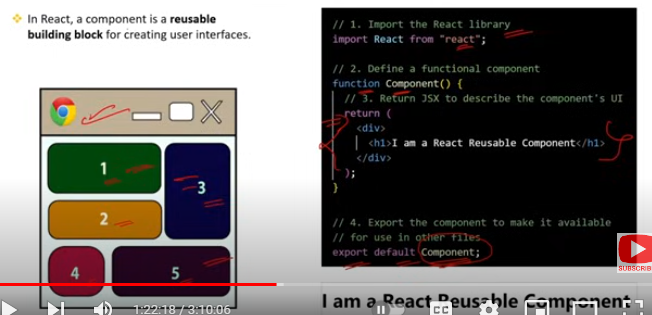
Q1)What are React components ?What are the main elements of it ?



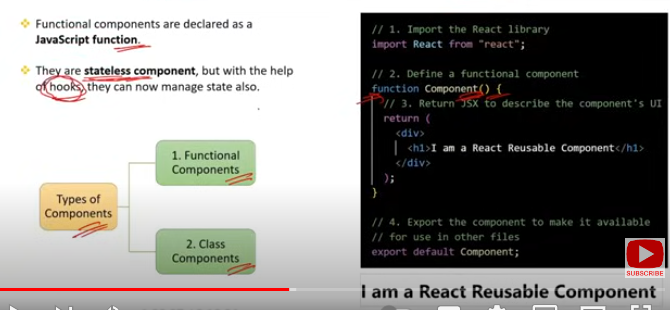
### What are React Components?

**React Components** are the building blocks of a React application. They are reusable pieces of UI that can have their own state and logic.

### Main Elements of a React Component:

1. **JSX**: JavaScript XML syntax that combines HTML with JavaScript.
2. **Props**: Short for properties; these are inputs to components, allowing data to be passed from parent to child.
3. **State**: Internal data storage for a component, used to manage dynamic data and re-render the UI when it changes.
4. **Lifecycle Methods**: Special methods that get called at different stages of a component’s lifecycle, such as componentDidMount or componentWillUnmount.

Q2)What are the types React components ?What are the Functional Components ?



### Types of React Components

**Class Components**:

* 1. ES6 classes that extend from React.Component.
  2. Can hold and manage state and lifecycle methods.

**Functional Components**:

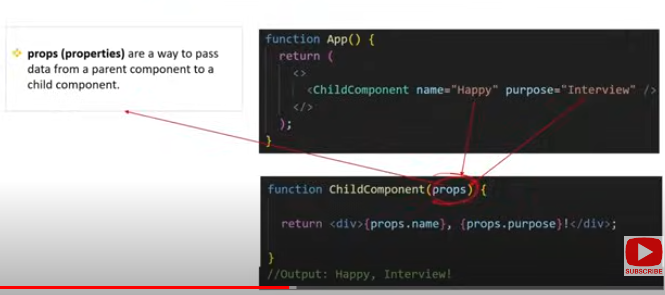
* 1. Simple JavaScript functions.
  2. Accept props as an argument and return JSX.
  3. Can use hooks to manage state and lifecycle features.

### Functional Components

**Functional Components** are JavaScript functions that:

* Receive props as an argument.
* Return JSX, defining the UI.
* Use React hooks (e.g., useState, useEffect) to manage state and lifecycle events.

Q3) How do you pass data between functional components in React ?



### Passing Data Between Functional Components in React

**Props** are the primary method for passing data from one functional component to another in React.

### Example:

#### Parent Component:



Child Component:



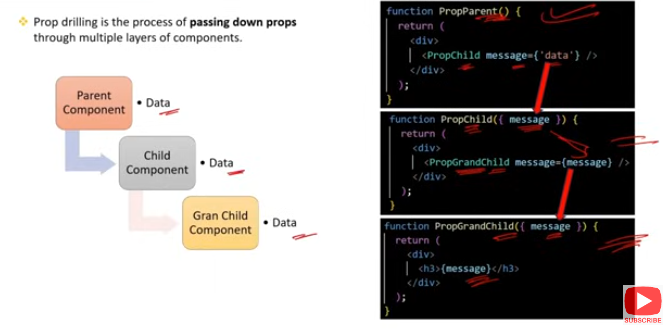
### Steps:

1. **Define Data in Parent**: In ParentComponent, define the data you want to pass.
2. **Pass Data via Props**: In the JSX, use a custom attribute to pass the data as props (e.g., message={data}).
3. **Access Props in Child**: In ChildComponent, access the passed data via props (e.g., props.message).

### Additional Methods:

1. **Callback Functions**: Pass functions as props to handle actions in the child and update the parent state.
2. **Context API**: Share data globally without explicitly passing props at each level.
3. **State Management Libraries**: Use libraries like Redux for complex state management and data flow.

