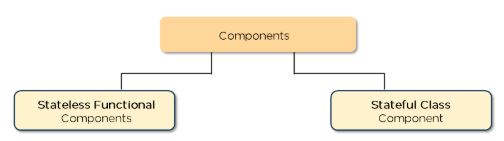
ReactJS Interview Questions on Components

### **What are the components in React?**

Components are the building blocks of any React application, and a single app usually consists of multiple components. A component is essentially a piece of the user interface. It splits the user interface into independent, reusable parts that can be processed separately.

There are two types of components in React:



* **Functional Components:**These types of components have no state of their own and only contain render methods, and therefore are also called **stateless components**. They may derive data from other components as props (properties).

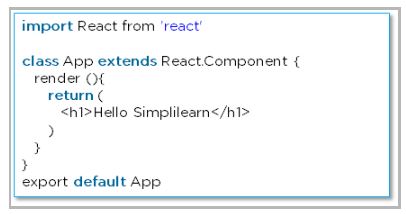
|  |
| --- |
| function Greeting(props) {    return <h1>Welcome to {props.name}</h1>;  } |

* **Class Components:**These types of components can hold and manage their own state and have a separate render method to return JSX on the screen. They are also called Stateful components as they can have a state.

|  |
| --- |
| class Greeting extends React.Component {    render() {      return <h1>Welcome to {this.props.name}</h1>;    }  } |

### **What is the use of render() in React?**

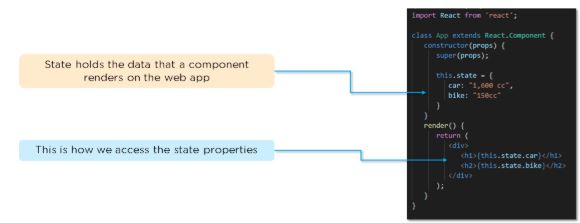
* It is required for each component to have a render() function. This function returns the HTML, which is to be displayed in the component.
* If you need to render more than one element, all of the elements must be inside one parent tag like <div>, <form>.



### **What is a state in React?**

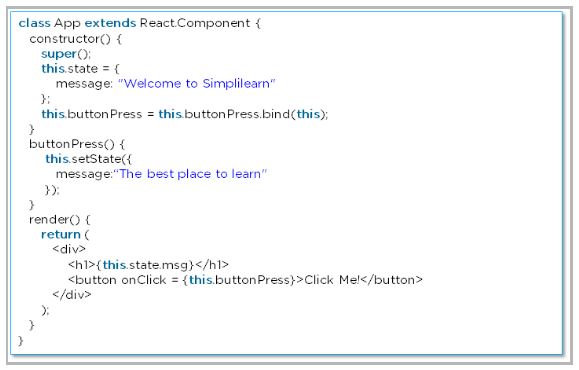
* The state is a built-in React object that is used to contain data or information about the component. The state in a component can change over time, and whenever it changes, the component re-renders.
* The change in state can happen as a response to user action or system-generated events. It determines the behavior of the component and how it will render.

### **How do you implement state in React?**



### **How do you update the state of a component?**

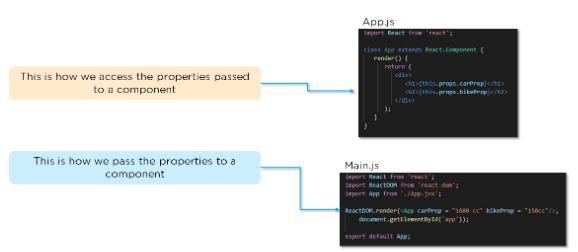
We can update the state of a component by using the built-in **‘setState()’**method:



### **What are props in React?**

* [Props](https://www.simplilearn.com/tutorials/reactjs-tutorial/react-props) are short for Properties. It is a React built-in object that stores the value of attributes of a tag and works similarly to HTML attributes.
* Props provide a way to pass data from one component to another component. Props are passed to the component in the same way as arguments are passed in a function.

### **How do you pass props between components?**



### **What are the differences between state and props?**

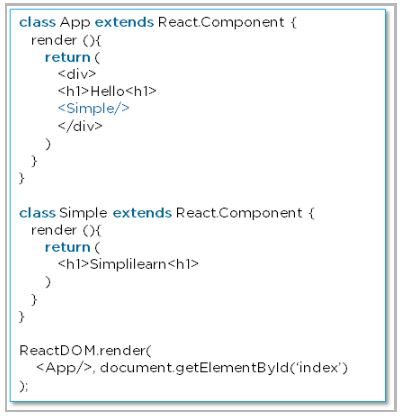
|  |  |  |
| --- | --- | --- |
|  | **State** | **Props** |
| Use | Holds information about the components | Allows to pass data from one component to other components as an argument |
| Mutability | Is mutable | Are immutable |
| Read-Only | Can be changed | Are read-only |
| Child components | Child components cannot access | Child component can access |
| Stateless components | Cannot have state | Can have props |

### **9. What is a higher-order component in React?**

A higher-order component acts as a container for other components. This helps to keep components simple and enables re-usability. They are generally used when multiple components have to use a common logic.

### **10. How can you embed two or more components into one?**

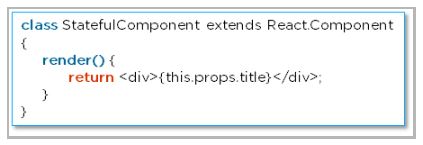
We can embed two or more components into one using this method:



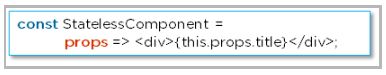
### **11. What are the differences between class and functional components?**

|  |  |  |
| --- | --- | --- |
|  | **Class Components** | **Functional Components** |
| State | Can hold or manage state | Cannot hold or manage state |
| Simplicity | Complex as compared to the stateless component | Simple and easy to understand |
| Lifecycle methods | Can work with all lifecycle methods | Does not work with any lifecycle method |
| Reusability | Can be reused | Cannot be reused |

* **Class components example:**



* **Functional components example:**



### **12. Explain the lifecycle methods of components.**

* **getInitialState():** This is executed before the creation of the component.
* **componentDidMount():** Is executed when the component gets rendered and placed on the DOM.
* **shouldComponentUpdate():** Is invoked when a component determines changes to the DOM and returns a “true” or “false” value based on certain conditions.
* **componentDidUpdate():** Is invoked immediately after rendering takes place.
* **componentWillUnmount():** Is invoked immediately before a component is destroyed and unmounted permanently.