- To create a project with a specific version of React, first we have to create the project with,

```
npx create-react-app <project-name>
```

- Once the project is created, we have to go in that directory with "**cd <project-name>**".

- Here we can install the specific version of react & react-dom by "**npm install**" command - & provide a specific version right-after the package name with the "**@**" character.

```
npm install react@<specific-react-version>
react-dom@<specific-react-version>
```

- If we want to install the "**16.14.0**" version then we have to run the command,

```
npm install react@16.14.0 react-dom@16.14.0
```

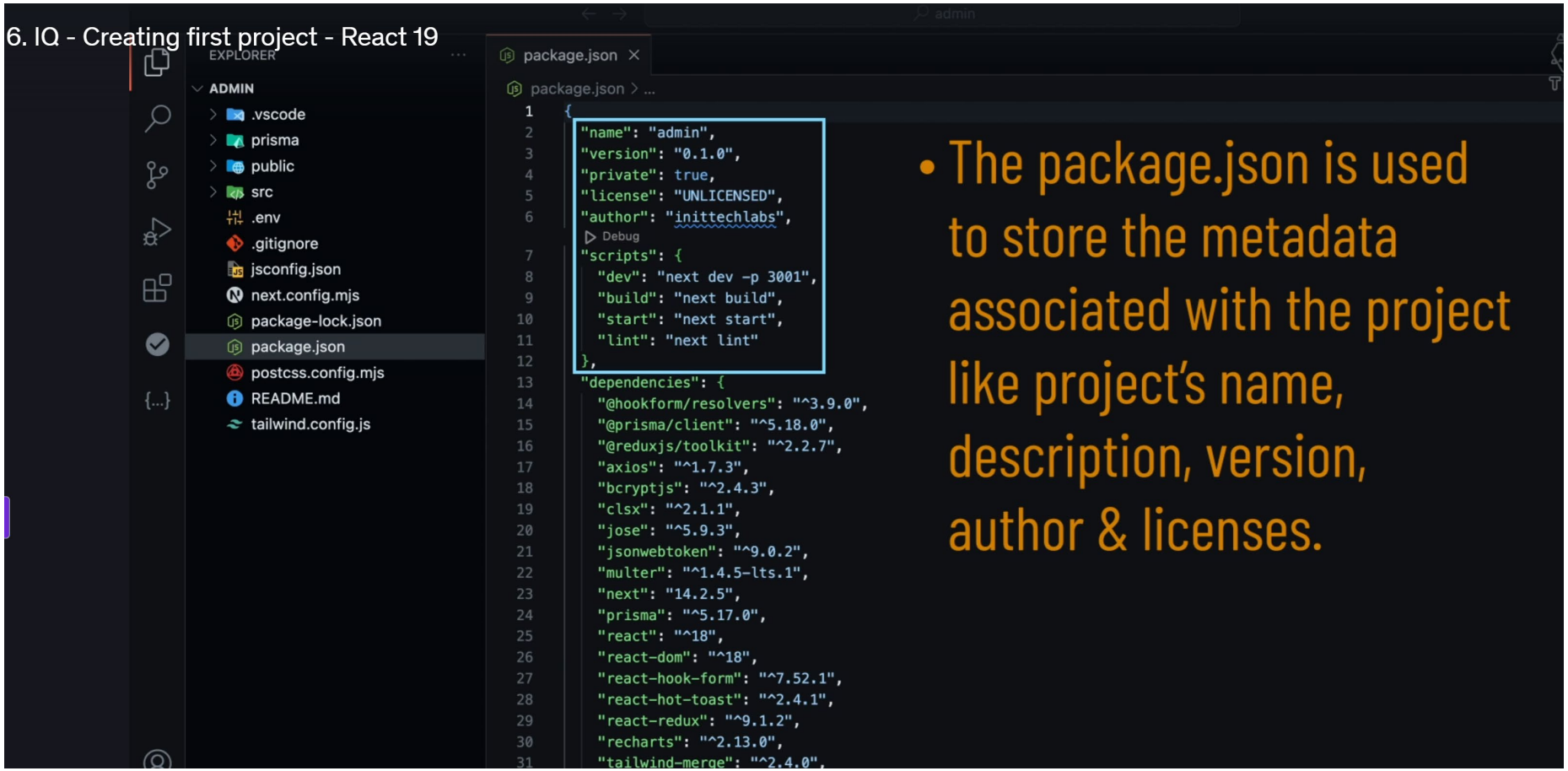
- We cannot create an RC project directly.
- First we need to create a project in react by running

  ```
  npx create-react-app <project-name>
  ```

  which will create a project with the latest React Version.
- Then inside the project directory, we have to install the RC version of react & react-dom separately.
- In the terminal we have to run following command -

  ```
  npm install --save-exact react@rc react-dom@rc
  ```