Q4)What are Prop Drilling in React?

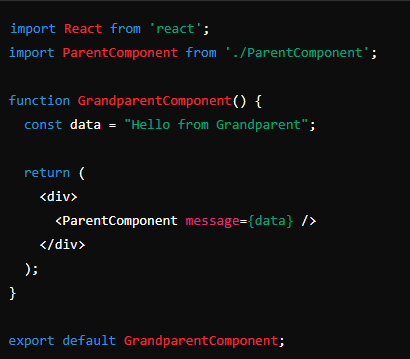


### Prop Drilling in React

**Prop Drilling** refers to the process of passing data from a parent component down to child components through multiple layers of intermediate components. This can lead to cumbersome code, especially in large applications with deep component hierarchies.

### Example of Prop Drilling:

#### Grandparent Component:



Parent Component:



Child Component:



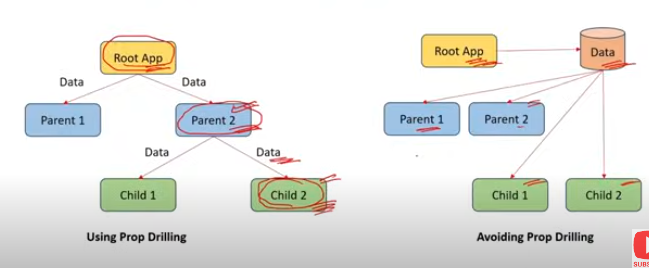
### Problems with Prop Drilling:

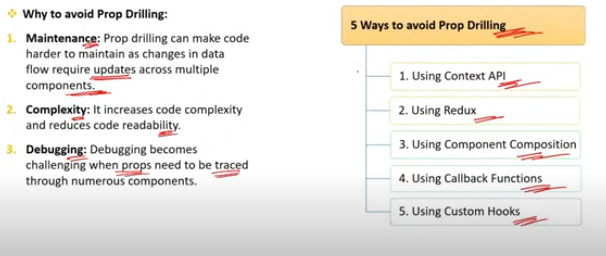
1. **Verbose and Repetitive Code**: Passing the same prop through many layers is repetitive.
2. **Tight Coupling**: Components become tightly coupled and harder to maintain or refactor.
3. **Scalability Issues**: Managing state and props becomes challenging as the application grows.

### Solutions to Avoid Prop Drilling:

1. **Context API**: Provides a way to pass data through the component tree without having to pass props down manually at every level.
2. **State Management Libraries**: Tools like Redux, MobX, or Recoil can manage the state globally and avoid excessive prop drilling.

Q5)Why to avoid Prop Drilling ?In how many ways can avoid Prop Drilling ?





### Why to Avoid Prop Drilling

1. **Verbose and Repetitive Code**: Manually passing props through multiple layers is repetitive and increases the codebase size unnecessarily.
2. **Maintainability**: Tight coupling between components makes refactoring and maintaining the code difficult.
3. **Scalability Issues**: As the application grows, managing state and props becomes increasingly challenging and error-prone.
4. **Readability**: Prop drilling can make the code harder to read and understand, as it's not immediately clear which props are used where.

### Ways to Avoid Prop Drilling

**Context API**:

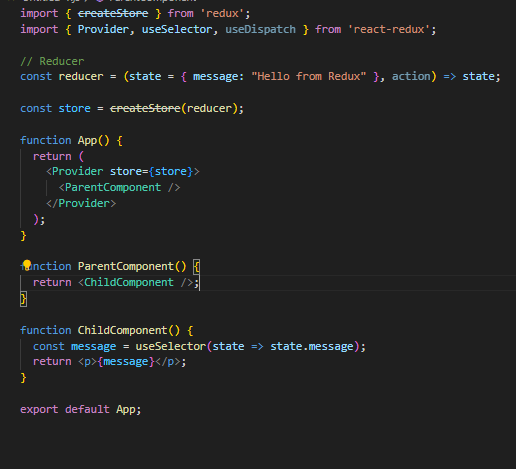
* 1. Allows sharing data across multiple components without explicitly passing props through every level.
  2. Ideal for global settings, themes, user data, etc.



