Understanding Props and PropTypes in React | Complete React Course in Hindi #6

In the last tutorial, we edited the TextUtils app using bootstrap. Let's run our application using the *npm start* command and continue our React journey.

Props:

Props stand for Properties. It is used to Pass information into the React Components. For example, if we create a custom Component, then by using Props, we can pass data into it in the form of Objects, strings, numbers, etc.
Earlier, we have installed the *'ES7React/redux/GraphQL/React-native snippets'*. This is a very useful extension while working in React, as it provides a shortcut way to write code. For example: If you type 'imp' then it will print:

```
import moduleName from 'module'
```

You can check out the snippet info of this extension by clicking here. There are many extensions similar to this extension. For example, Simple React snippets.

Named export and Default Export:

Before learning about Props and Prop-types, let's learn the Named export and Default Export:

1. Default Export: In this case, if import occurs by any name, Suppose XYZ, then the default value will be exported. It is generally used to export a single object, function, or variable. Let's understand with an example-

Create two files. We are creating module.mjs and modue2.mjs. We will be performing a default export from module.mjs to module2.mjs

In module.mjs:

```
const a = 'Harry';
const b = 'Rohan';
const c = 'Aakash';
const d = 'Lovish';
const e = 'Priyanka';
export default b;
```

This is an example of a Default export, Now let's import it.

In module2.mjs:

```
import XYZ from './module.js'
console.log(XYZ);
```

The default exported value, which is 'Rohan', will be imported. In default Export, the name of Import is Independent of the name of Export.

2. Named Export: In this case, we export some specific values. The name of import is dependent on the name of Export which means You can't use a different name in the Import and Export function. We use Curly brackets for Named export. For example: performing the same export using named export.

In module.mjs:

```
const a = 'Harry';
const b = 'Rohan';
const c = 'Aakash';
const d = 'Lovish';
const e = 'Priyanka';
export {b};
```

In module2.mjs:

```
import {b} from './module.js'
console.log(b);
```

The value of 'b' will be imported, which is 'Rohan'. Name Exports are used to export multiple values.

Creating Navbar Component:

Create a components folder and create Navbar.js, which is the Navbar component of our React app.

In navbar.js:

To generate the react Function-based component code, simply use the *'rfc'* snippet and enter. The below code will be generated:

```
import react from 'react'
export default function Navbar() {
    return (
        <div>
            //Code of your Navbar
        </div>)
}
```

Write the code of your desired navbar inside the div tag. Now we have created our navbar, but we have to import it to app.js to serve it.

In app.js:

We will use the Navbar component by writing <Navbar/> inside return(). Navbar Component will be automatically imported in App.js.

