Lesson Plan

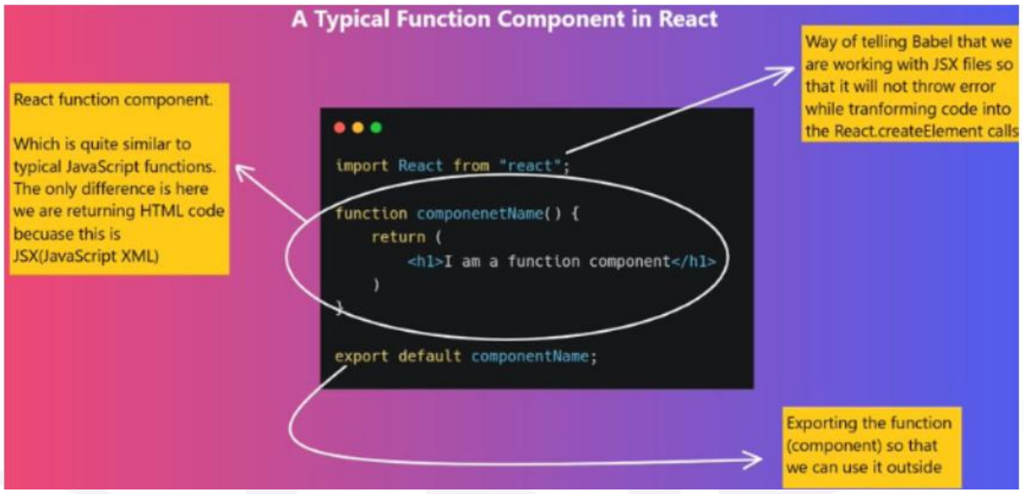
Functional component

</>
CSS
<HTML>
<HEAD>
<TITLE> Lorem ipsum </TITLE>
</HEAD>
<BODY>
<H1> Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam</H1>
<P> <IMG SRC= "Lorem ipsum"></P>
HTML

Topics

- Introduction to functional component
- Syntax and structure of functional component
- Introduction to react fragment

Introduction to functional component


A Typical Function Component in React
React function component.
Which is quite similar to typical JavaScript functions. The only difference is here we are returning HTML code becuase this is JSX(JavaScript XML)
Way of telling Babel that we are working with JSX files so that it will not throw error while tranforming code into the React.createElement calls
Exporting the function (component) so that we can use it outside

```
import React from "react";

function componenetName() {
    return (
        <h1>I am a function component</h1>
    )
}

export default componentName;
```

Functional components in React are a fundamental building block for creating user interfaces. Unlike class components, functional components are defined as JavaScript functions and are a key feature introduced with the advent of React Hooks.

In a functional component, you declare a function that takes in props as its argument and returns React elements to render. With the introduction of hooks, functional components can now manage state and lifecycle events, making them more powerful and versatile.

One of the significant advantages of functional components is their simplicity and readability. They offer a concise syntax and encourage the use of functional programming principles. Additionally, functional components are easier to test and optimize, contributing to a more maintainable and scalable codebase.

React Hooks, such as useState and useEffect, enable functional components to manage local component states and handle side effects, previously only achievable with class components. This shift toward functional components and hooks has become a standard practice in modern React development, promoting cleaner and more modular code.

Syntax and structure of functional component

The syntax and structure of a functional component in React are straightforward. Here's a basic example:

MyfunctionalComponent.js

```
import React from 'react';

// Functional Component
const MyFunctionalComponent = (props) => {
  // Component logic and JSX here
  return (
    <div>
      <h1>Hello, {props.name}!</h1>

  <p>This is a functional component.</p>
    </div>
  );
};
export default MyFunctionalComponent;
```

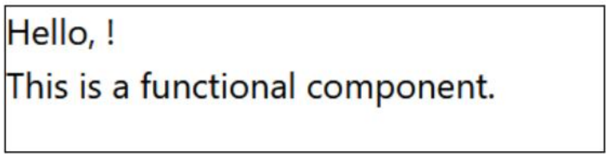
App.js

```
import React from 'react';

import MyFunctionalComponent from "./components/
Myfunctionalcomponent";

function App() {
  return (
    <div className="App">
      <MyFunctionalComponent />
    </div>
  );
}
export default App;
```

Browser output



```
Hello, !
This is a functional component.
```

Import React:

- Import the React module, which is necessary for writing JSX.

