props is a convention, but you can technically call the parameter anything you want when defining a functional component.

https://www.w3schools.com/REACT/showreact.asp?filename=demo\_react\_props2

const MyComponent = (myProps) => {

return <div>{myProps.message}</div>;

};

However, props is the standard, and it makes your code more readable and familiar to others. React's documentation and most tutorials use props, so it's best to stick with the convention unless there's a specific reason to use a different name.

In React, children is a special prop that automatically refers to any nested elements within a component. You don't need to explicitly pass it down; React does this for you. It's a convention to call it children, but technically, you can rename it to anything you'd like.

const MyComponent = ({ children }) => {

return <div>{children}</div>;

};

This is the typical usage. However, you could rename children to something else if needed:

const MyComponent = ({ content }) => {

return <div>{content}</div>;

};

// Usage

<MyComponent>

<p>This is some content.</p>

</MyComponent>;

In this case, you would need to change the prop name to whatever you choose, but it's generally better to stick with children for consistency.

In React, the children prop is automatically passed to a component when you nest elements inside it, but it's a specific part of the props object. You can think of children as a special prop that React uses to pass the content inside the component tags to the component.

const MyComponent = ({ children }) => {

return <div>{children}</div>;

};

// Usage

<MyComponent>

<p>This is some content inside MyComponent.</p>

</MyComponent>;

In this case, the <p> tag inside <MyComponent> is automatically passed as the children prop, not as a normal props key. So when you destructure the props in the function signature like { children }, React will look for the nested content (anything between the opening and closing tags of the component) and assign it to children.

**Difference between children and props**

* props is the object that contains all the properties passed to a component.
* children is a special property within props, used specifically to represent whatever elements or content are nested inside the component.

const MyComponent = (props) => {

console.log(props); // props will include 'children' among other keys

return <div>{props.children}</div>;

};

// Usage

<MyComponent>

<p>This is inside children</p>

</MyComponent>;

Here, props will contain children (the <p> tag) as part of it, but React automatically picks it out as children to make it easier to access without needing to dig through the entire props object.

In React, props is an object that contains all the properties or attributes passed to a component. These can include:

**Custom Props**: Any custom properties you define when using the component.  
  
  
  
const MyComponent = ({ name, age }) => {

return <div>{name} is {age} years old</div>;

};

// Usage

<MyComponent name="John" age={30} />

**children**: The special prop that contains any nested elements or content passed between the opening and closing tags of a component.  
  
const MyComponent = ({ children }) => {

return <div>{children}</div>;

};

// Usage

<MyComponent>

<p>Some content</p>

</MyComponent>

**Key Prop (key)**: The key prop is automatically passed by React when rendering lists of components. It helps React identify which items have changed, been added, or removed for efficient re-rendering. It is **not** passed into the component itself, but React uses it internally.  
  
const MyList = ({ items }) => {

return (

<ul>

{items.map(item => <li key={item.id}>{item.name}</li>)}

</ul>

);

};

1. **ref**: ref is used to get a reference to a DOM element or a class component instance. It's passed through props but is handled differently.  
     
   const MyComponent = React.forwardRef((props, ref) => {

return <div ref={ref}>{props.children}</div>;

});

// Usage

const myRef = React.createRef();

<MyComponent ref={myRef} />

**defaultProps**: If you don’t pass a prop to a component, React can use the defaultProps of that component as a fallback value.  
  
  
const MyComponent = ({ name }) => {

return <div>{name}</div>;

};

MyComponent.defaultProps = {

name: 'John Doe',

};

// Usage (no name prop passed)

<MyComponent /> // "John Doe" will be used

1. **style**: If you want to pass inline styles to a component, you can use the style prop. It's an object that holds CSS properties as JavaScript objects.  
     
     
     
   const MyComponent = ({ style }) => {

return <div style={style}>Styled Component</div>;

};

// Usage

<MyComponent style={{ color: 'red', fontSize: '20px' }} />

**Example of a component receiving multiple props:**

const MyComponent = ({ title, children, style }) => {

return (

<div style={style}>

<h1>{title}</h1>

<div>{children}</div>

</div>

);

};

// Usage

<MyComponent title="Hello" style={{ color: 'blue' }}>

<p>This is the children content.</p>

</MyComponent>;

In the above example:

* title is a custom prop.
* children will automatically contain any nested elements like the <p> tag.
* style is passed as an object containing CSS styles.