**State Management Libraries**:

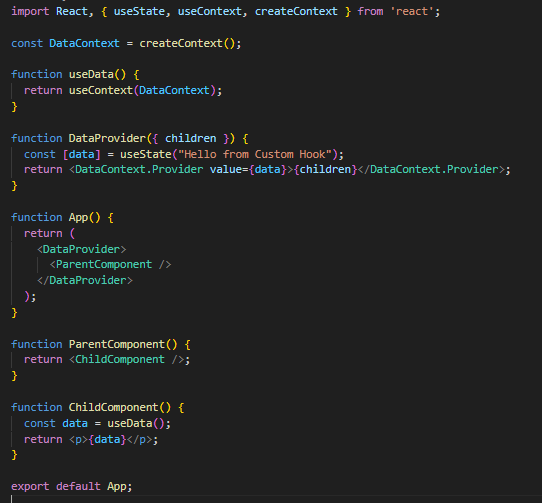
* **Redux**: Centralized state management library that can manage the state of your entire application.
* **MobX**: Simple, scalable state management library with observables.
* **Recoil**: State management library developed by Facebook specifically for React.

**Redux Example**:

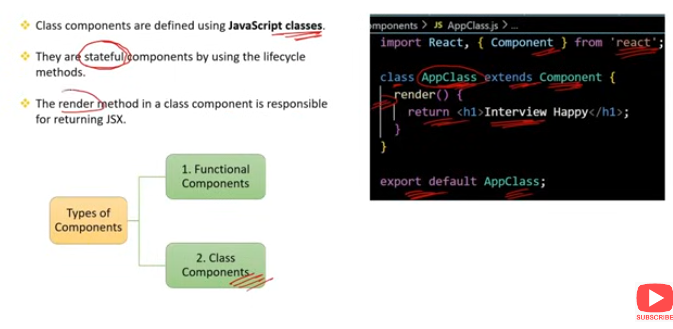


**Custom Hooks**:

* Create custom hooks to encapsulate and reuse stateful logic across multiple components without prop drilling.



Q6)What are class components ?



### Class Components in React

**Class Components** are ES6 classes that extend from React.Component and can have state and lifecycle methods.

### Characteristics of Class Components:

1. **State**: Can hold and manage their own state using this.state.
2. **Lifecycle Methods**: Have built-in lifecycle methods (e.g., componentDidMount, componentDidUpdate, componentWillUnmount) for managing component lifecycle events.
3. render() **Method**: Must implement a render() method that returns JSX.

### Example of a Class Component:



### Key Features:

1. **Constructor**: Used for initializing state and binding methods.
2. this.state: Stores the component's state.
3. this.setState: Updates the component's state and triggers a re-render.
4. **Lifecycle Methods**: Methods like componentDidMount, componentDidUpdate, and componentWillUnmount allow executing code at specific points in a component's lifecycle.

### Lifecycle Methods Overview:

* componentDidMount: Called after the component is rendered for the first time.
* componentDidUpdate: Called after the component updates (re-renders).
* componentWillUnmount: Called right before the component is removed from the DOM.

Q7)How to pass data between class components ?

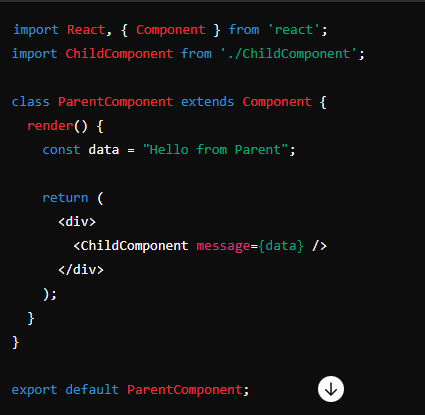


### Passing Data Between Class Components in React

To pass data between class components, you typically use **props** for parent-to-child communication and **callback functions** or **state management solutions** for child-to-parent or sibling-to-sibling communication.

### Example of Passing Data from Parent to Child:

#### Parent Component:

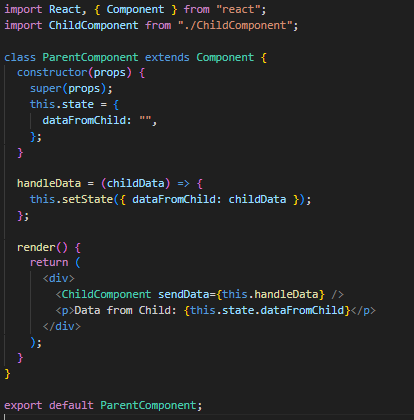


Child Component:

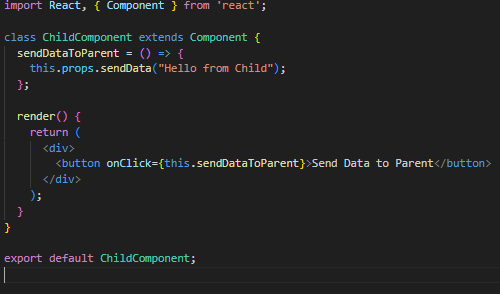


### Passing Data from Child to Parent:

#### Parent Component:



Child Component:

