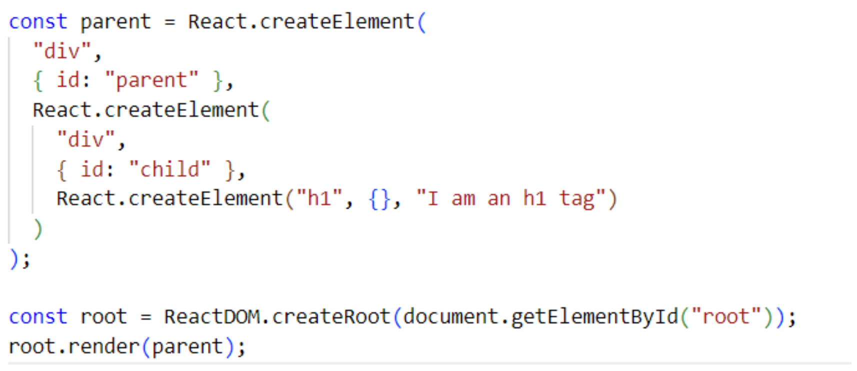
```
> heading
<· ▼{$$typeof: Symbol(react.element), type: 'h1', key: null, ref: null, pro
    ps: {…}, …} i
      $$typeof: Symbol(react.element)
      key: null
    ▶ props: {id: 'heading', xyz: 'abc', children: 'Hello world'}
      ref: null
      type: "h1"
      _owner: null
    ▶ _store: {validated: false}
      _self: null
      _source: null
    ▶ [[Prototype]]: Object
```

How to create nested elements using react?

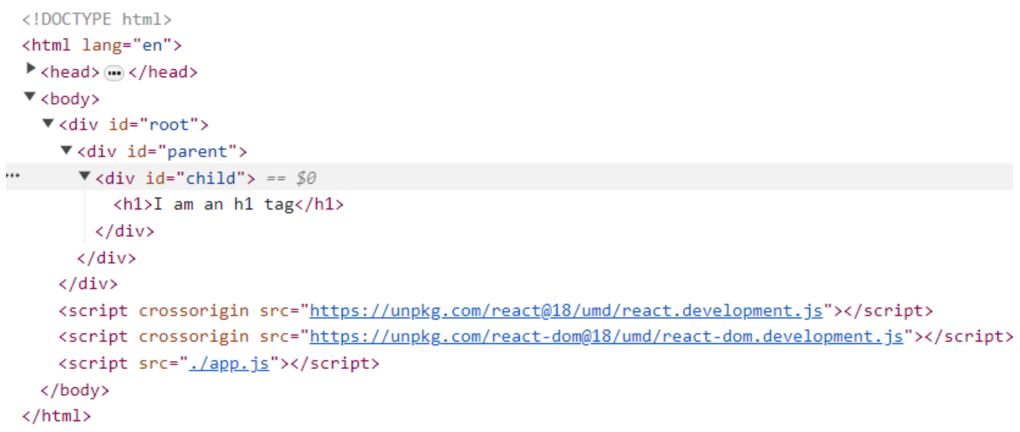
Representation of 2 level Nested elements using html

```
<div id="parent">
  <div id="child">
    <h1>I am an h1 tag</h1>
  </div>
</div>
```

Representation of 2 level Nested elements using react



```
const parent = React.createElement(
  "div",
  { id: "parent" },
  React.createElement(
    "div",
    { id: "child" },
    React.createElement("h1", {}, "I am an h1 tag")
  )
);

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

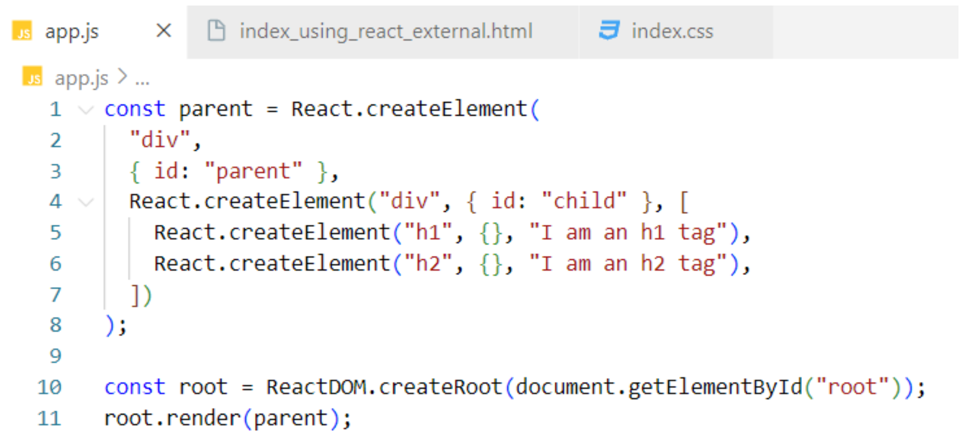


