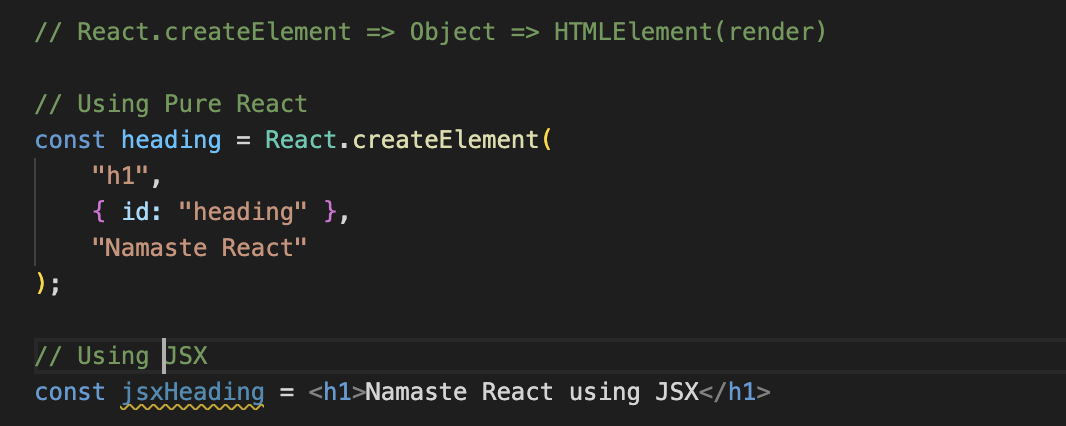
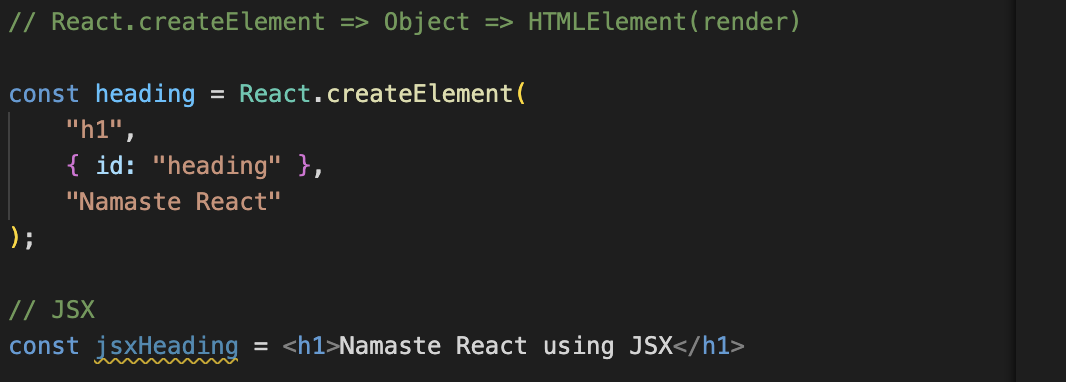
● What is JSX?

1. JSX is HTML-like or XML-like syntax. JSX stands for JavaScript XML. It's a syntax extension for JavaScript.
2. It is not a part of React. React apps can be built even without JSX but the code will become very hard to read.
3. It is not HTML inside JavaScript.
4. JavaScript engine cannot understand JSX as it only understands ECMAScript  
     
     
     
   
5. When we log *heading* and *jsxHeading*, it gives the same object.

● Superpowers of JSX

1. Sanitizes the data: -
   1. If someone gets access to your JS code and sends some malicious data which will then get displayed on the screen, that attack is called cross-site scripting.
   2. It can read cookies, local storage, session storage, get cookies, get info about your device, and read data. JSX takes care of your data.
   3. If some API passes some malicious data JSX will escape it. It  
      prevents cross-site scripting and sanitizes the data before  
      rendering
2. Makes code readable: -
   1. JSX makes it easier to write code as we are no longer creating  
      elements using React.createElement()
3. Makes code simple and elegant
4. Show more useful errors and warnings
5. JSX prevents code injections (attacks)

● Role of type attribute in script tag? What options can I use there?

The type attribute in the <script> tag specifies the scripting language of the code inside the tag or the external script file referenced by the src attribute. While it is often omitted for standard JavaScript, it still plays a significant role in some scenarios.

**Common Options for the type Attribute:**

1. **text/javascript**: (Default) Standard JavaScript. This can usually be omitted.
2. **module**: For JavaScript modules, enabling ES6+ features like import and export.
3. **application/json**: For embedding JSON data within the script tag, preventing it from being executed.
4. **text/babel**: Used when writing JSX or modern JavaScript syntax that needs to be transpiled by Babel (requires Babel setup).
5. **text/vbscript**: For embedding VBScript (mostly obsolete now).

● {TitleComponent} vs {<TitleComponent />} vs {<TitleComponent></TitleComponent> } in JSX

In JSX, the syntax you use to include components can vary, and each variation has a specific meaning and use case. Let's break down the differences between {TitleComponent}, {<TitleComponent />}, and {<TitleComponent></TitleComponent>}:

**1. {TitleComponent}**

* **Meaning**: This syntax refers to the component itself as a reference, not as an instance of the component.
* **Use Case**: You use {TitleComponent} when you want to pass the component itself as a prop to another component, store it in a variable, or conditionally render it later.

**2. {<TitleComponent />}**

* **Meaning**: This is the shorthand syntax for rendering a component with no children. It creates an instance of TitleComponent and renders it immediately.
* **Use Case**: Use this when you want to render the TitleComponent without any children or additional content inside it.

**3. {<TitleComponent></TitleComponent>}**

* **Meaning**: This is the full syntax for rendering a component with children or additional content. It creates an instance of TitleComponent and can include children elements or other components between the opening and closing tags.
* **Use Case**: Use this when you need to nest content inside the TitleComponent or when you prefer a more explicit syntax, even if there are no children.

**Summary:**

* **{TitleComponent}**: Refers to the component itself as a reference. It's not rendering anything, just holding the component as a value.
* **{<TitleComponent />}**: Renders the component with no children. It's a self-closing tag, used for simple components.
* **{<TitleComponent></TitleComponent>}**: Renders the component and can include children or nested content. It's more explicit and flexible for components that might have inner content.