# Props in React JS

<https://www.w3schools.com/react/react_props.asp>

www.geeksforgeeks.org/reactjs-props-set-1/

**Note :** Unidirectional data flow is a technique that is mainly found in functional reactive programming. It is also known as one-way data flow, which means the data has one, and only one way to be transferred to other parts of the application. In essence, this means child components are not able to update the data that is coming from the parent component. In React, data coming from a parent is called **props**.

**Example :** https://towardsdev.com/parent-to-child-component-communication-in-react-ea5b673de6ce

<https://www.javatpoint.com/react-state-vs-props>

https://www.freecodecamp.org/news/react-js-for-beginners-props-state-explained/#:~:text=Props%20are%20used%20to%20pass%20data%2C%20whereas%20state,but%20is%20private%20%28cannot%20be%20accessed%20from%20outside%29

State and Props are used to control the data flow in the react application. With the help of them we can render the component with dynamic data.

## What are props and how do we use them?

# In React, props stand for "properties" and they are the way we pass data from one React component to another. Props are passed as arguments into a React component. React’s data flow between components is uni-directional (from parent to child only).

The syntax for passing props is the same for class and functional components, but the syntax for accessing these props is different.

## How to Pass Props to Class Component

To pass props to a class component you define the name of the prop and then pass a value inside curly brackets.

To access the props you will use the **this** keyword to access the **props** object which contains the props you passed into the component.

## How to Pass Props to Functional Component

For a functional component, you will pass props the same way as class components, define the name of the prop and then pass a value inside curly brackets.

Where class and functional components differ is in accessing props. In a functional component, you will see the props object as a parameter and can access it without the **this** keyword.

Example :

// example of props in functional and class component

import React from 'react';

import ReactDOM from 'react-dom';

/\*

function BookDetails(getData){

    return (<div>

                <span>{getData.send.title}</span>

                <p>{getData.send.description}</p>

           </div>);

}

function SetDetails(){

    const book = {

        title : "Mysterious Person",

        description : "Stay Away"

    };

    return (<BookDetails send={book}/>);

}

\*/

class BookDetails extends React.Component{

    render(){

        console.log(typeof BookDetails);

        console.log(this);

        return (<div>

                <span>{this.props.send.title}</span>

                <p>{this.props.send.description}</p>

           </div>);

}

}

//class SetDetails extends BookDetails{

    class SetDetails extends React.Component{

        render(){

    const book = {

        title : "Mysterious Person in the world",

        description : "Stay Away with them"

    };

     // console.log(this);

    return (<BookDetails send={book}/>);

}

}

ReactDOM.render(<SetDetails/>,document.getElementById("root"));

# What is state?

# React has another special built-in object called state, which allows components to create and manage their own data. So unlike props, components cannot pass data with state, but they can create and manage it internally.

# Here is an example showing how to use state:

# class Test extends React.Component {

# constructor() {

# this.state = {

# id: 1,

# name: "test"

# };

# }

# 

# render() {

# return (

# <div>

# <p>{this.state.id}</p>

# <p>{this.state.name}</p>

# </div>

# );

# }}

# How do you update a component’s state?

# State should not be modified directly, but it can be modified with a special method called setState( ).

# this.state.id = “2020”; // wrong

# this.setState({ // correct

# id: "2020"});

# What happens when state changes?

# why must we use setState( )?

# Why do we even need the state object itself?

# Answer : A change in the state happens based on user-input, triggering an event, and so on. Also, React components (with state) are rendered based on the data in the state. State holds the initial information.

# So when state changes, React gets informed and immediately re-renders the DOM – not the whole DOM, but only the component with the updated state. This is one of the reasons why React is fast.

# And how does React get notified?

# You guessed it: with setState( ).

# The setState( ) method triggers the re-rendering process for the updated parts. React gets informed, knows which part(s) to change, and does it quickly without re-rendering the whole DOM.

# In summary, there are 2 important points we need to pay attention to when using state:

# State shouldn’t be modified directly – the setState( ) should be used

# State affects the performance of your app, and therefore it shouldn’t be used unnecessarily

# Can I use state in every component?

# Another important question you might ask about state is where exactly we can use it. In the early days, state could only be used in class components, not in functional components.

# That’s why functional components were also known as stateless components. However, after the introduction of React Hooks, state can now be used both in class and functional components.