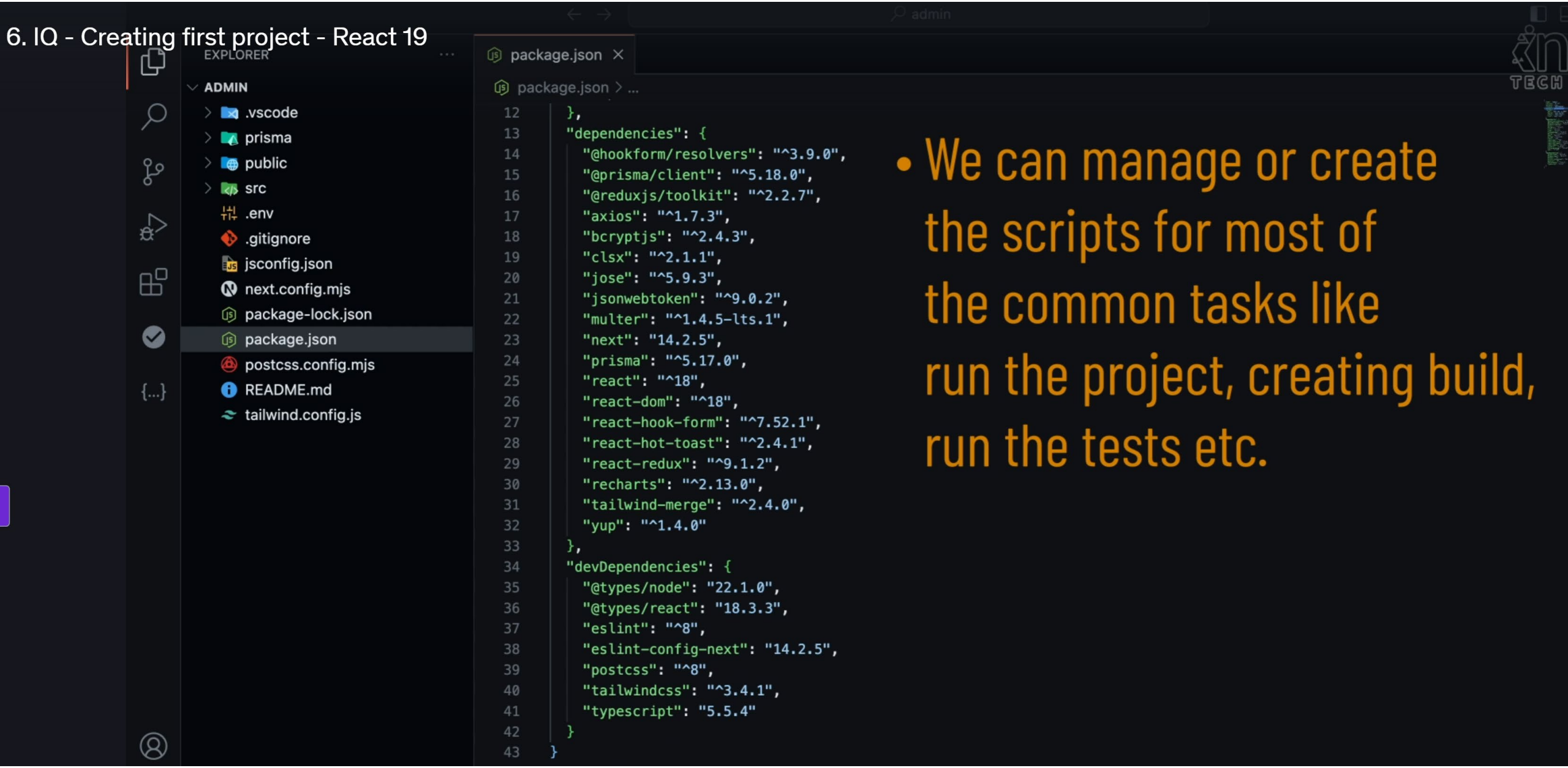
6. IQ - Creating first project - React 19



6. IQ - Creating first project - React 19

- It also manages the list of all the dependencies as well as libraries used in the project along with their specific versions.

6. IQ - Creating first project - React 19



- We can manage or create the scripts for most of the common tasks like run the project, creating build, run the tests etc.

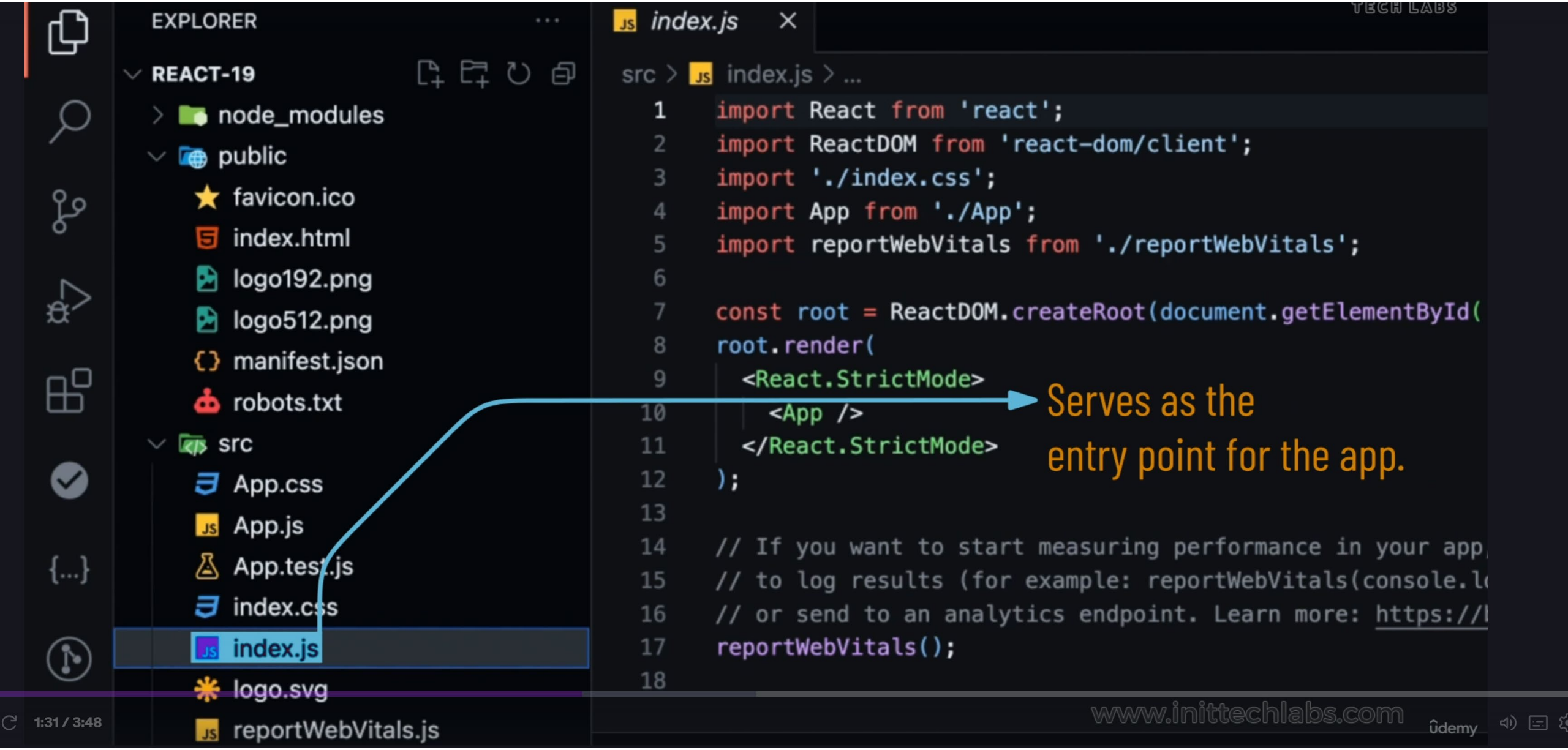
6. IQ - Creating first project - React 19

