

Functional-Based Components

Topics covered:

- What are functional components?
 - Arrow functional components
 - Stateless functional components
- How functional components are different from class components?
- How to use functional components?
 - Syntax of functional components
 - Passing props to functional components
 - State in functional components

1. Functional components:

A functional component is like a **JavaScript function**.

It begins with the term "function", then comes a name enclosed in parenthesis and curly braces.

For Example

PascalCase { 1st Letter Capital Always }

```
Function FunctionName() {  
    Return(  
        //JSX  
        <React.Fragment>  
            <h1>Hello React Users</h1>  
        </React.Fragment>  
    )  
}
```

Functional components did not offer state management before React Hooks' introduction in version 16.5 of React.

Consequently, **stateless components** were referred to as functional components.

With the creation of Hooks, we can essentially perform everything that a class component can do, but faster and with fewer lines of code.

→ why function components are preferred?

A common function-based component contains the following:

- For developing JSX, a React component was loaded from the Reacting library.
- A function declaration should begin with a capital letter, followed by the function name.
- Between parentheses, a parameter is passed; in React, this parameter is called props.
- A return method has JSX in it.
- Export with the same name as the function.
- The ES 6 syntax can also be used to create functional components, such as arrow functions. The function keyword is removed, but everything else is the same.

→ React Fragment

- There is **only one container we can return**. For instance, in this case, the div container is the one we are returning, and we are unable to return any tags outside of it.
Example:

```
src > components > JS Container.js > ...
1  import React from 'react'
2
3  export default function Container() {
4    return (
5      <div>Container</div>
6    )
7  }
8
```

Arrow Functional components:

- Over components with particular function keywords, arrow functions are the preferred method for most developers when creating functional components.
- The arrow function makes it easier to design components, and the code is clean and easy to understand.
- It also has the advantage of dealing with this context dynamically, whereas **arrow functions** deal with it lexically.

Here is an example of a functional component for React that uses the Arrow function:

→ Stateless

```
src > components > JS Container.js > ...
1  import React from "react";
2
3  const Container = (props) => {
4    return (
5      <div>
6        <h1>Arrow functions</h1>
7        <p>{props.text}</p>
8      </div>
9    );
10 };
11 export default Container;
12
13
```

① Don't have its own internal state.

② No Life cycle Methods

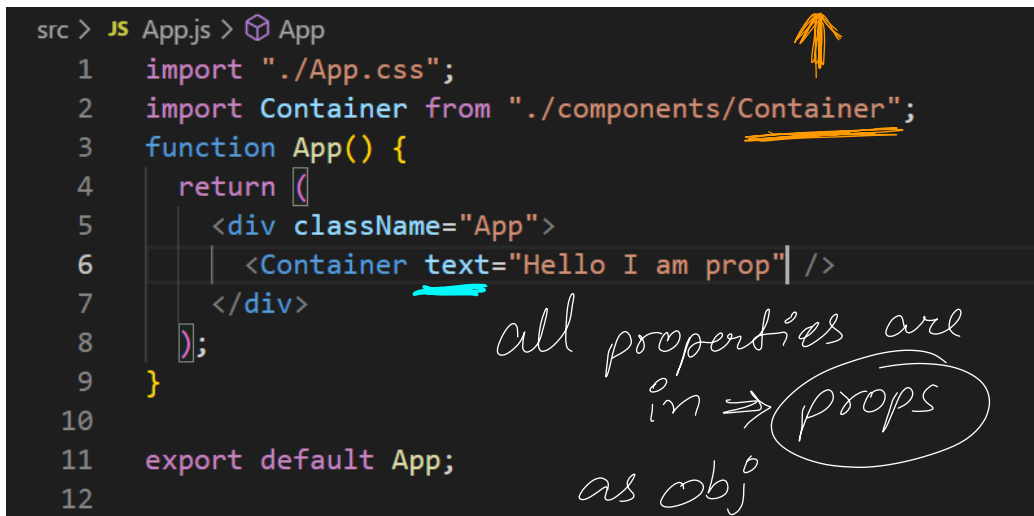
③ Using Props from Parent to child

- Even so, nothing has changed concerning props, thus we can still send props to the arrow functions. We shall export this component at the end so that it can be used in future components.

