# **What is react?**

# **Intro to React Elements**

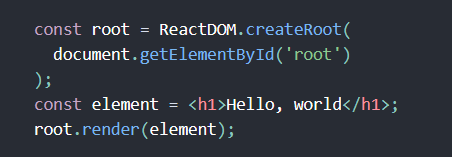
## **Adding React to your project**

* The first step here of adding some links to the CDN of react. CDN stands for Content Delivery Network
* In the <head> tag of HTML page adds the links for the React library, React DOM
* So, react-dom is everything we need to actually add this react to the page. Think of this link as being a quick way to add the react library to your page and the browser so that all of the functions and features you might need from react are preloaded.

## **Creating React Element**

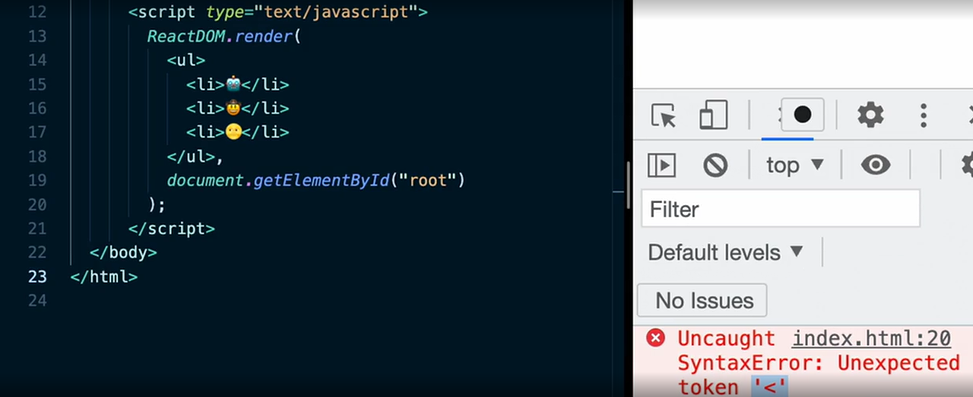


* We call this a “root” DOM node because everything inside it will be managed by React DOM.
* Applications built with just React usually have a single root DOM node. If you are integrating React into an existing app, you may have as many isolated root DOM nodes as you like.
* To render a React element, first pass the DOM element to [ReactDOM.createRoot()](https://reactjs.org/docs/react-dom-client.html#createroot), then pass the React element to root.render():



## **Refactoring Elements using JSX**

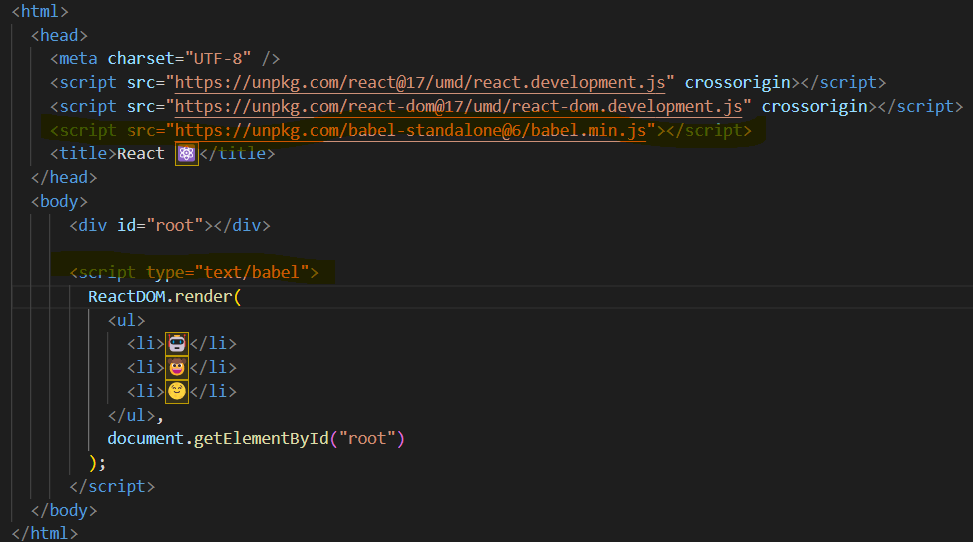
* JSX or JavaScript as XML is an HTML-like syntax for creating UI elements.



