# React Notes: Components and Props

## 1. What are Components?

In React, a component is a function (or class) that represents a part of the UI. For example: header, footer, button, or card can all be separate components.

## 2. What are Props?

Props (short for properties) allow you to pass data from a parent component to a child component. Props are read-only, means you can’t modify them inside the child component.

## 3. Example Code

App.js  
-------  
import React from "react";  
import Greeting from "./Greeting";  
  
export default function App() {  
 return (  
 <div>  
 <h1>React Components & Props Example</h1>  
 <Greeting name="Abhay" />  
 <Greeting name="Vishwakarma" />  
 </div>  
 );  
}  
  
Greeting.js  
-----------  
import React from "react";  
  
export default function Greeting(props) {  
 return <h2>Hello, {props.name}! Welcome to React 🚀</h2>;  
}

## 4. Key Points

- A component is like a function that returns JSX.  
- Props are used to make components dynamic.  
- Props are passed from the parent to the child component and are read-only.

# React Props and Components - Interview Questions & Tasks

This document contains important interview questions, tasks, and their solutions for React Components and Props. It will help you prepare for interviews and practice coding concepts effectively.

## 1. Interview Questions

* 1. What is the difference between functional and class components in React?
* 2. How do props work in React?
* 3. Can props be changed inside a component? Why or why not?
* 4. What are default props in React and how do you define them?
* 5. How can you pass a function as a prop? Give an example.
* 6. Explain the concept of children prop in React.
* 7. How do you validate props using PropTypes?
* 8. Difference between props and state?
* 9. How do you pass multiple props to a component?
* 10. What are some best practices for using props in React?

## 2. Tasks with Solutions

### Task 1: Create a Functional Component using Props

Write a component to display a user's name and age passed as props.

Solution:

import React from 'react';  
  
function UserInfo(props) {  
 return (  
 <div>  
 <h2>Name: {props.name}</h2>  
 <p>Age: {props.age}</p>  
 </div>  
 );  
}  
  
export default UserInfo;

### Task 2: Pass Function as Prop

Create a button component that triggers a function passed as a prop.

Solution:

import React from 'react';  
  
function ClickButton({ handleClick }) {  
 return <button onClick={handleClick}>Click Me</button>;  
}  
  
export default ClickButton;  
  
// Usage Example  
function App() {  
 const sayHello = () => alert('Hello from prop function!');  
 return <ClickButton handleClick={sayHello} />;  
}

### Task 3: Using Children Prop

Create a Card component that accepts content as children.

Solution:

import React from 'react';  
  
function Card({ children }) {  
 return <div className='card'>{children}</div>;  
}  
  
export default Card;  
  
// Usage Example  
function App() {  
 return (  
 <Card>  
 <h3>This is inside the card</h3>  
 <p>Reusable and flexible!</p>  
 </Card>  
 );  
}

# Class Component vs Functional Component in React

In React, components can be created in two ways: Class Components and Functional Components. Earlier, Class Components were widely used because they supported state and lifecycle methods. However, with the introduction of Hooks in React 16.8, Functional Components have become more popular.

## Key Differences

|  |  |  |
| --- | --- | --- |
| **Feature** | **Class Component** | **Functional Component** |
| Syntax | Created using 'class' keyword | Created using 'function' keyword |
| State Management | Uses this.state & this.setState() | Uses useState Hook |
| Lifecycle Methods | Uses lifecycle methods like componentDidMount, componentDidUpdate | Uses useEffect Hook |
| Code Simplicity | More complex and verbose | Simpler and more readable |
| Performance | Comparatively slower | Faster and lightweight |
| Hooks | Cannot use Hooks | Can use Hooks |
| ‘this’ Keyword | Requires 'this' keyword | No need for 'this' keyword |

## Examples

### Class Component Example

import React, { Component } from "react";  
  
class Counter extends Component {  
 constructor(props) {  
 super(props);  
 this.state = { count: 0 };  
 }  
  
 increment = () => {  
 this.setState({ count: this.state.count + 1 });  
 };  
  
 render() {  
 return (  
 <div>  
 <h2>Count: {this.state.count}</h2>  
 <button onClick={this.increment}>Increment</button>  
 </div>  
 );  
 }  
}  
  
export default Counter;

### Functional Component Example (with Hooks)

import React, { useState } from "react";  
  
function Counter() {  
 const [count, setCount] = useState(0);  
  
 return (  
 <div>  
 <h2>Count: {count}</h2>  
 <button onClick={() => setCount(count + 1)}>Increment</button>  
 </div>  
 );  
}  
  
export default Counter;

# React Hooks - Definitions and Examples

Hooks are special functions in React that allow you to use state and other React features in functional components. Before Hooks, only class components could manage state and lifecycle methods. Hooks make functional components more powerful and easier to use.

