# Types of React Components

## Functional, Class, Pure, and Higher-Order Components

**What are React Components?**

React components are the building blocks of any React application. They are re-usable and can be nested inside other components. Components serve the same purpose as JavaScript functions, but work individually to return JSX code as elements for our UI.

Components usually come in two types, functional components and class components, but today we will also be talking about pure components and higher-order components.

## Types of React Components Discussed

1. **Stateless Functional Components**— functions that return JSX. Can do more stuff with React Hooks.
2. **Stateful Class Components**— classes that can manipulate state, props, and lifecycle methods.
3. **Pure Components** — functional components that performs shouldComponentUpdate() automatically. Re-renders only if state or props is different from previous state or props.
4. **Higher-Order Components**— functions that return one or multiple components, depending on array data.
5. **Stateless Functional Components**

A stateless function component is a typical React component that is defined as a function that does not manage any state.

There are no constructors needed, no classes to initialize, and no lifecycle hooks to worry about.

These functions simply take props as an input and return JSX as an output.

They can be as simple as this:

const Title = ({ title }) => {

return <h1>{title}</h1>;

};

The above example is an arrow function component which let us write the clearest and most concise react component possible.

Stateless function components are really nice to work with and I'm a big fan of them because these components have consistent output based on their inputs since they have no state or side-effects.

If you give a stateless function component a set of props it will **always** render the same JSX.

This differs from a Stateful component where state can affect the JSX output, even if the same inputs are given.

## Why Use Function Components?

Breaking up your React code into stateless function components can help in many ways.

It's easier to test a stateless function component since there are fewer unexpected outputs. The JSX should always be consistent when you are given an input.

You don't have to define so many things. There is significantly less boilerplate when creating function components.

Also, these components offer better readability and an easier way to create new components. Other developers can understand functions without needing to understand the inherited class methods and logic of traditional class components. In other words, no need to extend React.Component or have any knowledge around what calls the render() method.

### I like this concept, but I still need states!

You can combine this approach with traditional stateful components to get the best of both worlds.

React Hooks make it possible to add state to Function Components. There is a specific hook called the**[useState hook](https://testsuite.io/use-state-hooks-in-react)** that manages state in your component with a single line of code. Goodbye class component boilerplate!

And since you no longer have access to traditional React lifecycle methods like onComponentDidMount, you can achieve almost all of the same logic by taking advantage of React's **useEffect hook**.

Hooks are a big improvement that have made it easier to learn and work with React in general.

Functional vs. Class components

1. Functional components are simple components receiving props and returning a declaration. That’s why we use functional component as much as possible.
2. This key word is absent in functional component that leads to simplicity.
3. In functional component we have solutions without state. So the maintenance and debugging of application is much easier.
4. Functional components tend to be without any complicated logic and mainly responsible for the UI.
5. That’s why functional component is called the stateless components.

Class components on the other hand,

1. More feature rich
2. They can maintain their own private data also called state.
3. They contain complicated logic
4. Most importantly they provide life cycle hooks.
5. Class component that’s why called Stateful component.

### Class Components

Class components have previously been the most commonly used among the four component types. This is because class components are able to do everything a functional component do but more. It can utilize the main functions of React, **state**,**props**, and **lifecycle methods**.

// Class Component Example

import React from 'react';

class HelloWorld extends React.Component {  
 render() {  
 return (  
 <div>  
 <p>Hello World!</p>  
 </div>  
 )  
 }  
}export default HelloWorld;

Class components use extends React.Component after declaring the class HelloWorld and requires a render() method to return JSX code. In this class component, you can declare a state, set it to a JavaScript object, and use props to be the initial state or change the state in **lifecycle methods**. Some lifecycle methods are componentDidMount(), componentDidUpdate(), and componentWillUnmount(). These are actions that a functional component cannot perform unless they use React Hooks.

### 3. Pure Component

Pure components are the most simple and fast components that someone can write. As pure components don’t modify or depend on the state of different variables outside of their scope, they can easily replace the simple functional components. The components which merely return the render function are most adequate for pure components. The biggest use case of the Pure component is to provide optimizations.

The big difference between the React.Component and React.PureComponent is that the pure component exhibits **shallow comparison**on the state change. The pure components automatically manage the shouldComponentUpdate().

React Components are re-rendered mostly when:

* setState() Is called
* forceUpdate() Is called
* props Values are updated

### 4. Higher-Order Components

Higher-order components also known as HOC are less like a React component and more like a pattern, the result of React’s Compositional nature. The main use case of HOC is to share the logic with other components.

// HOC Example

import React from 'react';

import MyComponent from './components/MyComponent';

class HelloEmizen extends React.Component {

render() {

return(

<div>

{this.props.myArray.map((element) => (

<MyComponent data={element} key={element.key} />

))}

</div>

)

}

}

export default HelloEmizen;

The above image of code has the simple component to describe the higher-level component. Here the key code is this.props.myArray.map((element) => (<MyComponent />). This function returns the components. The number of components are simply dependent on the number of elements in the array which are called HOC. The function takes an array from state and thus maps each element in the array by transforming every element into the React Component.

Followings are the Higher-Order Component Simple Rundown:

1. Get the data from props or state
2. Map over the array and return the Reach component for every element.