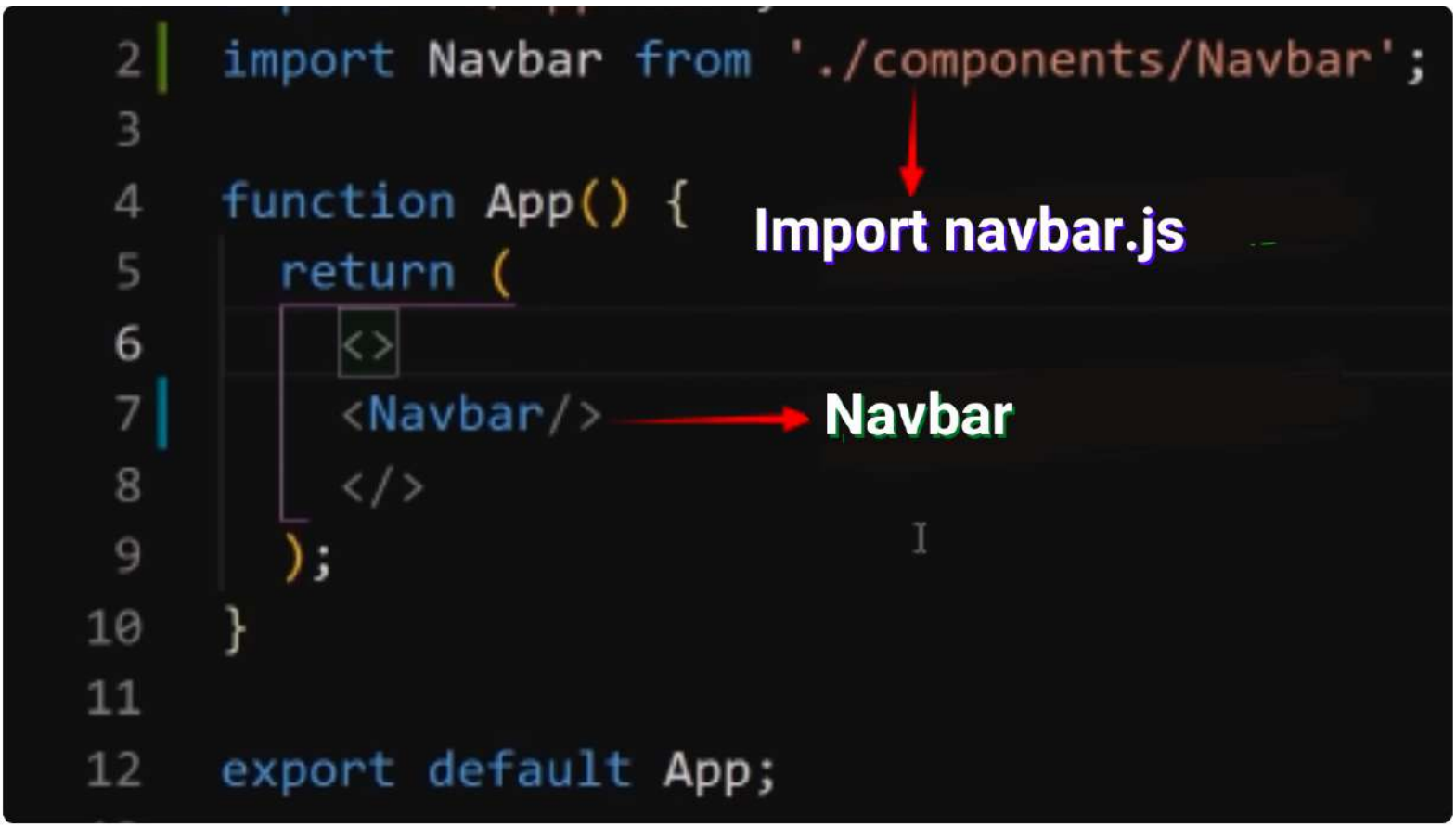
import Navbar from './components/Navbar';
function App() {
return (
<>
<Navbar/>
</>
);
}
export default App;
Import navbar.js
Navbar

Figure1.1: Imported Navbar in App.js

Your Navbar will be displayed in your application.
You might be wondering, Why import it as a component? Can't we write the whole navbar code in App.js? The answer is Yes, you can. But then, you have to write your complete code in app.js inside the return() function, and hence organizing your large set of code will be a nightmare. Here are some advantages of structuring your app in Component.

Advantages of Component:

- By dividing your app into components you can reuse the component in the same or different app again and again.
- It also makes it easier to find errors in a large set of your code.

So now on, we will be creating components of our react app and will be importing them in app.js.

Changing Values Using Props:

Now let's Suppose you want to use the above navbar in your different applications but with different titles and About, or pass New values to your existing Navbar. You can do so by using Props in React.

In navbar.js:

