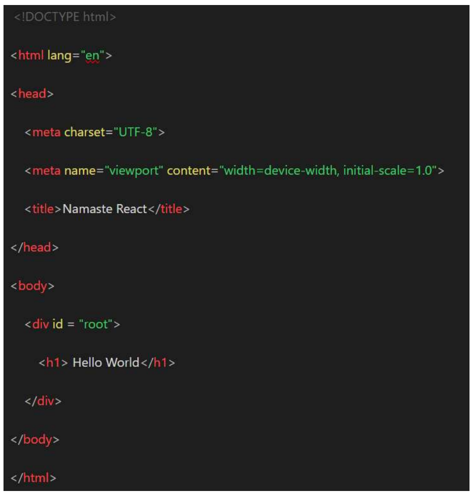
ReactJS - Inception

13 June 2025 12:12 PM

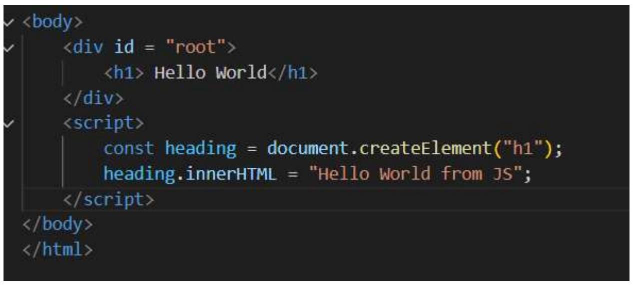
- **First we started by creating a Hello World program using Basic HTML**



```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>

</head>

<body>

  <div id = "root">

    <h1> Hello World</h1>

  </div>

</body>

</html>
```

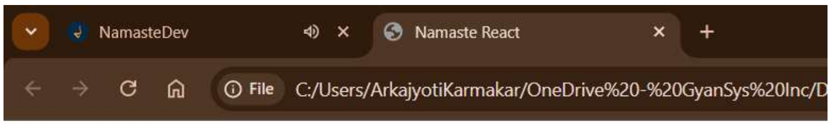
- **Secondly, we will now do the same thing but this time using JavaScript**

  - Here at present I have created a <H1> tag using my document.createELement in JS.
    - Now we need to find a way to put this h1 tag in our root.



```html
<body>
    <div id = "root">
        <h1> Hello World</h1>
    </div>
    <script>
        const heading = document.createElement("h1");
        heading.innerHTML = "Hello World from JS";
    </script>
</body>
</html>
```

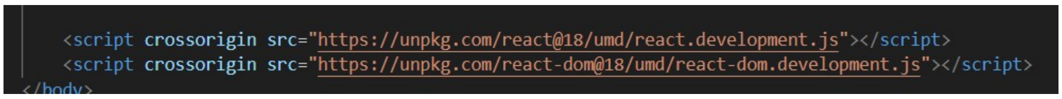
  - Now we will use the **Document.getElementById** to fetch the root.
  - Then we will use the **appendChild** to send the heading to the root.



C:/Users/ArkajyotiKarmakar/OneDrive%20-%20GyanSys%20Inc/D

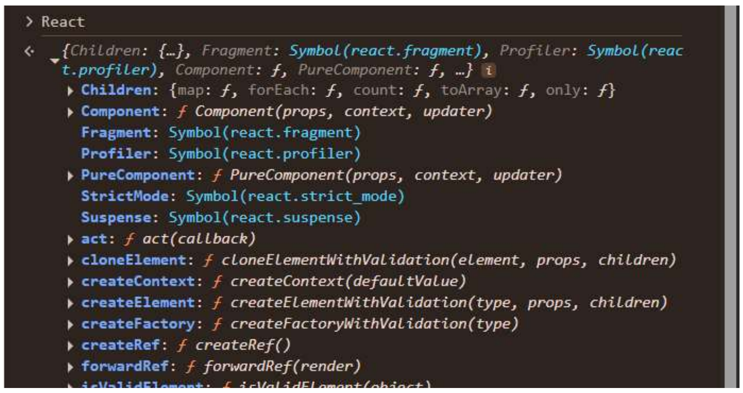
- **Hello World from JS**

- **Now we will fetch the CDN links and paste the same in our script tag.**



```html
    <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>
</body>
```