**Special Props:**

* **dangerouslySetInnerHTML**: Used for inserting HTML content directly into a component, but should be used with caution to avoid XSS (cross-site scripting) vulnerabilities.
* **suppressContentEditableWarning**: Used to suppress warnings for elements with contentEditable attributes.
* **defaultChecked, defaultValue**: These are used in form elements to set default values, but they don’t update the state automatically like the regular checked or value props.

To summarize, props can contain:

* Custom props you pass
* children (for nested content)
* Special React props like key, ref, and style

**props** in React can be thought of as any attributes (or properties) passed to a component when it is used, similar to how HTML tags have attributes like class, id, src, etc. These attributes in React are passed as **props** to the component.

For example, in an HTML <img> tag, you might have attributes like src and alt. When you use an image in a React component, those attributes become **props**:

**Example:**

const ImageComponent = ({ src, alt }) => {

return <img src={src} alt={alt} />;

};

// Usage

<ImageComponent src="image.jpg" alt="An example image" />

In this case:

* The src and alt attributes of the <img> tag are passed as props to the ImageComponent.
* Inside ImageComponent, you destructure src and alt from the props and pass them to the img tag.

**React and HTML Tags**

When you create a React component, the props passed to it act like attributes of an HTML element. For example:

const Button = ({ label, onClick }) => {

return <button onClick={onClick}>{label}</button>;

};

// Usage

<Button label="Click Me" onClick={() => alert("Button clicked")} />

Here, label and onClick are like custom attributes that the <Button> component uses, similar to how HTML tags like <button> can have attributes such as onclick or type.

**props vs. HTML Attributes**

While React components can use standard HTML attributes (like className instead of class, htmlFor instead of for, etc.), custom props are passed to components in the same way as HTML attributes, but you decide the names and behavior for those props.

**Key Takeaways:**

* **HTML attributes** (like src, alt, href) are passed as **props** in React components.
* In React, **props** allow you to customize the behavior of components, similar to how HTML attributes affect elements.
* You can pass **custom props** to components, just as you pass attributes to HTML elements, but you’re free to choose the names and the values that make sense for your component.

Here are the key things **built into** **props** that React handles automatically or provides as part of the React system:

**1. children:**

* This is the most important built-in prop, automatically passed to a component when content is nested inside the component.
* It allows components to render nested content dynamically.

const MyComponent = ({ children }) => {

return <div>{children}</div>;

};

// Usage

<MyComponent>

<p>This is some child content!</p>

</MyComponent>

* In the example above, the <p> element inside <MyComponent> is automatically passed as the children prop.

**2. key:**

* This is a special prop used for efficiently rendering lists of components in React. key helps React identify which items have changed, been added, or removed.
* key is passed automatically to child components in list rendering and is **not** directly accessible in props.

const MyList = ({ items }) => {

return (

<ul>

{items.map(item => (

<li key={item.id}>{item.name}</li>

))}

</ul>

);

};

* In the example, the key helps React manage list items efficiently during re-rendering.

**3. ref:**

* This is a special prop used to get a reference to a DOM element or a class component instance. It is passed automatically for DOM elements.
* We see this with useRef()
* In function components, it’s usually used with React.forwardRef.

const MyComponent = React.forwardRef((props, ref) => {

return <div ref={ref}>Hello, world!</div>;

});

// Usage

const myRef = React.createRef();

<MyComponent ref={myRef} />;

* Here, the ref allows you to reference the <div> inside MyComponent.

**4. defaultProps:**

* React provides the ability to define default values for props if they are not passed in by the parent.
* It’s not a direct part of props, but it provides default behavior when props are missing.

const MyComponent = ({ name }) => {

return <div>{name}</div>;

};

MyComponent.defaultProps = {

name: 'John Doe'

};

**5. propTypes (optional, not automatically built-in):**

* Though not technically part of React's core functionality, propTypes are often used in React to validate the types of props passed into components.
* It’s helpful for documentation and validation during development.

import PropTypes from 'prop-types';

const MyComponent = ({ name, age }) => {

return <div>{name} is {age} years old</div>;

};

MyComponent.propTypes = {

name: PropTypes.string.isRequired,

age: PropTypes.number.isRequired

};

**Summary of what React automatically provides for props:**

* **children**: Automatically passed to a component when it has nested content.
* **key**: Used for efficient re-rendering in lists; handled internally by React.
* **ref**: Used for referencing DOM elements, passed with React.forwardRef.
* **defaultProps**: Sets default values for props.
* **propTypes**: Used for type-checking (optional).

React also allows you to pass **custom props** into components, which can contain any type of data you need, and you can define their behavior and structure.