### Passing Data with Props in React

In React, \*\*props\*\* (short for "properties") are a way to pass data from one component to another, typically from a parent component to a child component. Props are read-only, meaning they are immutable and cannot be modified by the child component receiving them.

---

### \*\*1. Understanding Props\*\*

Props allow components to be dynamic and reusable by allowing data to be passed into them. Here's a basic overview:

- \*\*Parent Component:\*\* The component that passes the data.

- \*\*Child Component:\*\* The component that receives the data and uses it to render content.

### \*\*2. Passing Props from Parent to Child\*\*

#### \*\*Example: Basic Prop Usage\*\*

Let’s start with a simple example where a parent component passes a `name` prop to a child component.

```javascript

**// ParentComponent.js**

**import React from 'react';**

**import ChildComponent from './ChildComponent';**

**function ParentComponent() {**

**return (**

**<div>**

**<h1>Parent Component</h1>**

**<ChildComponent name="Alice" />**

**</div>**

**);**

**}**

**export default ParentComponent;**

```

```javascript

**// ChildComponent.js**

**import React from 'react';**

**function ChildComponent(props) {**

**return (**

**<div>**

**<h2>Child Component</h2>**

**<p>Hello, my name is {props.name}.</p>**

**</div>**

**);**

**}**

**export default ChildComponent;**

```

\*\*Explanation:\*\*

- In `ParentComponent`, we pass a prop named `name` with the value `"Alice"` to `ChildComponent`.

- In `ChildComponent`, we access the `name` prop using `props.name` and display it.

\*\*Output:\*\*

```

Parent Component

Hello, my name is Alice.

```

### \*\*3. Passing Multiple Props\*\*

You can pass multiple props to a child component. For instance, let's pass both `name` and `age` props.

```javascript

**// ParentComponent.js**

**import React from 'react';**

**import ChildComponent from './ChildComponent';**

**function ParentComponent() {**

**return (**

**<div>**

**<h1>Parent Component</h1>**

**<ChildComponent name="Bob" age={25} />**

**</div>**

**);**

**}**

**export default ParentComponent;**

```

```javascript

**// ChildComponent.js**

**import React from 'react';**

**function ChildComponent(props) {**

**return (**

**<div>**

**<h2>Child Component</h2>**

**<p>Hello, my name is {props.name} and I am {props.age} years old.</p>**

**</div>**

**);**

**}**

**export default ChildComponent;**

```

\*\*Output:\*\*

```

Parent Component

Hello, my name is Bob and I am 25 years old.

```

### \*\*4. Destructuring Props\*\*

To make your code cleaner and more readable, you can destructure the props directly in the function parameters.

```javascript

**// ChildComponent.js**

**import React from 'react';**

**function ChildComponent({ name, age }) {**

**return (**

**<div>**

**<h2>Child Component</h2>**

**<p>Hello, my name is {name} and I am {age} years old.</p>**

**</div>**

**);**

**}**

**export default ChildComponent;**

```

This approach is especially useful when your component is receiving multiple props.

### \*\*5. Passing Functions as Props\*\*

Props can also be used to pass functions from a parent component to a child component. This allows the child component to communicate back to the parent.

#### \*\*Example: Passing a Function Prop\*\*

```javascript

**// ParentComponent.js**

**import React from 'react';**

**import ChildComponent from './ChildComponent';**

**function ParentComponent() {**

**const handleClick = () => {**

**alert('Button clicked!');**

**};**

**return (**

**<div>**

**<h1>Parent Component</h1>**

**<ChildComponent onButtonClick={handleClick} />**

**</div>**

**);**

**}**

**export default ParentComponent;**

```

```javascript

**// ChildComponent.js**

**import React from 'react';**

**function ChildComponent({ onButtonClick }) {**

**return (**

**<div>**

**<h2>Child Component</h2>**

**<button onClick={onButtonClick}>Click Me</button>**

**</div>**

**);**

**}**

**export default ChildComponent;**

```

\*\*Explanation:\*\*

- `ParentComponent` defines a `handleClick` function that shows an alert.

- This function is passed to `ChildComponent` via the `onButtonClick` prop.

- In `ChildComponent`, the `onButtonClick` function is attached to a button’s `onClick` event. When the button is clicked, the function from the parent component is executed.

### \*\*6. Default Props\*\*

You can define default prop values in case they are not provided by the parent component.

```javascript

**// ChildComponent.js**

**import React from 'react';**

**function ChildComponent({ name, age }) {**

**return (**

**<div>**

**<h2>Child Component</h2>**

**<p>Hello, my name is {name} and I am {age} years old.</p>**

**</div>**

**);**

**}**

**ChildComponent.defaultProps = {**

**name: 'John Doe',**

**age: 18,**

**};**

**export default ChildComponent;**

```

\*\*Explanation:\*\*

- If the parent component doesn't pass the `name` or `age` props, the `ChildComponent` will use the default values `"John Doe"` and `18`, respectively.

### \*\*7. Prop Types Validation (Optional)\*\*

You can validate the types of props your component receives using `PropTypes` to avoid bugs and improve code robustness.

```javascript

**import React from 'react';**

**import PropTypes from 'prop-types';**

**function ChildComponent({ name, age }) {**

**return (**

**<div>**

**<h2>Child Component</h2>**

**<p>Hello, my name is {name} and I am {age} years old.</p>**

**</div>**

**);**

**}**

**ChildComponent.propTypes = {**

**name: PropTypes.string.isRequired,**

**age: PropTypes.number,**

**};**

**export default ChildComponent;**

```

\*\*Explanation:\*\*

- `PropTypes.string.isRequired` ensures that `name` is a required prop and must be a string.

- `PropTypes.number` ensures that `age` is a number, but it's not required.

### \*\*Summary\*\*

- \*\*Props\*\* are used to pass data from parent components to child components in React, making components dynamic and reusable.

- \*\*Destructuring\*\* props in the function parameters improves code readability.

- \*\*Functions\*\* can be passed as props, enabling child components to trigger actions in parent components.

- \*\*Default props\*\* ensure that components have fallback values when some props are not provided.

- \*\*PropTypes\*\* allow you to enforce type-checking on props, making your code more robust.

Understanding how to work with props is fundamental in React, as it allows components to communicate and share data effectively.