- **This gives us the superpower of React.**



```
> React
< {Children: {…}, Fragment: Symbol(react.fragment), Profiler: Symbol(reac
  t.profiler), Component: ƒ, PureComponent: ƒ, …} i
  ▸ Children: {map: ƒ, forEach: ƒ, count: ƒ, toArray: ƒ, only: ƒ}
  ▸ Component: ƒ Component(props, context, updater)
    Fragment: Symbol(react.fragment)
    Profiler: Symbol(react.profiler)
  ▸ PureComponent: ƒ PureComponent(props, context, updater)
    StrictMode: Symbol(react.strict_mode)
    Suspense: Symbol(react.suspense)
  ▸ act: ƒ act(callback)
  ▸ cloneElement: ƒ cloneElementWithValidation(element, props, children)
  ▸ createContext: ƒ createContext(defaultValue)
  ▸ createElement: ƒ createElementWithValidation(type, props, children)
  ▸ createFactory: ƒ createFactoryWithValidation(type)
  ▸ createRef: ƒ createRef()
  ▸ forwardRef: ƒ forwardRef(render)
  ▸ isValidElement: ƒ isValidElement(object)
```

- **But there are 2 CDN links why?**

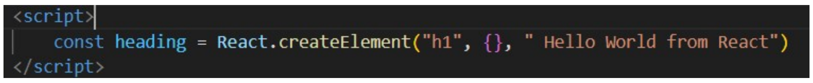
  This is because React is divided into 2 separate libraries.

- **Core Development Library.**
  - **Contains the Core React API : useState, useEffect, createElement, Component etc:**

- **Renderer for the Web :**
  - **Takes React components and **renders them into the actual browser DOM**.**

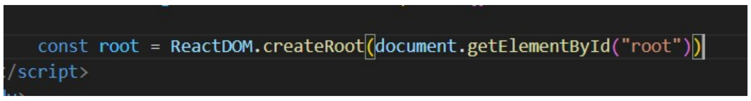
- If we want to create a element in React we generally use the syntax : **React.createElement().**

  - **There are 3 parameters in this function.**
  - We can see while creating an element we are mentioning **3 parameters: (type, props, children).**
  - **type:** A string (e.g., 'div', 'span') for HTML elements, or a React component.
  - **props:** An object containing the properties/attributes for the element.
  - **children:** One or more children, which can be strings, numbers, other React elements, etc



```
<script>
    const heading = React.createElement("h1", {}, " Hello World from React")
</script>
```

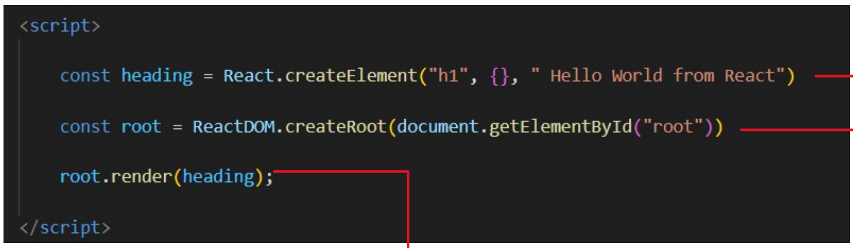
- **Now we need to render this in our root. Things are a bit different from what they are in the traditional JS.**
- First, React needs a route where it does all the DOM manipulation.
- Now we will be using **ReactDOM.createRoute().**
- This is because creating an element is the core concept of React which comes from the React core dev.
- But routes and all these comes from the **ReactDOM library** package where these manipulations are done.

- 

```
    const root = ReactDOM.createRoot(document.getElementById("root"))
/script>
```

- Here we have created a root where our React app is to be rendered.

- **Now we will render it with the heading that we created using the React createElement.**



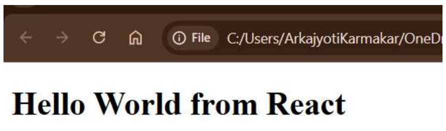
```
<script>

    const heading = React.createElement("h1", {}, " Hello World from React")

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

    root.render(heading);

</script>
```

Here, the React element is being created.

Here, first we will be creating the route on which the React ap will be rendered.

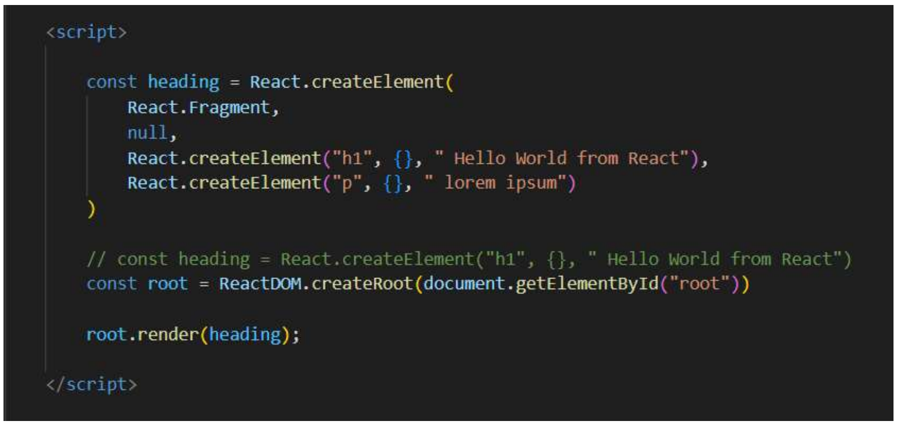
Here, React is being used to add the heading to the route.