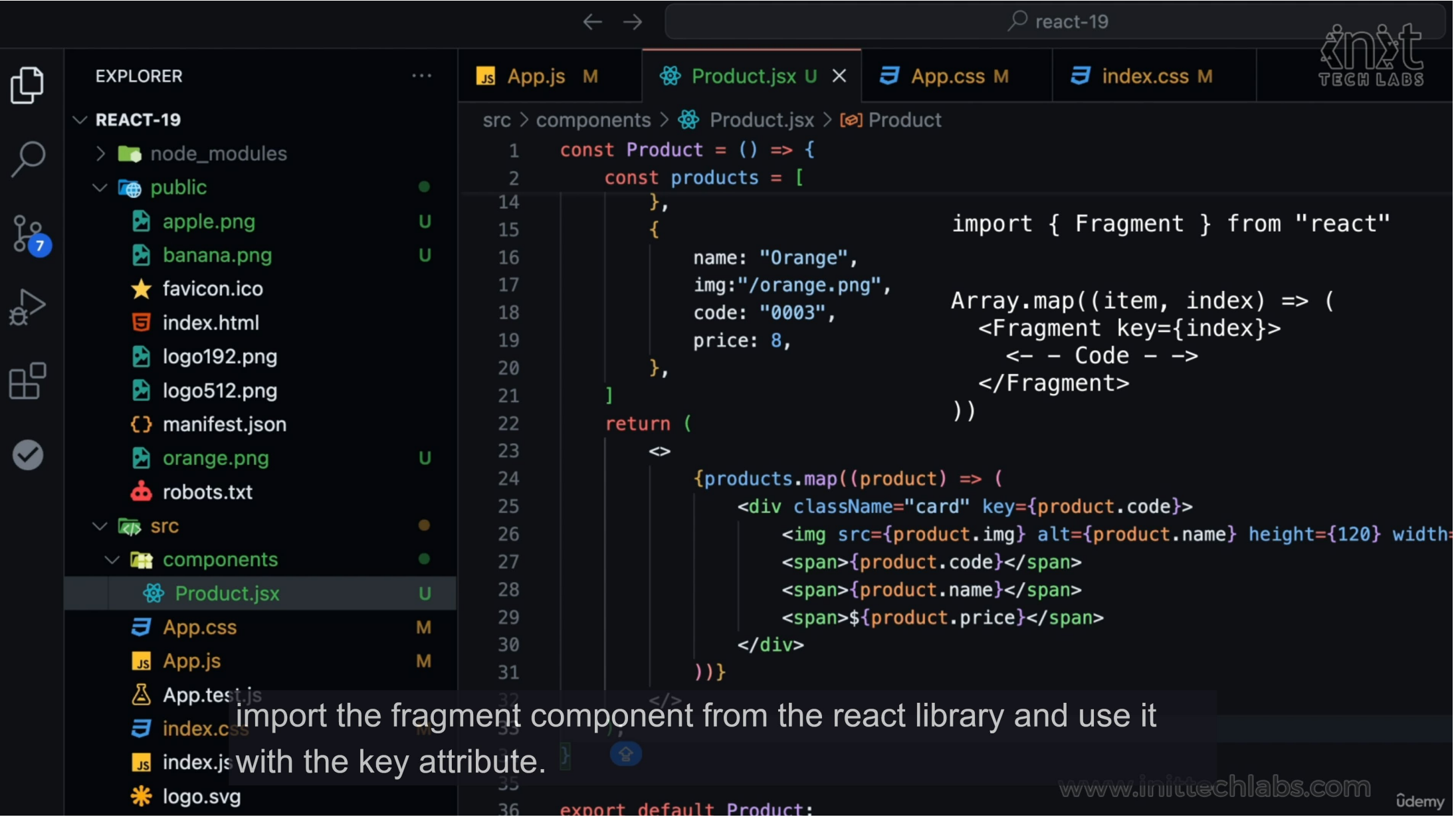
- It has “devDependencies” section in which we can add those dependencies which are used only for development & testing purposes.
And it’s not required on the production i.e., live website.

- The major role of the "**package-lock.json**" file is to ensure that libraries & packages are installed consistently across different environments.
- For that, it locks the exact version of the dependencies & sub-dependencies - which are used in the project.
- So that it prevents the unexpected version updates of the dependencies.