```
<!DOCTYPE html>
<html lang="en">
▶<head> ··· </head>
▼<body>
  ▼<div id="root">
    ▼<div id="parent">
      ▼<div id="child"> == $0
          <h1>I am an h1 tag</h1>
        </div>
      </div>
    </div>
    <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
    <script src="./app.js"></script>
  </body>
</html>
```

Representation of sibling elements using html – (in below example h1 and h2 are siblings)

```
<div id="parent">
  <div id="child">
    <h1>I am an h1 tag</h1>
    <h2>I am an h2 tag</h1>
  </div>
</div>
```

Representation of sibling elements using react - react wraps these two siblings inside an array
For one child it does not use array as we have seen above.



```
const parent = React.createElement(
  "div",
  { id: "parent" },
  React.createElement("div", { id: "child" }, [
    React.createElement("h1", {}, "I am an h1 tag"),
    React.createElement("h2", {}, "I am an h2 tag"),
  ])
);

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

Representation of Nested sibling elements using html

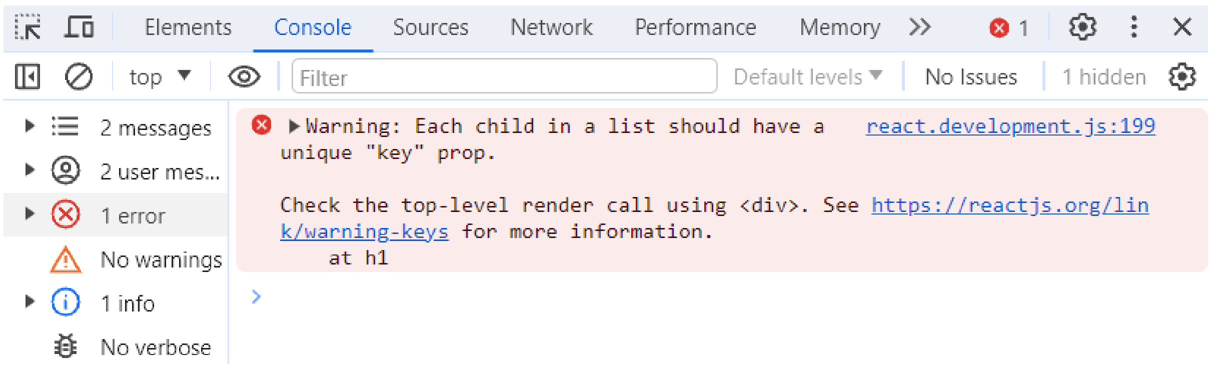
```
<div id="parent">
  <div id="child1">
    <h1>I am an h1 tag</h1>
    <h2>I am an h2 tag</h1>
  </div>
  <div id="child2">
    <h1>I am an h1 tag</h1>
    <h2>I am an h2 tag</h1>
  </div>
</div>
```

Representation of Nested sibling elements using react

```
const parent = React.createElement("div", { id: "parent" }, [
  React.createElement("div", { id: "child1" }, [
    React.createElement("h1", {}, "I am an h1 tag"),
    React.createElement("h2", {}, "I am an h2 tag"),
  ]),
  React.createElement("div", { id: "child2" }, [
    React.createElement("h1", {}, "I am an h1 tag"),
    React.createElement("h2", {}, "I am an h2 tag"),
  ]),
]);

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

The code works fine but we get and warning in the console when multiple children were introduced

This error is genuine because we have not provided Key for each of the children, because react tracks these elements by their Key. We will discuss more on keys in later chapters.



```
▶Warning: Each child in a list should have a    react.development.js:199
unique "key" prop.