④ Don't Manage State

⑤ Focus on presentational purpose

- The procedure for importing the Container component and employing it to render the data is demonstrated below:



```

src > JS App.js > App
1  import './App.css';
2  import Container from './components/Container';
3  function App() {
4      return (
5          <div className="App">
6              <Container text="Hello I am prop" />
7          </div>
8      );
9  }
10
11  export default App;
12

```

all properties are in => props as obj

Stateless functional components:

- The purpose of building stateless functional components is to concentrate on the user interface. Stateless functional components lack state and lifecycle management.
- It is simple to create and comprehend by other developers. We also don't have to be concerned about this keyword.
- It enhanced overall performance because we no longer had to worry about managing the state.

2. Functional components vs class components:

- A functional component is a pure JavaScript function that takes props as an argument and returns a React element (JSX). You must extend from React to create a class component. Create a component and a render method that returns a React element.
- In functional components, no render method is used. It must have a render() method that returns JSX (which is syntactically similar to HTML)
- When the function is returned, functional components no longer exist and operate from top to bottom. The class component is created, and several life cycle methods are maintained alive and executed, and invoked depending on the phase of the class component.
- Functional components are sometimes known as stateless components because they only accept data and show it in some manner, and they are primarily responsible for UI rendering. On the other hand, class-based components use logic and state, they are also called stateful components.
- Functional components cannot use React lifecycle methods. Within class components, React lifecycle methods can be used.

- Hooks may easily be utilized to create functional components stateful.
example: `const [name, setName] = React.useState('')`
- The syntax for hook implementation inside a class component must be distinct.

example: `constructor(props) {

 super(props);

 this.state = {name: ''}

}`

- Constructors are not used in functional components. Constructors are used because the state must be stored in class-based components.

3. How to use functional components:

a. Syntax of functional component:

A functional component is a simple function that returns a valid React element, as previously stated.

```
1  import React from "react";
2  import "./App.css";
3
4  function App() {
5      return (
6          <div className="App">
7              <h1>My First Functional Component</h1>
8          </div>
9      );
10 }
11
12 export default App;
13
```

This is a very fundamental functional part that displays static text on the screen.

Note: The ES6 arrow function can also be used to create functional components.

b. Passing props in functional components:

React's functional components are pure javascript functions. It accepts an object called props (which stands for properties) as an argument and produces JSX.

Let's break this down with some code.

Inside src, create a folder called components, i.e. /src/components. This is merely a practice that all react developers use; we will write all of our components in this folder and then import them into App.js.

Let's make a file called Person.js in the components folder:

```
src > components > JS Person.js > ...
1  import React from "react";
2  import "./Person.css";
3
4  const Person = (props) => {
5    return (
6      <div className="person">
7        <h2>Name: {props.name}</h2>
8        <h2>Age: {props.age}</h2>
9      </div>
10   );
11 };
12
13 export default Person;
14
```

And App.js seems to be as follows:

```
src > JS App.js > [⌕] default
1  import React from "react";
2  import "./App.css";
3  import Person from "../components/Person";
4
5  function App() {
6    return (
7      <div className="App">
8        <Person name="David" age={20} />
9      </div>
10   );
11 }
12
13 export default App;
```

c. State in functional components:

useState is a React hook for the state (explain with an example)

React Hooks are functions that allow function components to access React's state and lifecycle attributes. Hooks allow us to manipulate the state of our functional components without converting them to class components.

```
src > JS App.js > [⌕] default
1  import React, { useState } from 'react'
2  import './App.css';
3
4  function App() {
5
6      const [counter, setCounter] = useState(0)
7
8      const clickHandler = () => {
9          const updatedCounter = counter + 1;
10         setCounter(updatedCounter);
11     }
12
13     return (
14         <div className="App">
15             <div>Counter Value: {counter}</div>
16             <button onClick={clickHandler}>
17                 Increase Counter
18             </button>
19         </div>
20     );
21 }
22
23 export default App;
24
```

Internal State (with arrow pointing to `useState(0)`)

absence of () while calling function (with arrow pointing to `clickHandler`)

To clarify what we've said above, we're utilizing a state variable named counter, with an initial value of 0.

When we press the "Increase Counter" button, the current counter value is increased by one, and the current state is updated using the setCounter method.

When the state is altered or updated, the UI is refreshed or the component is re-rendered, we see the revised counter value on the screen.

Note that a state might be a text, number, array, object, or boolean, whereas props is an object.