- Now we will be able to see that the React app is created.



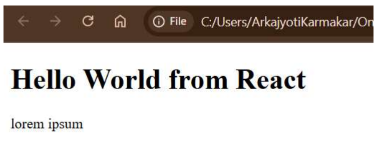
Hello World from React

- In case if we want to create multiple elements then we have to do those as a conjugation of other elements.

- 

```
<script>

    const heading = React.createElement(
        React.Fragment,
        null,
        React.createElement("h1", {}, " Hello World from React"),
        React.createElement("p", {}, " lorem ipsum")
    )

    // const heading = React.createElement("h1", {}, " Hello World from React")
    const root = ReactDOM.createRoot(document.getElementById("root"))

    root.render(heading);

</script>
```

- It has to be nested inside a React Fragment.

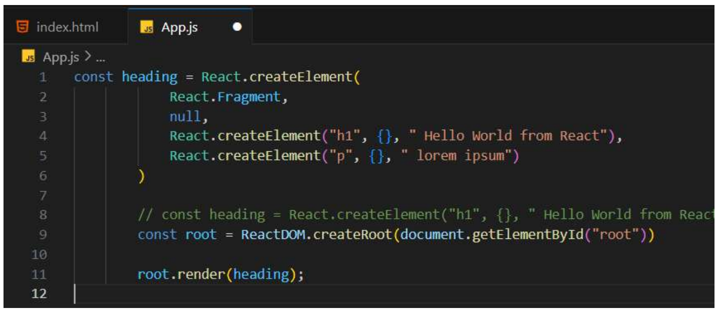
- 

C:/Users/ArkajyotiKarmakar/On

Hello World from React

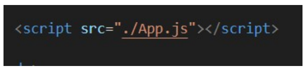
lorem ipsum

- Now we should create a new file and put the react code in there. **(App.js file).**



```
const heading = React.createElement(
        React.Fragment,
        null,
        React.createElement("h1", {}, " Hello World from React"),
        React.createElement("p", {}, " lorem ipsum")
    )

    // const heading = React.createElement("h1", {}, " Hello World from Reac
    const root = ReactDOM.createRoot(document.getElementById("root"))

    root.render(heading);

```

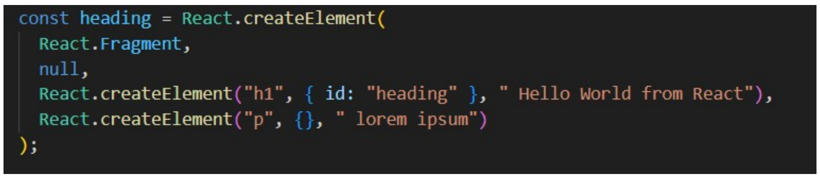
- **Then we will need to inject that file in our index.html file by using the src tag of script.**



```
<script src="./App.js"></script>
```

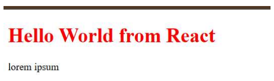
- The most important thing is the management of the DOM.
- React comes with the motto of manipulating the document with the use of JS.

Now we will see about how to give attributes to a particular element.

- 

```
const heading = React.createElement(
  React.Fragment,
  null,
  React.createElement("h1", { id: "heading" }, " Hello World from React"),
  React.createElement("p", {}, " lorem ipsum")
);
```

- We passed an attribute to it.

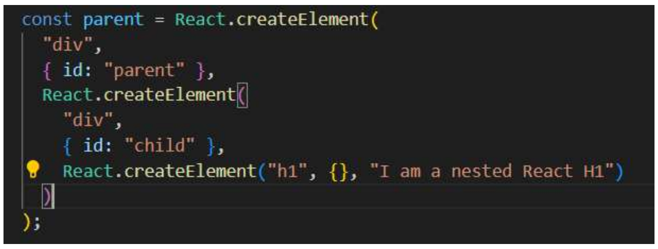


Hello World from React

lorem ipsum

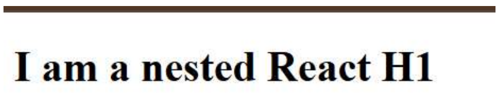
- When we are using the **React.createElement()** we are actually creating a JS object.
- It is converted into an H1 tag when it is being rendered into the root.