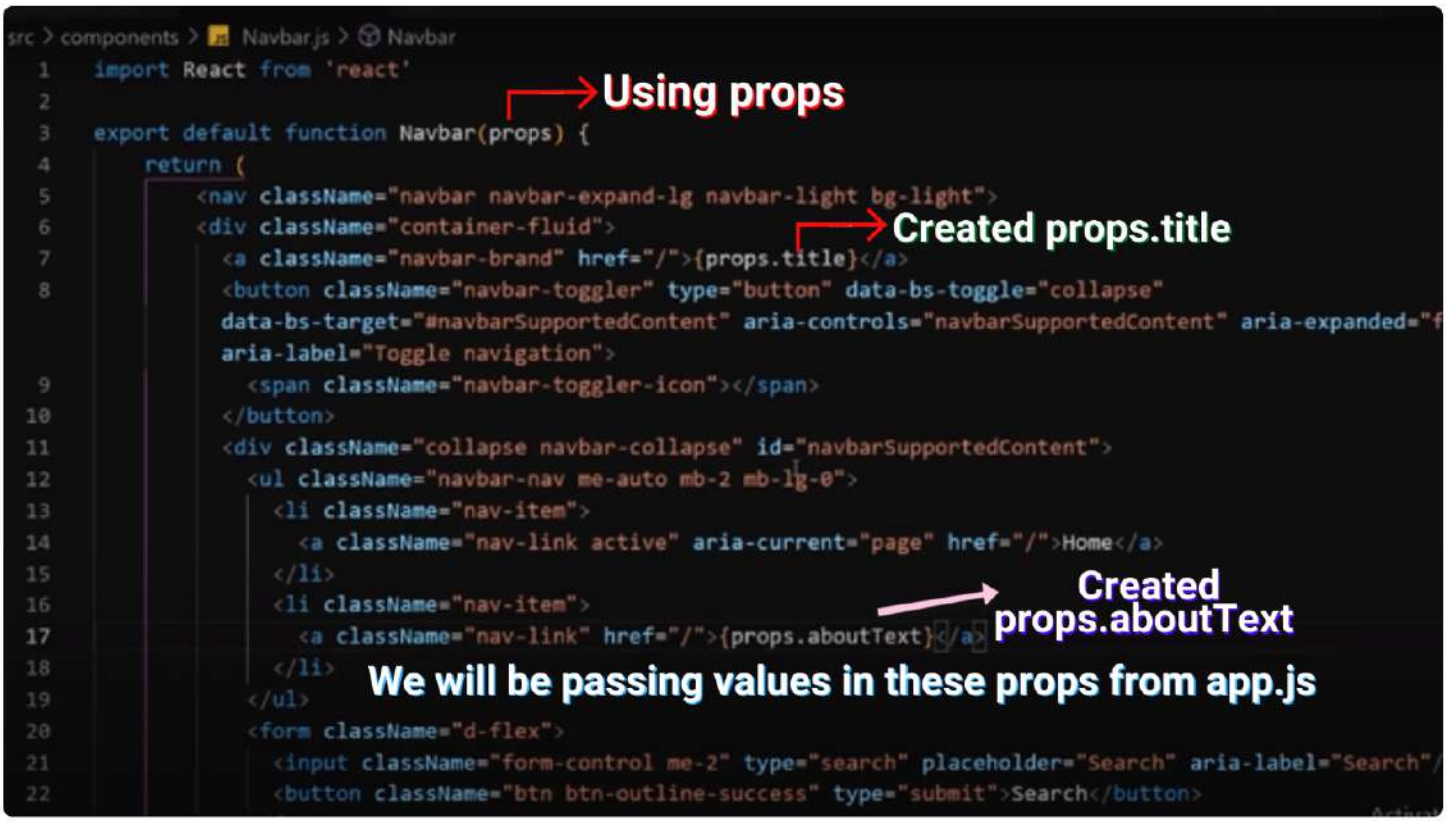

Figure1.2: Creating Props

Here, we are passing props in our Navbar component and have created two props:

- props.title
- props.aboutText

Now we will be passing values in these props.

In app.js:

Passing values: You can pass the values in the created props as shown below:

```
function app() {
    return (
        <>
            <navbar title="Textutils" aboutText="About Textutils" />
        </>
    );
}
```

Here, we have passed the 'Textutils' in props.title and 'About Textutils' in props.aboutText.

Now we have created such a React Navbar component, whose title and About can be easily changed by using props. Remember, we can even pass objects or links, etc., in our component. You should never directly change the value of props.

Prop-type:

Firstly, we need to import prop types to do so, use this command

```
import PropTypes from 'prop-types'
```

1. String/object/Number Prop-Type

```
navbar.propTypes = {title: propTypes.string.isrequired, aboutText: PropTypes.string};
```

It means that my prop-type of title is a string which means on passing any other value, like Number, it will show an error in the console. Hence we can only pass a string in props.title and props.aboutText. We can use *'isrequired'* keyword, which makes sure that we won't leave that prop blank. If you do so, it will show an error in the console. Similarly, you can use an object, Number, etc. prop-type as well.

2. Default Prop-Type

```
navbar.defaultProps = {
    title: 'Set title here',
    aboutText: 'About text here'
};
```

If no props are being passed in the component, then the Default content in this prop-type will be displayed.