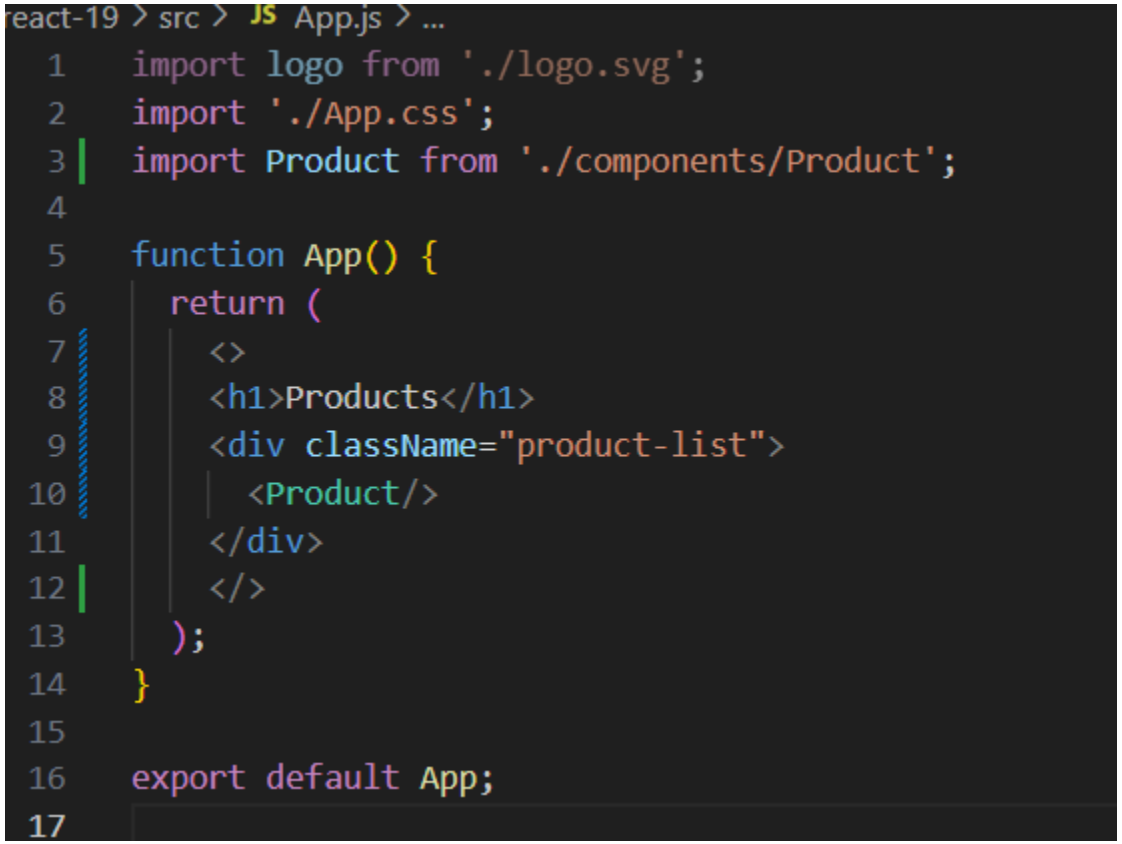
- Deleting the package-lock.json file will not immediately break/harm the project.
- It'll be regenerated when we run the "**npm install**" command.
- In some cases it is possible that versions might vary from the previous lock file.
- Due to that, we might face unpredictable behavior in the project.

- The best practice is never remove the package-lock.json file manually.
- By doing so it may harm the project in some of the cases.


EXPLORER
REACT-19
node_modules
public
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
src
App.css
App.js
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
index.js
src > index.js > ...
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import './index.css';
4 import App from './App';
5 import reportWebVitals from './reportWebVitals';
6
7 const root = ReactDOM.createRoot(document.getElementById(
8 root.render(
9 <React.StrictMode>
10 <App />
11 </React.StrictMode>
12);
13
14 // If you want to start measuring performance in your app,
15 // to log results (for example: reportWebVitals(console.l
16 // or send to an analytics endpoint. Learn more: https://
17 reportWebVitals();
18
Serves as the
entry point for the app.
1:31 / 3:48
www.inittechlabs.com
udemy



Se for necessário usar fragments como parente de um array, teremos que importar o objeto Fragment do React.



Ser não for pai de array pode ser sem importação

- We cannot use the empty fragment as a parent container in the "Array.map()" method. Because we cannot assign a unique key to the Empty Fragments (<>...</>).
- The keys or attributes will be given to a named element only - & the Empty fragments is an element with no name.
- If we want to use fragments as a parent container of the "Array.map()" method, then we have to explicitly import the "Fragment" component from the "react" library. & use it with the key attribute.

import the fragment component from the react library and use it with the key attribute.

component.js

```
export default function
Button() {
...
}
```

one default export

components.js

```
export function
Slider() {
...
}
```

```
export function
Checkbox() {
...
}
```

multiple named export

mixedcomponents.js

```
export function
Avatar() {
...
}
```

```
export default function
FriendsList() {
...
}
```

named export(s) &
one default export

With the named imports, the name has to match with what we have used while exporting it & it should be in curly brackets in the import statement.

```
import defaultComp, { namedComp1, namedComp2 }
```

- In ES6, there are two primary ways to import or export functions or modules: "**default**" & "**named**" import/export.

component.js

```
export default function
Button() {
...
}
```

one default export

components.js

```
export function
Slider() {
...
}
```

```
export function
Checkbox() {
...
}
```

multiple named export

mixedcomponents.js

```
export function
Avatar() {
...
}
```

```
export default function
FriendsList() {
...
}
```

named export(s) &
one default export

- To export the component as default export, we have to use the "**default**" keyword in the export statement.

```
export default <variable or component>
```

- Whereas in the named exports we just have to write the "**export**" keyword just before the declaration of a module or function.

```
export const variable = "" or
export function abc()
```

- To import the default exported component, we just have to use the import statement along with any name.

```
import ComponentName from "./file.jsx;
```

- Whereas while importing named components or modules - we have to use the same name used during declaration.

```
import { Name } from "./file.jsx
```

- Developers often use default exports if the file exports only one component & use named exports if it exports multiple components or modules.

Rules for using the JSX in React

- We can return only one parent element from the component, it is not possible to return multiple parent components. So, we have to wrap them with a single parent container - for which ideally we use a or the Fragments.
- If we use the then it can create unnecessary nodes to the DOM, so we prefer to use the Fragments where needed.

Rules for using the JSX in React

Why do multiple JSX tags need to be wrapped?

- The JSX is transformed into plain JavaScript objects under the hood.
- We cannot return multiple objects from a function without wrapping them into an array.