- **Here we are creating the nested elements of React.**



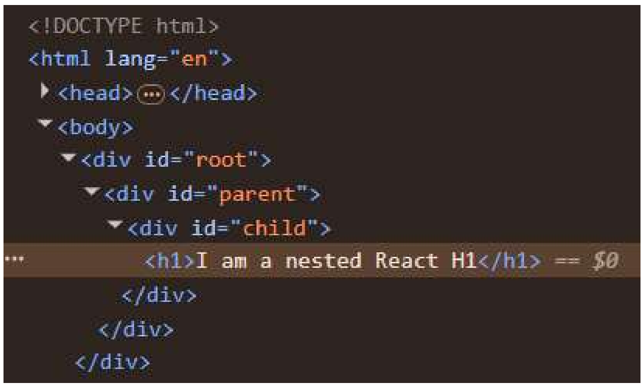
```
const parent = React.createElement(
  "div",
  { id: "parent" },
  React.createElement(
    "div",
    { id: "child" },
    React.createElement("h1", {}, "I am a nested React H1")
  )
);
```

- We can see the result.



I am a nested React H1

- Here we can see that in the HTML the type of nesting we wanted has been formed after rendering the React object.



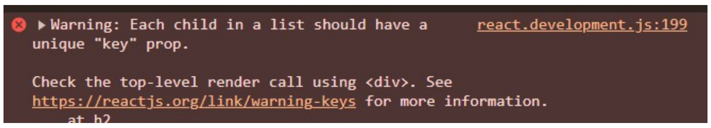
```
<!DOCTYPE html>
<html lang="en">
  <head>…</head>
  <body>
    <div id="root">
      <div id="parent">
        <div id="child">
          <h1>I am a nested React H1</h1> == $0
        </div>
      </div>
    </div>
```

- Now we have learnt nesting.
- But how to make siblings.

Siblings:

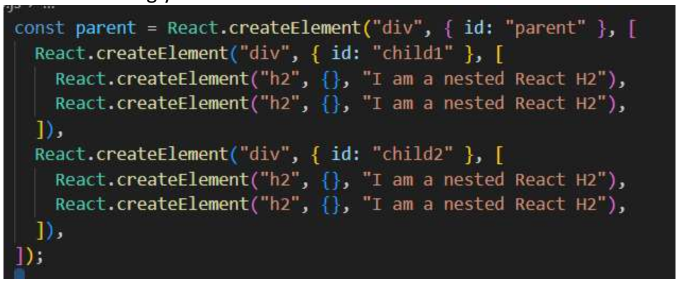
```
const parent = React.createElement(
  "div",
  { id: "parent" },
  React.createElement("div", { id: "child" }, [
    React.createElement("h2", {}, "I am a nested React H2"),
    React.createElement("h2", {}, "I am a nested React H2"),
  ])
);
```

- So, we need to encapsulate the siblings inside a **[] --> Angular Brackets.**
- But also an error is thrown.

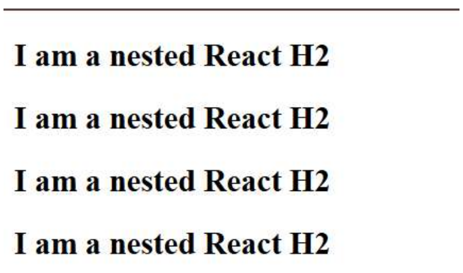


```
▸Warning: Each child in a list should have a      react.development.js:199
unique "key" prop.

Check the top-level render call using <div>. See
https://reactjs.org/link/warning-keys for more information.
    at h2
```

- To **create siblings** we need to enclose it in **Angular Brackets.**

- If we have to create a more nested structure like : **2 childs and their nested elements.**
- But this looks ugly.
- 

```
const parent = React.createElement("div", { id: "parent" }, [
  React.createElement("div", { id: "child1" }, [
    React.createElement("h2", {}, "I am a nested React H2"),
    React.createElement("h2", {}, "I am a nested React H2"),
  ]),
  React.createElement("div", { id: "child2" }, [
    React.createElement("h2", {}, "I am a nested React H2"),
    React.createElement("h2", {}, "I am a nested React H2"),
  ]),
]);
```



I am a nested React H2

I am a nested React H2

I am a nested React H2

I am a nested React H2

- **So now it will be reduced using React.**
- **This is why JSX comes into the picture.**

```
<div id = "root">
    <h1>I am here</h1>
</div>
```

Previously this was there.

- Now if we do **root.render(parent);**
- Then it will replace the parent react element with the text that was there previously.
- This is because as it starts executing then the data, First the line is loaded then as React is loaded it loads the parent element along with the children elements.

- We call React as a library because it can work in the place particularly mentioned by us.