Check the top-level render call using <div>. See https://reactjs.org/lin
k/warning-keys for more information.
    at h1
```

If we have deep level nested elements, writing code like this will not be useful as it is difficult to read, maintain and test. To overcome this problem JSX was introduced. We will discuss JSX in later chapters.

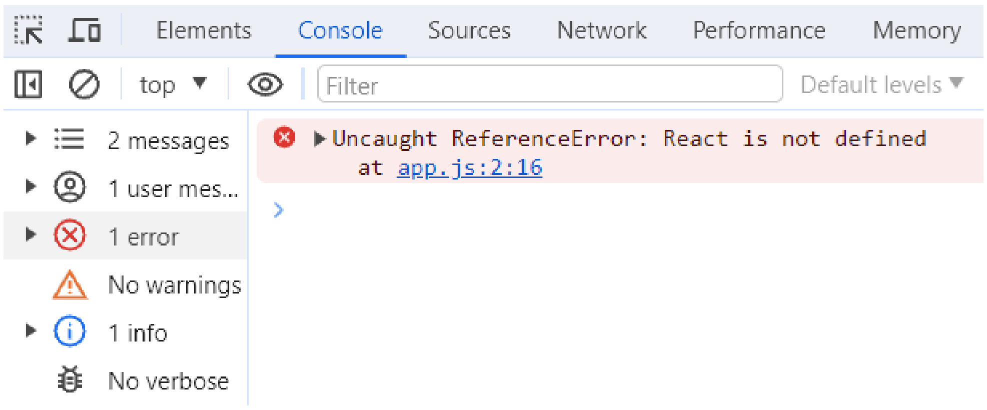
Important Observations

1. Order of the script tag matters.

We have moved our script tag from bottom to two levels up that is before CDN fetches react code into our browser.

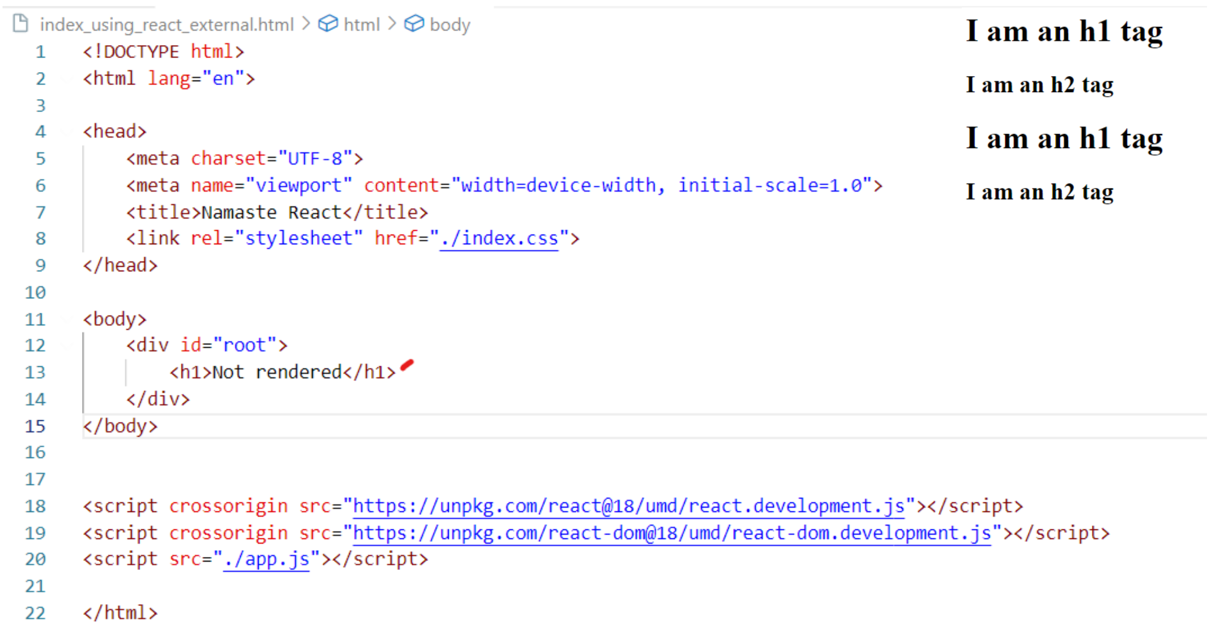
```
<script src="./app.js"></script>
<script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
```

In this case we get an error that React is not defined. We are trying to access React before react source code is available to our browser.



```
▶Uncaught ReferenceError: React is not defined
    at app.js:2:16
```

2.While rending react element into the dom React overrides the children elements present inside root element.



index_using_react_external.html > html > body

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Namaste React</title>
    <link rel="stylesheet" href="./index.css">
</head>

<body>
    <div id="root">
        <h1>Not rendered</h1>
    </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>
<script src="./app.js"></script>

</html>
```