* We are getting the error while rendering the JSX tags because it’s not supported by the browser. This can be solved using the **Bable**.

## **Incorporating Bable**

* When we use a tool like Babel, it's going to take this code that's not supported by the browser and turn it into code



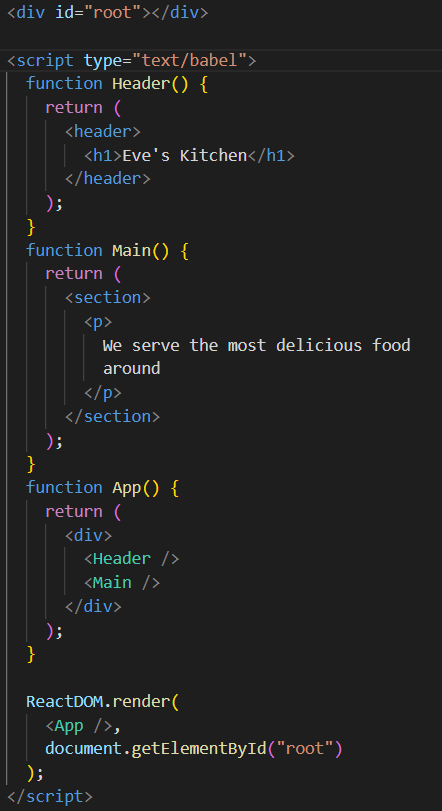
## **Working with JSX syntax**

* We can use it to inject dynamic content into our tags simply by referencing them by variable name

# **React Components**

## **Creating a React Component**

* Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
* Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.



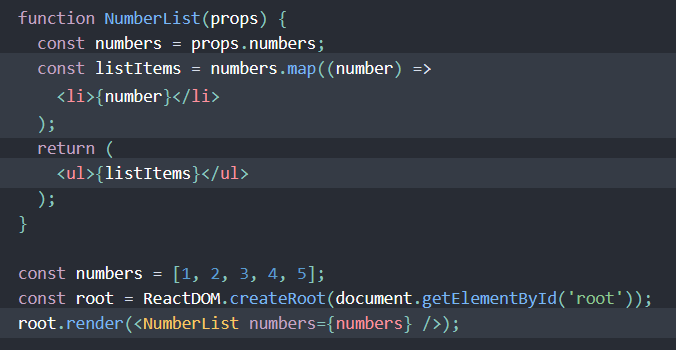
## **Adding Component Properties**

* So, we want to think about the props object as being this little container where you can place information in for any component. When we render the component, we'll pass the properties into the component. And then within that component, we'll reference them in the props object with dot notation



## **Working with lists**

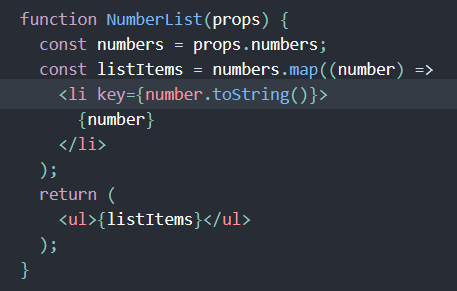
* Usually, you would render lists inside a [component](https://reactjs.org/docs/components-and-props.html). We have created a component that accepts an array of numbers and outputs a list of elements.



* When you run this code, you’ll be given a warning that a key should be provided for list items. **A “key” is a special string attribute you need to include when creating lists of elements.**

## **Adding Keys to List items**

* Let’s assign a key to our list items inside numbers.map() and fix the missing key issue.