Rules for using the JSX in React

- JSX requires tags to be explicitly closed - e.g., self closing tags like <img> must be used as <img /> & wrapping the elements or texts with the opening & closing tags like - ... .

Rules for using the JSX in React

- We have to use the camelCase for declaring variables & the attributes of the elements.
- JSX turns into JavaScript & attributes written in JSX become keys of JavaScript objects. However JavaScript has limitations on variable names.
- In React, many HTML attributes are written in camelCase, e.g., - instead of “stroke-width” we use “strokeWidth”, same for the “html-for” attribute is written as “htmlFor” etc.

Rules for using the JSX in React

- Also by using the camelCase in declaring variables or attributes, it increases the code readability & helps the compiler to differentiate between the JavaScript keywords with the HTML attributes or custom variables.

- It is not possible to return multiple parent components, so, we have to wrap them with a parent container - which can be either "**div**" or we can use the **Fragments**.
- JSX requires tags to be explicitly closed - e.g., self closing tags like <img> must be used as <img /> & wrapping the elements or texts with the opening & closing tags like - ... .

- We have to use the camelCase for declaring variables & the attributes of the elements.

- The JSX is transformed into plain JavaScript objects under the hood.
- So we cannot return multiple objects from a function without wrapping them into an array.
- That is the reason we cannot return the multiple elements or JSX tags without wrapping them with the parent tag.

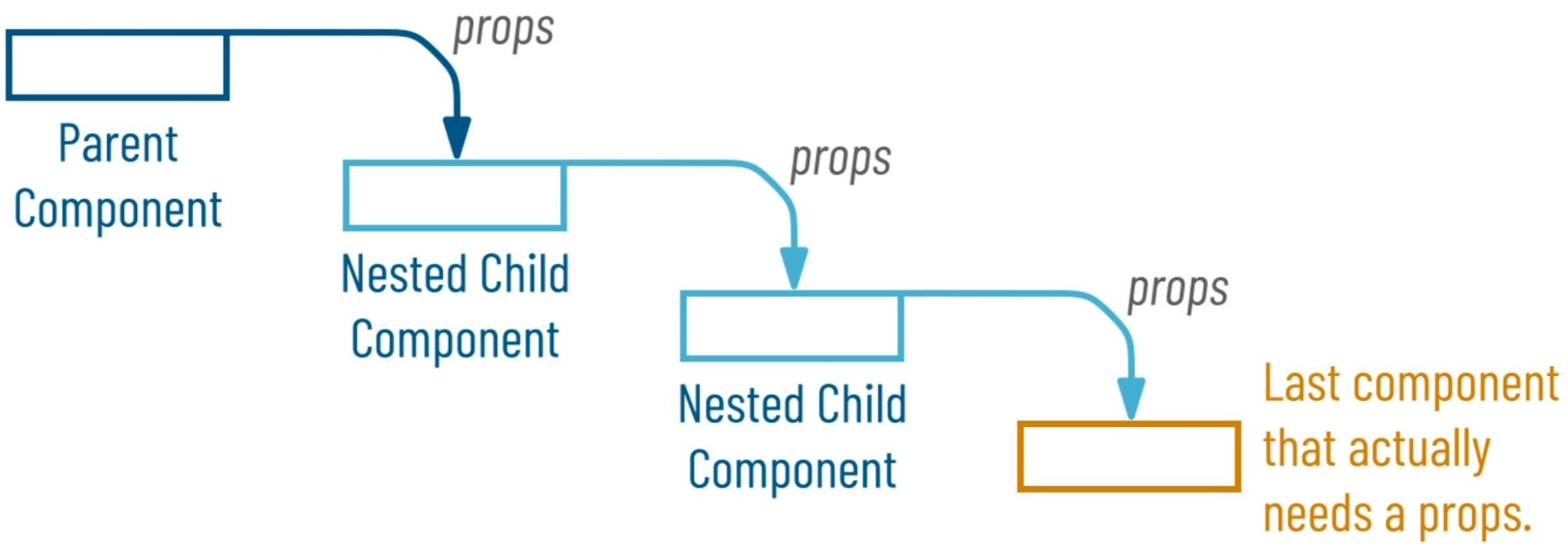
- We can render multiple elements by wrapping them with one parent element.
- And that will be assigned to a variable using parenthesis.

```
let a = (
    <div>
        <h1>Hello</h1>
        <h1>World</h1>
    </div>
);
```

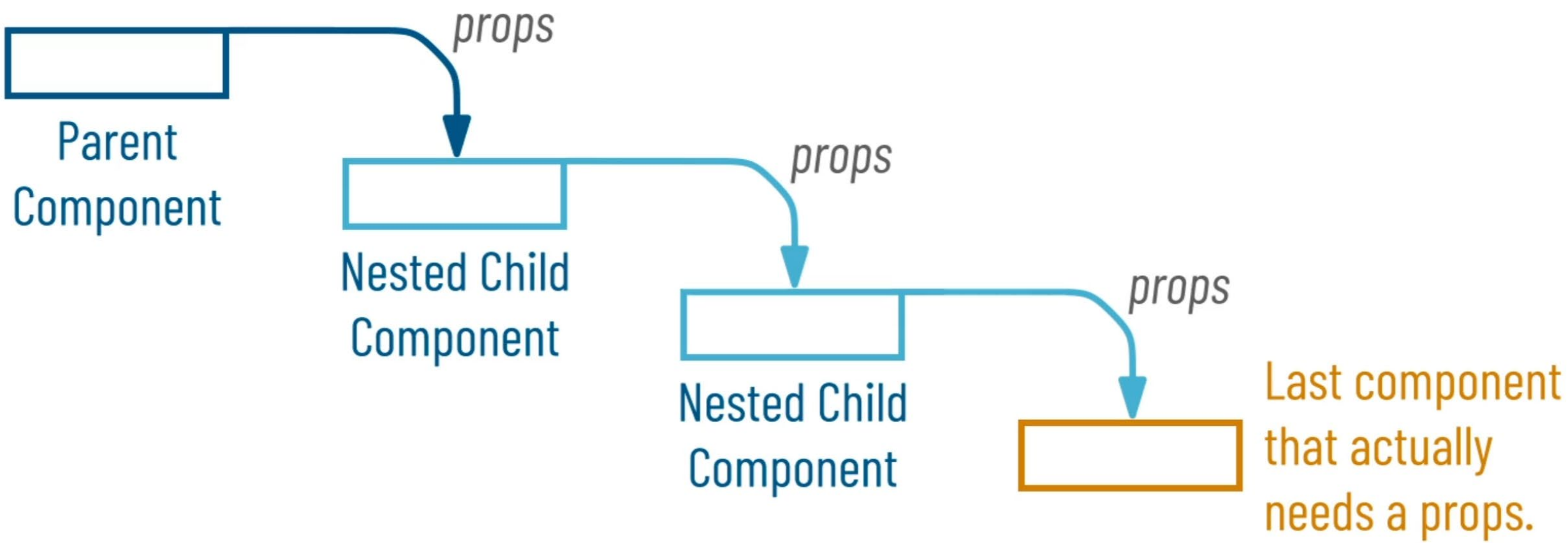
- To render it, we just write “{a}” in the returned JSX. And that will do the job.