I am an h1 tag

I am an h2 tag

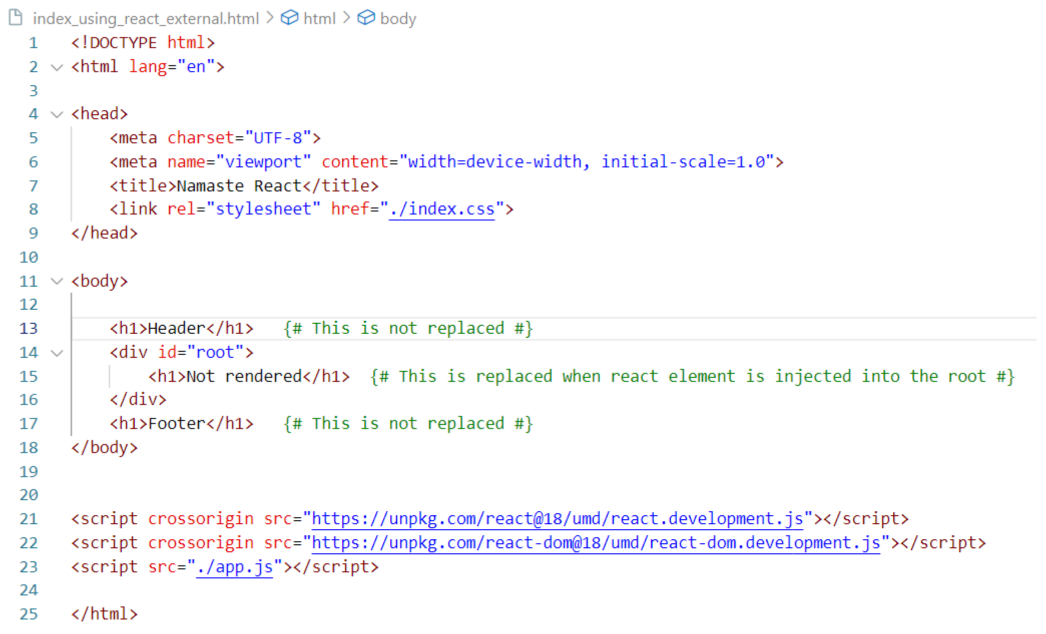
I am an h1 tag

I am an h2 tag

Not rendered will be replaced by the react element that we are pushing into the root. So, when we load the page Not rendered will be displayed for a fraction of seconds until react source code is fetched from the CDNs and available to the browser. Once react code is available, this will be replaced by the react element which is two children div elements under one parent div.

3. We can use react to a specific component or part of our application.

In next example, we are using react on root component, we are not using react on header and footer component. This is why we call react as a library. because we can use react in a small portion of our app and we as developers have control on our application.



index_using_react_external.html > html > body

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Namaste React</title>
    <link rel="stylesheet" href="./index.css">
</head>

<body>

    <h1>Header</h1>   {# This is not replaced #}
    <div id="root">
        <h1>Not rendered</h1>  {# This is replaced when react element is injected into the root #}
    </div>
    <h1>Footer</h1>   {# This is not replaced #}
</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>
<script src="./app.js"></script>

</html>
```

ReactDOM.createRoot

createRoot is a ReactDOM API which creates a root in the DOM where JS engine runs react application. It returns an object with 2 methods render and unmount

Syntax - ReactDOM.createRoot(document.getElementById(<rootID>))

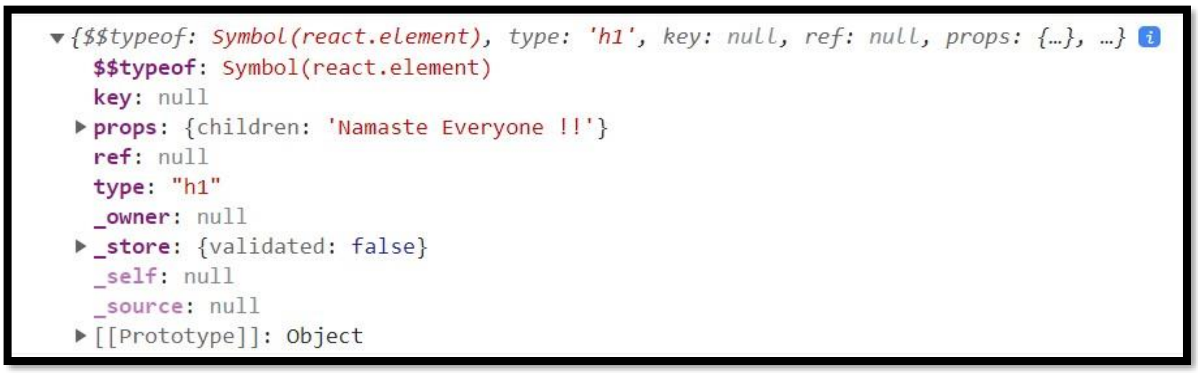
render

render method takes react element or JSX as argument and renders it inside the root element inside DOM.This method returns undefined.

Syntax - <root>.render(<JSX/React element>)

If we log heading in the console, we will get a plain JavaScript 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
```

What is difference between `Development` and `production` stage of an application?

In `Development` stage, application is not public meaning we don’t deploy the application into production. while `production` is the term used for the same application when it's made `public`. `Development build` is several times `slower` than the `production build`.