# If your project is not using React Hooks, then you can only use state in class components.

# What are the differences between props and state?

# Finally, let’s recap and see the main differences between props and state:

# Components receive data from outside with props, whereas they can create and manage their own data with state

# Props are used to pass data, whereas state is for managing data

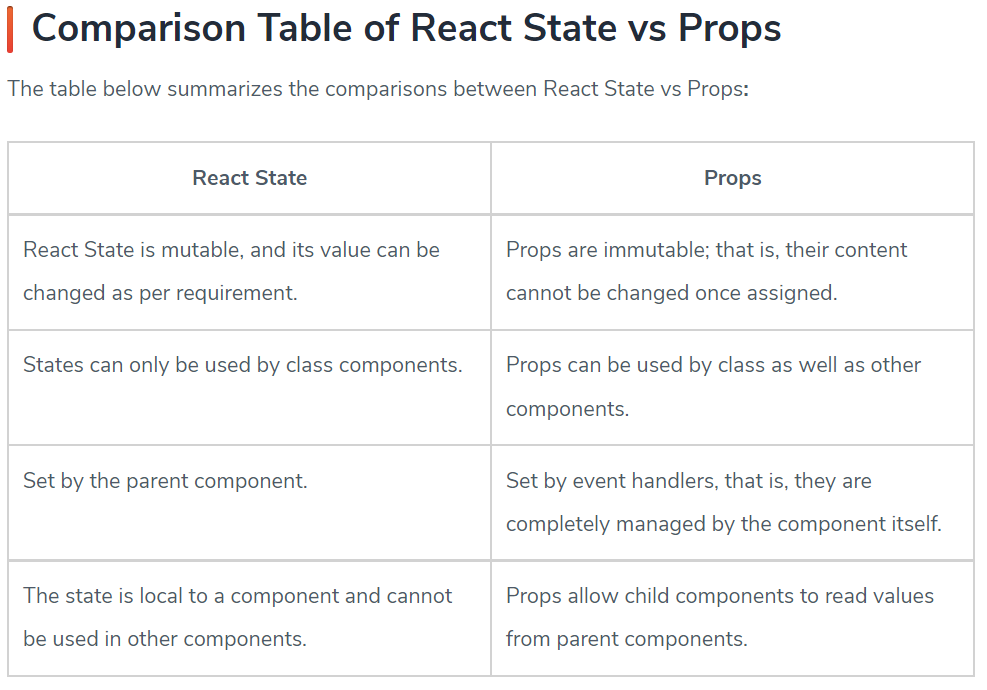
# Data from props is read-only, and cannot be modified by a component that is receiving it from outside

# State data can be modified by its own component, but is private (cannot be accessed from outside)

# Props can only be passed from parent component to child (unidirectional flow)

# Modifying state should happen with the setState ( ) method

1. Both Props and states are used for storing data related to a component.
2. The state is local to a component and cannot be used in other components, whereas Props allow child components to read values from parent components.



Note : super() is called inside a react component only if it has a constructor.However if we have a constructor then super() is mandatory.

The reason why “this” cannot be allowed before super() is because this is uninitialized if super() is not called.However even if we are not using this we need a super() inside a constructor because ES6 class constructors MUST call super if they are subclasses. Thus, you have to call super() as long as you have a constructor. We call super() inside the constructor if we have to use this.props.

If we do not put props inside super() like super(props), then in this case the value of props in Object{…props:undefined} is undefined. Which leads to some bugs in future. So if we use super(props) then in that case Object may looks like this Object{…props:{…}}.

Conclusion :

In React, both "state" and "props" are important concepts that are used to manage and pass data between components.

1. State: State is an internal data storage mechanism within a component. It represents the mutable values that can change over time. When the state of a component updates, React automatically re-renders the component, updating the UI to reflect the new state. State is managed using the `useState` hook in functional components or the `this.state` and `this.setState` methods in class components.

2. Props: Props (short for "properties") are read-only values passed from a parent component to its child components. They are used to pass data from a parent component down the component hierarchy. Props are immutable, meaning they cannot be modified by the child components. They are defined when rendering a component and can include any type of data, such as strings, numbers, objects, functions, or even other React components.

The main differences between state and props are:

- State is managed within a component and can be modified, while props are passed down from a parent component and are read-only in the child components.

- State is used to manage internal component data, while props are used to pass data from parent to child components.

- State is specific to a component instance, whereas props are passed to multiple instances of a component.

- State can only be accessed within the component it belongs to, while props can be accessed by the component receiving them.

Both state and props play important roles in React development, enabling components to handle dynamic data and communicate with each other effectively.

That's a great question and very common confusion when learning React.

Let's clarify the statement **“state is local to a component”** and how it still works when passing state to other components:

**✅ What "State is Local to a Component" Means:**

When we say **state is local**, we mean:

* The useState hook (or class component's this.state) creates state **inside a specific component**.
* That component is the **owner** of the state and is responsible for updating it.
* No other component can **directly access or modify** that state.

function Parent() {

const [count, setCount] = useState(0); // state is local to Parent

// Child cannot see this unless we pass it

}

**✅ How We Pass It to Other Components:**

React allows **unidirectional data flow** — data flows **from parent to child** through props.

So, while **state is defined locally**, we can **share it** (or its value) with child components by passing it via props:

function Parent() {

const [count, setCount] = useState(0); // local state

return <Child count={count} />; // passing value as prop

}

function Child({ count }) {

return <h1>{count}</h1>; // using parent's state value

}

Here, Child **cannot change** the count, unless we also pass down setCount.

**✅ Can Child Update Parent’s State?**

Yes, if you **pass the setState function** as a prop too:

function Parent() {

const [count, setCount] = useState(0);

return <Child count={count} updateCount={() => setCount(count + 1)} />;

}

function Child({ count, updateCount }) {

return (

<>

<h1>{count}</h1>

<button onClick={updateCount}>Increase</button>

</>

);

}

This still doesn’t break the rule. The state **still lives in the parent**, and only the parent can actually **modify** it. The child just **requests** the change via the function.

**✅ Why React Works This Way:**

This pattern makes apps easier to **debug, reason about, and maintain**. By controlling where the state lives and how it flows, React avoids messy two-way data binding (like in AngularJS).

**✅ Summary**

| **Concept** | **Explanation** |
| --- | --- |
| State is local | Only the component that creates it "owns" it |
| Sharing state | We can **pass state values via props** to children |
| Updating state from child | We can **pass the setter function** as a prop |
| Still local? | Yes — because only the owner can directly manage it |

Let me know if you want a diagram or code sandbox example!