## 1. useState

Definition: useState is a Hook that lets you add state to functional components.

Example:

import React, { useState } from "react";  
  
function Counter() {  
 const [count, setCount] = useState(0);  
  
 return (  
 <div>  
 <h2>Count: {count}</h2>  
 <button onClick={() => setCount(count + 1)}>+1</button>  
 <button onClick={() => setCount(count - 1)}>-1</button>  
 </div>  
 );  
}  
export default Counter;

## 2. useEffect

Definition: useEffect is a Hook that lets you perform side effects (like data fetching, subscriptions, or timers) in functional components. It works like lifecycle methods.

useEffect is a react hook used to perform side effects in functional components.

Example:

import React, { useState, useEffect } from "react";  
  
function Timer() {  
 const [time, setTime] = useState(0);  
  
 useEffect(() => {  
 const interval = setInterval(() => {  
 setTime((prev) => prev + 1);  
 }, 1000);  
  
 return () => clearInterval(interval); // cleanup  
 }, []);  
  
 return <h2>Time: {time}</h2>;  
}  
export default Timer;

## 3. useRef

Definition: useRef is a Hook that creates a mutable reference which can hold a value without causing re-renders. It is often used to access DOM elements directly.

Example:

import React, { useRef } from "react";  
  
function InputFocus() {  
 const inputRef = useRef(null);  
  
 const focusInput = () => {  
 inputRef.current.focus();  
 };  
  
 return (  
 <div>  
 <input ref={inputRef} type="text" placeholder="Type here..." />  
 <button onClick={focusInput}>Focus Input</button>  
 </div>  
 );  
}  
export default InputFocus;

## 4. useContext

Definition: useContext is a Hook that allows you to access global data directly without passing props through every level of the component tree.

## 5. useMemo

Definition: useMemo is a Hook that memoizes the result of a calculation so that it doesn’t have to be recalculated on every render.

## 6. useCallback

Definition: useCallback is a Hook that returns a memoized function, preventing it from being recreated on every render.

## Summary

- useState → Add and manage state in functional components  
- useEffect → Perform side effects (like API calls, timers)  
- useRef → Hold values or access DOM elements directly  
- useContext → Share global data without props  
- useMemo → Optimize expensive calculations  
- useCallback → Optimize functions

# React useState Hook

Definition: useState is a React Hook that allows you to add state (data that can change) to functional components. State is like a variable that React watches, and whenever it changes, React re-renders the component to show the new value.

## What is State?

- State is data that can change during the lifetime of a component.  
- Example: a counter number, form input value, toggle button status.  
- When state changes, the UI updates automatically.

## Syntax of useState

const [variableName, setVariableName] = useState(initialValue);

- variableName → The current value of the state.  
- setVariableName → Function to update the state.  
- initialValue → The default value of the state.

## Example 1: Counter App

import React, { useState } from "react";  
  
function Counter() {  
 const [count, setCount] = useState(0);  
  
 return (  
 <div>  
 <h2>Count: {count}</h2>  
 <button onClick={() => setCount(count + 1)}>Increment</button>  
 <button onClick={() => setCount(count - 1)}>Decrement</button>  
 </div>  
 );  
}  
export default Counter;

## Example 2: Input Box

import React, { useState } from "react";  
  
function NameInput() {  
 const [name, setName] = useState("");  
  
 return (  
 <div>  
 <input   
 type="text"   
 placeholder="Enter your name"   
 value={name}   
 onChange={(e) => setName(e.target.value)}   
 />  
 <h3>Hello, {name} 👋</h3>  
 </div>  
 );  
}  
export default NameInput;

## Important Points

1. useState returns an array: [value, setValue].  
2. You cannot change state directly (e.g., count++ ❌).  
3. Always use the setter function (e.g., setCount(newValue)).  
4. Each time setCount() is called, the component re-renders.

## Simple Summary

useState is like a container for data inside a component. The first value is the current data, and the second is a function to update it.