## **Displaying images with React**

* We use the <img/> tag to insert image in React component
* Add the alt=”” attribute to each image to make it more accessible

## **Using Fragments**

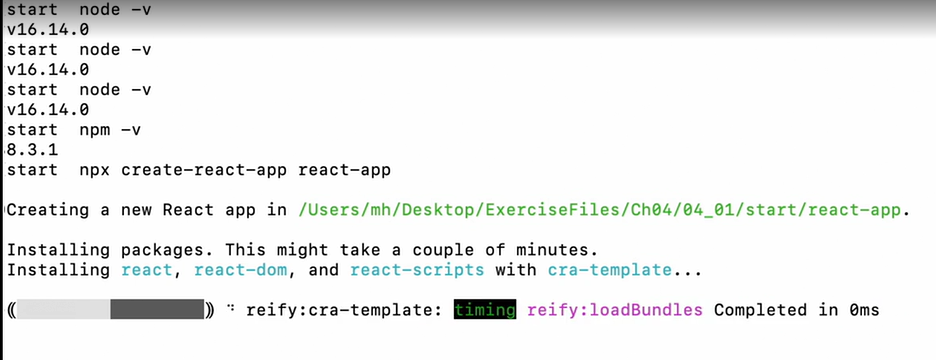
* A common pattern in React is for a component to return multiple elements.
* Fragments let you group a list of children without adding extra nodes to the DOM.



# **React State in the Component Tree**

## **Generating a project with Create React App**

* Download ***Node.js*** if you don’t have.
* Next, check the ***npm*** version if you have installed.
* Finally, run the command ***npx create-react-app react-app***



* Once app is created, go to the correct folder i.e., ***“react-app”*** and run the command as shown below to start –

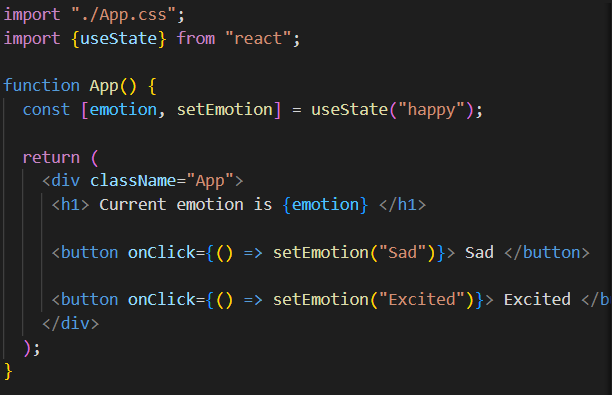


## **Destructuring Array and objects**

* In Array Destructuring we are assigning the variable name based on the position in array

## **Understanding the useState hook**

* useState is a Hook that lets you add React state to function components



* **What does calling useState do?** It declares a “state variable”. Our variable is called emotion but we could call it anything else, like banana. This is a way to “preserve” some values between the function calls — useState is a new way to use the exact same capabilities that this.state provides in a class. Normally, variables “disappear” when the function exits but state variables are preserved by React.

## **Working with useEffect hook**

* It's typically used to manage side effects that aren't related to components render
* Things like console messages, loading data, and working with animations can often benefit from useEffect.



## **Understanding the dependency array**



## **Incorporating useReducer**

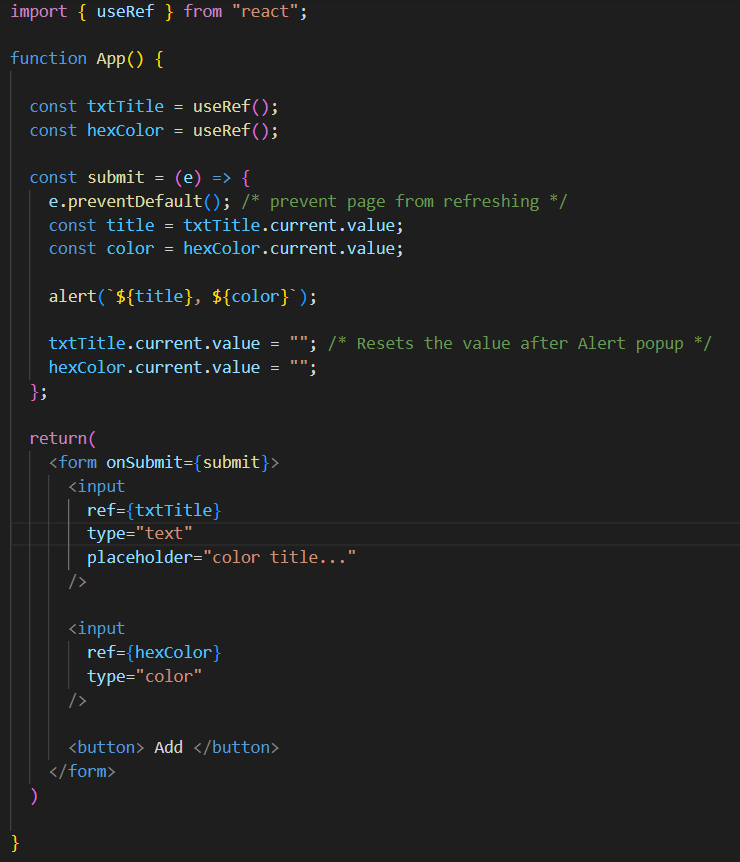
* useReducer is going to take in two different arguments, first function that we use to update the state