- Props are similar to the parameters used in functions.
  Or in other words - props are the parameters of components.
- React uses the Props concept - in which we can pass
  the data from one component to another component.
- In most of the cases the props will be passed
  from the parent component to the child component.
- Here, we can pass all types of data using props like -
  strings, numbers, objects, arrays, functions, variables etc.

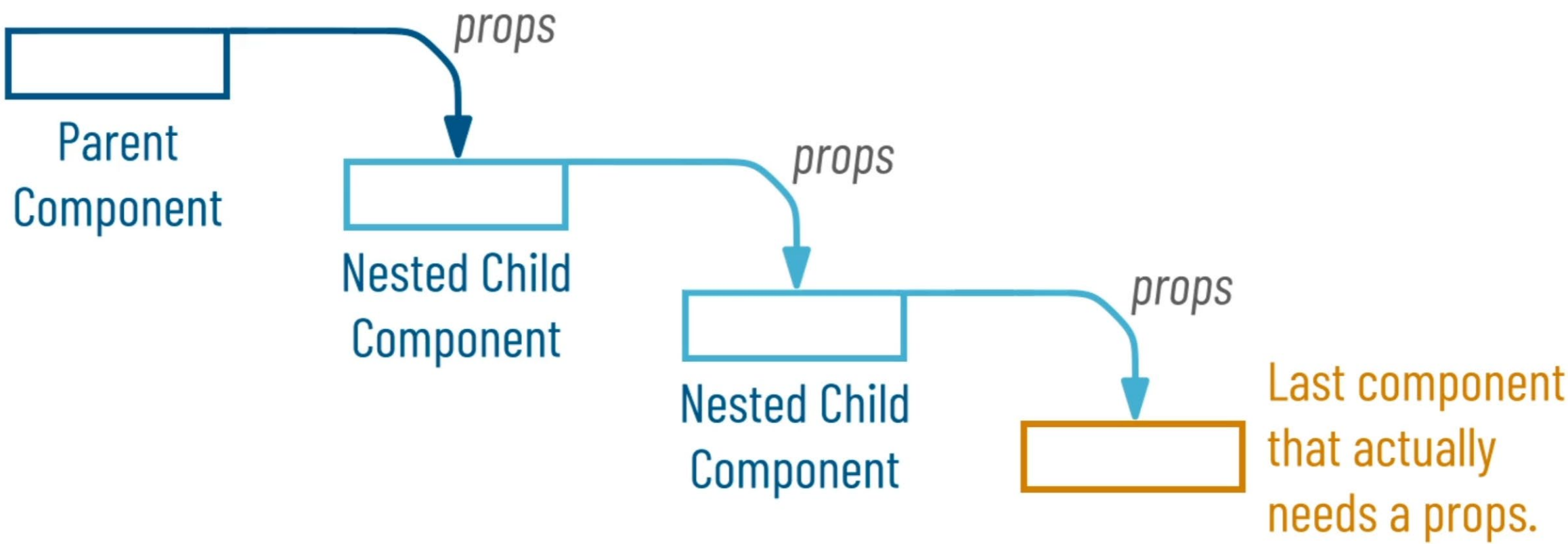
- The process of passing data or props from a parent component to one or more child components is called Prop Drilling.



- It occurs when a prop needs to be passed through several layers of child components to reach a deeply nested child component that actually needs the data or prop.



- In that process, each intermediary component in the hierarchy has to pass the props to another component, even if it does not use that prop itself.



- So, first we have to create a function which should be called on the button click.

```
const handleClick = () => { ... }
```

- Then we assign this function to the “onClick” property.

```
<button onClick={handleClick}>Click Me</button>
```

- So now when the button is clicked, the event will be triggered & the handleClick function will be called.

- If we use the function with parentheses directly in the onClick property - then the compiler calls the function when rendering the JSX.
- So this function gets called without button click which is an unexpected behaviour.

- To pass the parameters to the function, we need to return the function itself. And we can do it by using the arrow function inside the "**onClick**" property.