Functional Component Declaration:

- Declare a functional component using the const keyword.
- Provide a component name (in this example, **MyFunctionalComponent**).
- Use (props) as an argument to access the component's props.

Component Logic and JSX:

- Write the component logic and JSX inside the function body.
- Utilize the props object to access and use the properties passed to the component.

Return JSX:

- Use the return statement to return JSX elements.
- The returned JSX represents the structure and content of the component.

Export the Component:

- Export the functional component at the end of the file, making it available for use in other parts of your application.

Introduction to react fragment

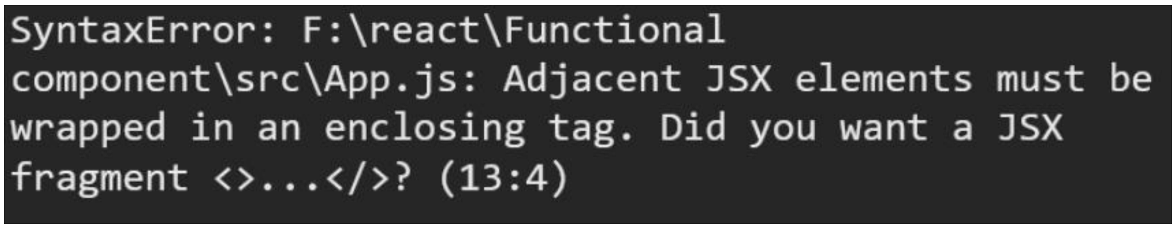
React Fragment is a feature in React that allows you to group multiple children elements without introducing an additional parent element in the DOM. In React, when you want to return multiple elements from a component's render method, you typically need to wrap them in a single parent element. This requirement can be problematic in situations where an extra wrapping element is not desired or disrupts the layout.

Here's an example of what can go wrong when multiple HTML elements are returned without using a Fragment:

```
import React from 'react';

const App = () => {
  return (
      <div>
        <h1>Hello</h1>
        <p>Welcome to my app</p>
      </div>
      <p>This is another paragraph</p>
  );
}
export default App;
```

Browser output



```
SyntaxError: F:\react\Functional
component\src\App.js: Adjacent JSX elements must be
wrapped in an enclosing tag. Did you want a JSX
fragment <>...</>? (13:4)
```

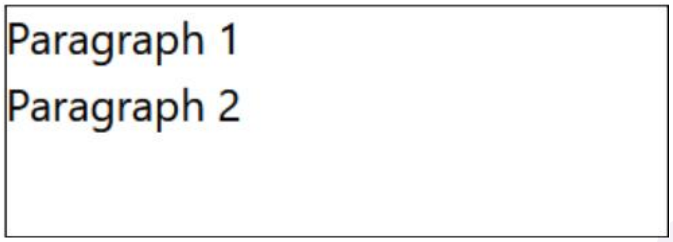
React Fragment provides a solution to this issue by allowing you to group elements without adding an extra node to the DOM. Fragments do not create a new DOM element; they are a lightweight way to group elements and don't produce any visible output themselves

Here's a basic example of using React Fragment:

```
import React from 'react';

const MyComponent = () ⇒ {
  return (
    <>
      <p>Paragraph 1</p>
      <p>Paragraph 2</p>
    </>
  );
};
export default MyComponent;
```

Browser output



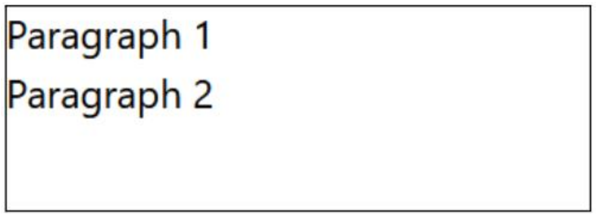
```
Paragraph 1
Paragraph 2
```

In the example above, the <>...</> syntax is a shorthand for <React.Fragment>...</React.Fragment>. It allows you to group the two <p> elements without introducing an additional wrapping element in the DOM.
You can also use the long-form syntax:

```
import React from 'react';

const MyComponent = () ⇒ {
  return (
    <React.Fragment>
      <p>Paragraph 1</p>
      <p>Paragraph 2</p>
    </React.Fragment>
  );
};
export default MyComponent;
```

Browser output



```
Paragraph 1
Paragraph 2
```

Using React Fragment helps to keep your component's output clean and avoids unnecessary nesting in the DOM structure. It's especially useful when returning adjacent JSX elements without introducing an additional container div or span.