and second argument is initial state

# **Handling forms in React**

## **Working with uncontrolled components**

* We use useRef hook. A Ref or Reference is way that we can visit individual element and check what’s its value is



* unlike useState where the component will re-render if there's some sort of a change, useRef is not going to re-render. We always are going to have to reach out to the current value, the current dot value to figure out what that value actually is by reaching out to the input itself.

## **Creating controlled form elements**

* It’s also possible to manage forms using State.
* Whenever we see a state value being used with a form. We want to think about this as being a controlled component, controlled meaning that we're controlling our form by creating state values for these form elements.

## **Building custom hooks**

* A custom hook is a function and it is always going to start with keyword “use”

## **Choosing a form library**

* Formic.org
* React-hook-form.org

# **Asynchronous React**

## **Asynchronous React Fetching data with hooks**

* How to fetch data from some sort of external API
* In following example, we are using useState to handle the data, useEffect to make call for that external data

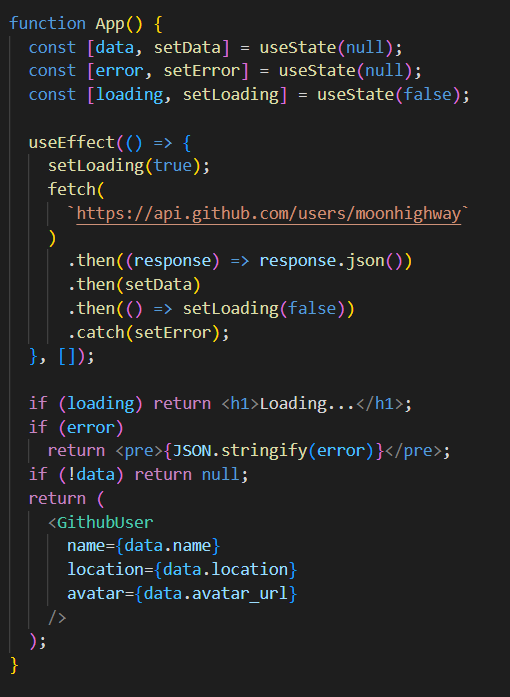


## **Displaying data from API**



## **Handling loading states**

* When we're fetching data from some sort of external API, our data can be in one of a few different states.
* First of all, we have a loading state. So, a loading state just means we're fetching the data but it hasn't yet come back.
* We have a success state where we have some data to display.
* Then we want to have some sort of an error state if something goes wrong.



## **Fetching data with GraphQL**

* GraphQL is a way of creating an API where you can specify what data you want by using its field.
* We're going to make a request to this endpoint, snowtooth.moonhighway.com.

## **Working with render props**

* How we use function to display right data at right time

# **Asynchronous React**

## **Installing React Router v6**

* Now, with React, we're dealing with different components. So how do we move from component to component when we're dealing with a single-age application?
* The solution for that is to incorporate React Router. React Router will allow for us to seamlessly move from page to page by displaying different components.
* Use command **npm install react-router-dom@6**

## **Configure the Router**

Changes in index.js file to configure the Router



## **Incorporating the Link component**

At first import a component from React Router called Link

## **Nesting links with React Router V6**

# **React Testing and Deployment**

## **Using Create React App as a testing platform**