Components : React components are independent and reusable code. They are the building blocks of any React application. 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 components:

Types of React Components

- Functional Components
- Class Components
- Pure Components
- Higher-Order Components

Functional Components

Functional components are functions that takes in props and return JSX. They do not natively have state or lifecycle methods, but this functionality can be added by implementing React Hooks. Functional components are usually used to display information. They are easy to read, debug, and test.

```
// Functional Component Example
import React from 'react';
const HelloWorld = () => {
   return (
      <div>
         <p>Hello World!</p>
      </div>
   )
}export default HelloWorld;
```

In the code above, it is a very simple component that consists of a constant variable HelloWorld that is assigned to an arrow function which returns JSX. Functional components do not need to be arrow functions. They can be declared with regular JavaScript functions. You can also pass in props to the function and use them to render data in the JSX code.

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***. Unlike functional components, class components are consist of … well, a class.

```
// Class Component Example
import React from 'react';
class HelloWorld extends React.Component {
   render() {
      return (
         <div>
            <p>Hello World!</p>
         </div>
      )
   }
}export default HelloWorld;
```

Class component syntax differs from functional component syntax. 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.

Pure Components

Pure components are like better functional components. Components that only return a render function are optimal for pure components to shine. However, you will need to understand what a pure component does.

Pure components are primarily used to provide optimizations. They are the simplest and fastest components we can write. They do not depend or modify the state of variables outside its scope. Hence, why pure components can replace simple functional components.

But, one major difference between a regular React.Component and a React.PureComponent is that pure components perform ***shallow comparisons*** on state change. Pure components take care of shouldComponentUpdate() by itself. If the previous state and/or props are the same as the next, the component is not re-rendered.

React Components are usually re-rendered when:

setState() is called

props values are updated

forceUpdate() is called

But in pure components, the React components do not re-render blindly without considering the updated state and props. Thus, if state and props are the same as the previous state and props , then the component does not re-render. This is very useful when you do not want to re-render a certain component while the props and state remain the same.

For example, you don't want one component, let's say a music player, to re-render because another component updates, maybe when you create a playlist. You might want the music player to stay constant at one song and not replay whenever you change a different component's state or props. Pure components would be very useful in these types of situations.

Higher-Order Components

Higher-order components(HOC) is an advanced technique in React for reusing component logic. It is not a React component you can find in the API. It is a pattern that emerged from React's compositional nature. Basically, HOCs are functions that return a component(s). They are used to share logic with other components.

```
// HOC Exampleimport React from 'react';
import MyComponent from './components/MyComponent';
class HelloWorld extends React.Component {
   render() {
      return(
         <div>
            {this.props.myArray.map((element) => (
               <MyComponent data={element} key={element.key} />
            ))}
         </div>
      )
   }
}export default HelloWorld;
```

Conclusion
A short summary of the components discussed:

Functional Components — functions that return JSX. Can do more stuff with React Hooks.

Class Components — classes that can manipulate state, props, and lifecycle methods.

Pure Components — functional components that performs `shouldComponentUpdate()` automatically. Re-renders only if state or props is different from previous state or props.

Higher-Order Components — functions that return one or multiple components, depending on array data.