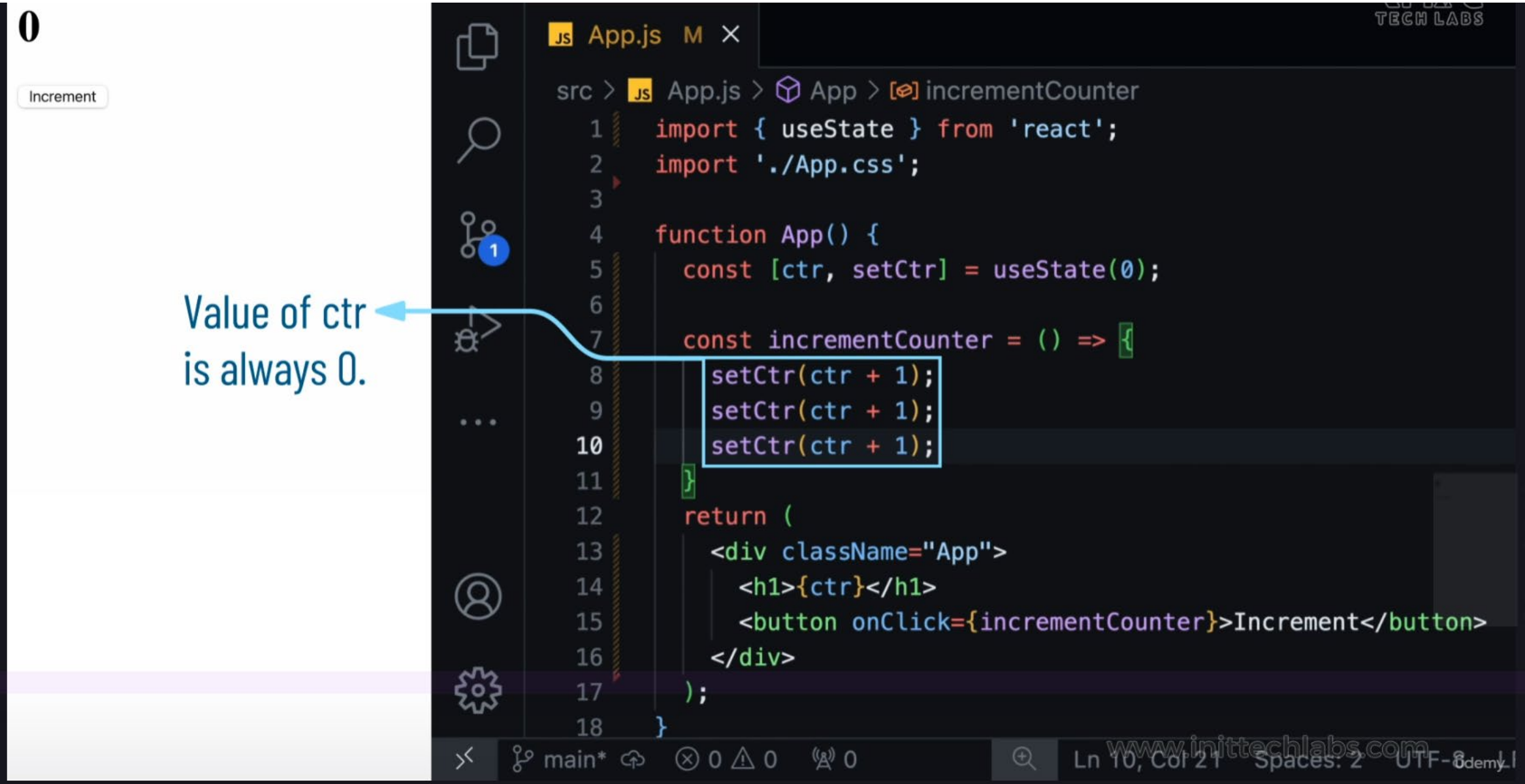
```
<button onClick={() => handleClick("Hello World")}>
Click Me</button>
```

- By using the arrow function we can avoid the function getting called while rendering the JSX.

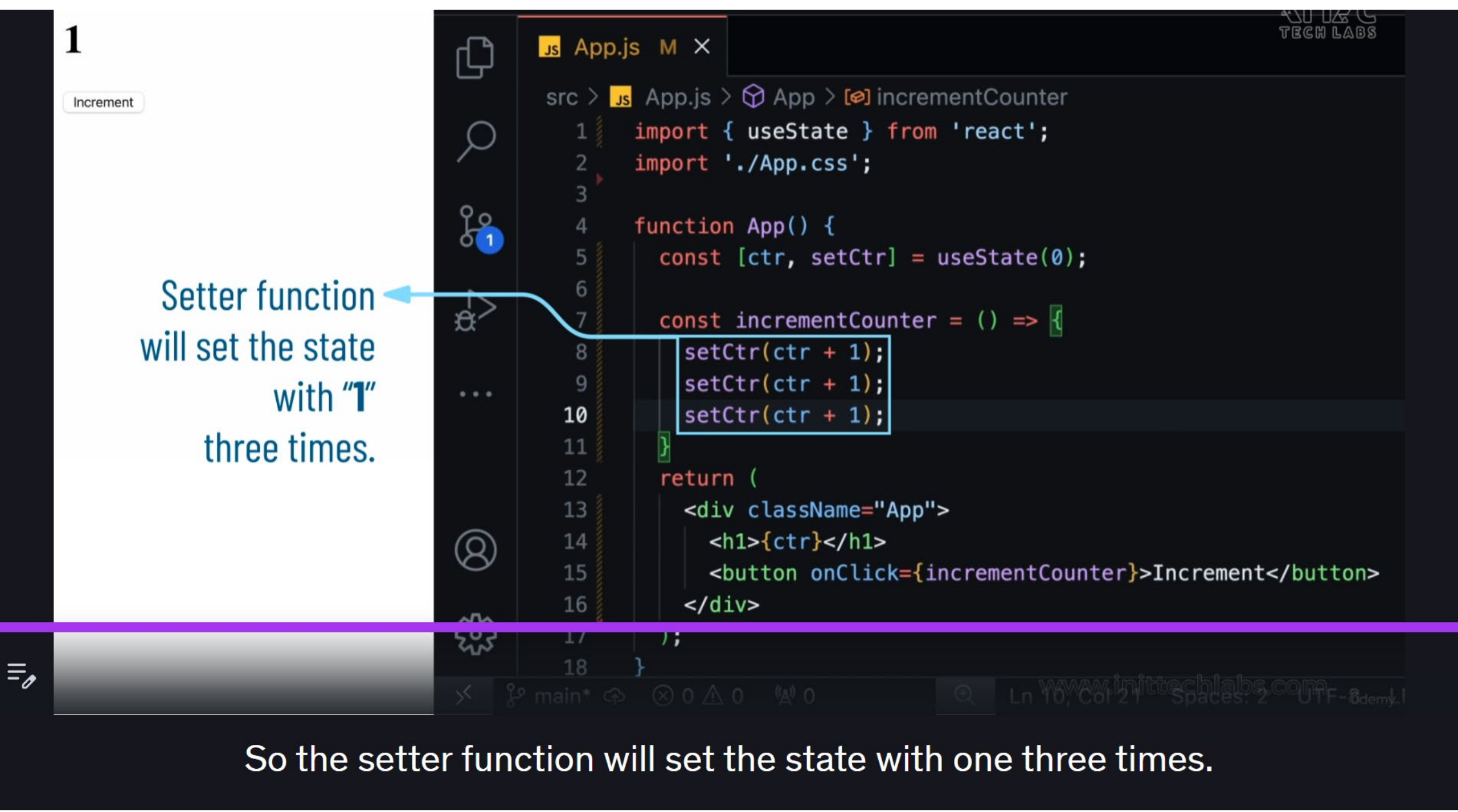
- A normal variable which is not a state is treated internally for JavaScript but as long as React's DOM manipulation is concerned, it will not work.
- React watches for states only.
- The rendering of the component will not be called in case of change to a regular variable.

```
const sdArr = useState(10);
//or
let sdArr = useState(10);
i=sdArr[0];
setCtr=sdArr[1];
```

