1. **Defining props in React**

In React, props (short for properties) are a mechanism for passing data and functionality from a parent component to its child components. Think of them as the arguments you pass to a function, but for React components.

Here's a detailed explanation:

1. Data transfer

* Props are the primary way to pass information from one component to another in React.
* Data typically flows unidirectionally, meaning from parent components down to child components.

2. Read-only nature

* Props are immutable, which means that once a component receives props, it cannot change or modify them.
* Child components can only read the data passed through props and should not attempt to alter it.

3. Customization and reusability

* Props allow you to customize the behavior and appearance of child components based on the data they receive from their parent.
* This makes components reusable, as you can pass different props to the same component to render varied content or behavior.

4. Analogy to HTML attributes

* Props can be thought of as similar to HTML attributes, but you can pass any JavaScript value through them, including objects, arrays, functions, and more.

5. Usage in functional components

* In functional components, props are received as the first argument of the function.
* You can access individual props using dot notation (e.g., props.name) or destructure them directly in the function parameters for cleaner code (e.g., ({ name, age }) => { ... }).

6. Usage in class components

* In class components, props are available via the this.props object.

7. Default props

* You can set default values for props using defaultProps in class components or by providing default values directly during destructuring in functional components.
* This ensures that a component behaves as expected even if a particular prop is not provided by the parent.

Example

javascript

function App() {

const userName = "Alice";

const userAge = 30;

return (

<UserProfile name={userName} age={userAge} />

);

}

function UserProfile(props) {

return (

<div>

<h1>Hello, {props.name}!</h1>

<p>You are {props.age} years old.</p>

</div>

);

}

1. **Explain Default Props**

Default props are a feature in React that allows you to specify default values for a component's props. This means if a parent component doesn't explicitly pass a value for a particular prop to its child component, the child component will use the default value instead.

Why are default props necessary?

* Robustness: They help make your components more robust by ensuring that they always receive a value for a prop, even if the parent forgets to provide one. This prevents potential errors and unexpected behavior in your application.
* Predictability: Default props make a component's behavior more predictable. You can rely on the default value being present, which simplifies component logic.
* Reusability: They enhance component reusability. You can design a component that works out of the box with default settings, and only explicitly pass props when you need to override those defaults.
* Documentation: Default props act as a form of self-documentation, clearly indicating what props a component expects and what their typical values are.

How to use default props

1. In functional components (using default parameters or defaultProps)

* Default Parameters (ES6 Feature): This is the most common and recommended way to set default props in functional components. You can directly assign default values to the destructured props in the function signature.

javascript

import React from 'react';

const Greeting = ({ name = "Guest", message = "Hello" }) => {

return (

<div>

<h1>{message}, {name}!</h1>

</div>

);

};

export default Greeting;

* Using Component.defaultProps (Less common for functional components now): While it's more typical for class components, you can also use the defaultProps static property for functional components.

javascript

import React from 'react';

const Greeting = (props) => {

return (

<div>

<h1>{props.message}, {props.name}!</h1>

</div>

);

};

Greeting.defaultProps = {

name: "Guest",

message: "Hello"

};

export default Greeting;

2. In class components (using static defaultProps or Component.defaultProps)

* static defaultProps (Recommended): Define static defaultProps as a property within the class itself.

javascript

import React, { Component } from 'react';

class Greeting extends Component {

static defaultProps = {

name: "Guest",

message: "Hello"

};

render() {

return (

<div>

<h1>{this.props.message}, {this.props.name}!</h1>

</div>

);

}

}

export default Greeting;

* Component.defaultProps (Outside the class): Define defaultProps as a property of the component class outside the class definition.

javascript

import React, { Component } from 'react';

class Greeting extends Component {

render() {

return (

<div>

<h1>{this.props.message}, {this.props.name}!</h1>

</div>

);

}

}

Greeting.defaultProps = {

name: "Guest",

message: "Hello"

};

export default Greeting;

Example usage

javascript

import React from 'react';

import Greeting from './Greeting';

function App() {

return (

<div>

<Greeting /> {*/\* Renders "Hello, Guest!" \*/*}

<Greeting name="Alice" /> {*/\* Renders "Hello, Alice!" \*/*}

<Greeting message="Hi" /> {*/\* Renders "Hi, Guest!" \*/*}

<Greeting name="Bob" message="Greetings" />

</div>

);

}

export default App;

1. **Identify the differences between State and Props**

|  |  |
| --- | --- |
| **PROPS** | **STATE** |
| The Data is passed from one component to another. | The Data is passed within the component only. |
| It is Immutable (cannot be modified). | It is Mutable ( can be modified). |
| Props can be used with state and functional components. | The state can be used only with the state components/class component (Before 16.0). |
| Props are read-only. | The state is both read and write. |

1. **Explain reactDOM.render()**

In React, the ReactDOM.render() function plays a crucial role in displaying your React components on a webpage by rendering them into a specified Document Object Model (DOM) node.

Here's how it works:

1. Role and purpose

* The ReactDOM.render() method is part of the react-dom package, which provides DOM-specific methods for React applications.
* It serves as the entry point for your React application, taking a React element (which can be a component or a plain HTML tag) and rendering it into the actual DOM.
* This method is responsible for bridging the gap between React's virtual DOM and the actual browser DOM, ensuring that your UI is efficiently updated when data changes.

2. Syntax and arguments

The ReactDOM.render() method takes two required arguments and an optional third argument:

* element: The React element or component you want to render.
* container: The DOM node where you want to render the element. This is usually a <div> element in your index.html file.
* callback (optional): A function that executes after the component has been rendered or updated.

Example

javascript

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

ReactDOM.render(

<App />, *// The React element (your main component)*

document.getElementById('root')

);

Use code with caution.

In this example, the App component will be rendered inside the HTML element with the ID root.

3. How it works

* When ReactDOM.render() is called, React begins a process called reconciliation.
* It compares the new virtual DOM representation of your component with the previous one.
* Based on these differences, React efficiently updates only the necessary parts of the actual DOM, avoiding costly full page reloads and optimizing performance.

4. ReactDOM.render() vs. ReactDOM.createRoot()

* While ReactDOM.render() is still valid, in React 18 and later, the new root API using createRoot() is the recommended way to render components.
* createRoot() offers better performance and enables new concurrent features like automatic batching of updates, streaming server-side rendering, and transitions.

Example of createRoot()

javascript

import { createRoot } from 'react-dom/client';

import App from './App';

const root = createRoot(document.getElementById('root'));

root.render(<App />);