React re-renders the component wherever the state is updated. Any change made to a regular variable will not trigger rendering of a component.

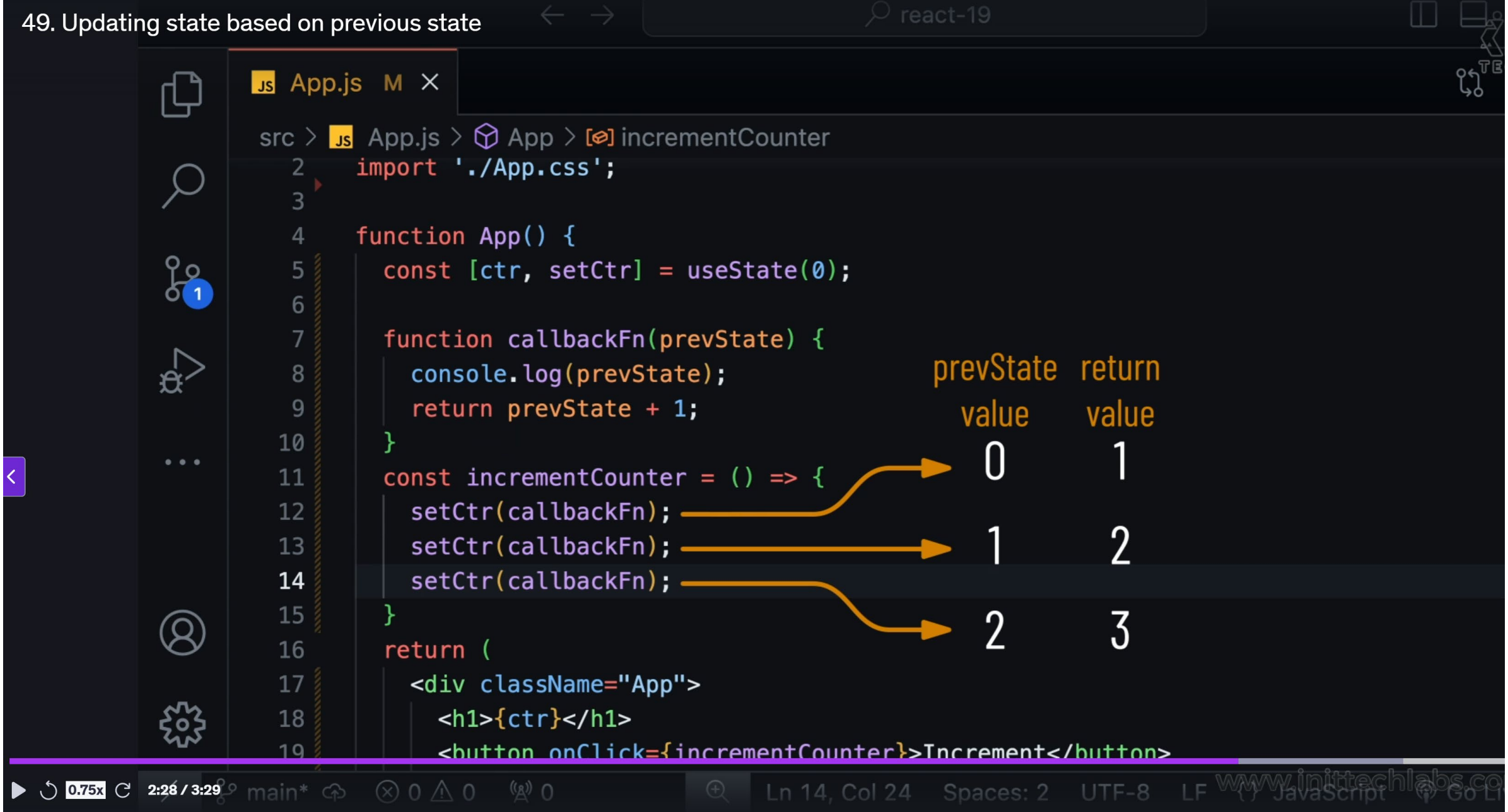


The value of CTR is always zero.



So the setter function will set the state with one three times.

49. Updating state based on previous state



So after calling all the setter functions, the state value will be three.

- If we want to update the state which is calculated from the previous state, then we can use the updater function.
- There is no harm in using it, but it is also not always necessary.
- In most of the cases, there is no difference between the setter method with updater function & the regular setter method.

- React Always make sure that intentional user actions, like clicks - the state variable would be updated before the next click.
- If we do multiple updates using the setter function within the same event, then the updater functions can be helpful for setting the state.