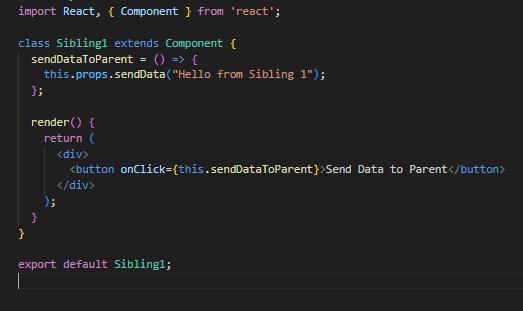


### Passing Data Between Sibling Components:

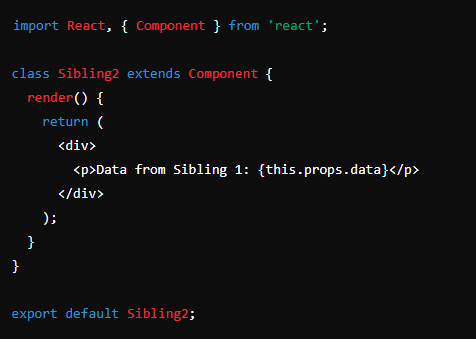
To pass data between sibling components, the data must go up to the parent and then be passed down to the other sibling.

#### Parent Component:



Sibling 1 Component:  


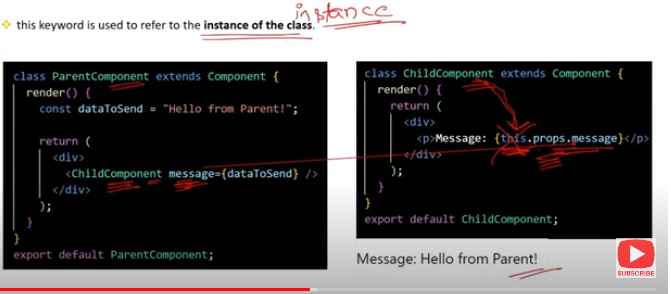
Sibling 2 Component:



### Summary

* **Parent to Child**: Pass data through props.
* **Child to Parent**: Use callback functions to send data back up to the parent.
* **Sibling to Sibling**: Use the parent component as an intermediary to manage and pass data between siblings.

Q8)What is the role of this keyword in class components ?



### The Role of the this Keyword in Class Components

In React class components, the this keyword refers to the current instance of the component. It is essential for accessing component properties and methods.

### Common Uses of this in Class Components:

**Accessing Props and State**:

* 1. this.props: Accesses the component's props.
  2. this.state: Accesses the component's state.

**Calling Methods**:

* 1. Allows invoking methods defined within the class component.

**Binding Event Handlers**:

* 1. Ensures the correct context when calling event handlers.

Example:



### Key Points:

**Accessing Props and State**:

* 1. this.props: Access the properties passed to the component.
  2. this.state: Access and update the component's internal state.

**Binding Event Handlers**:

* 1. In ES6 classes, methods are not bound by default. You need to bind them in the constructor.
  2. Alternatively, use arrow functions for event handlers to avoid manual binding.

**Using Methods**:

* 1. Invoke methods using this.methodName(), ensuring they are defined and bound correctly.

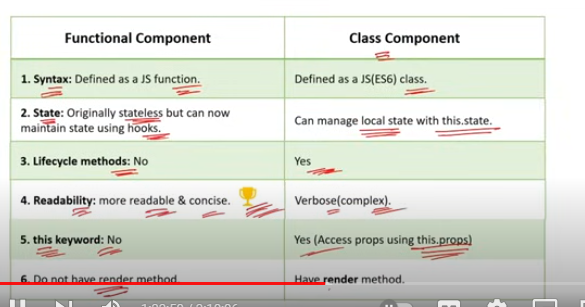
### Example of Binding with Arrow Functions:



### Summary:

* this is used to access props, state, and methods within a class component.
* **Binding** event handlers is crucial to ensure the correct this context.
* **Arrow functions** provide an alternative to binding in the constructor.

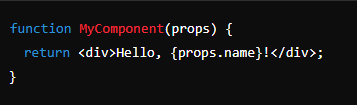
Q9)What are the 5 different between Functional components and class components?



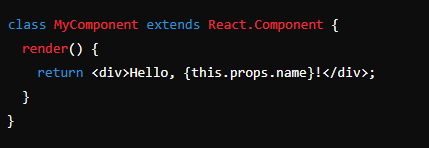
### 5 Differences Between Functional Components and Class Components

**Syntax and Structure**:

* 1. **Functional Components**: Defined as plain JavaScript functions. They are simpler and easier to read.

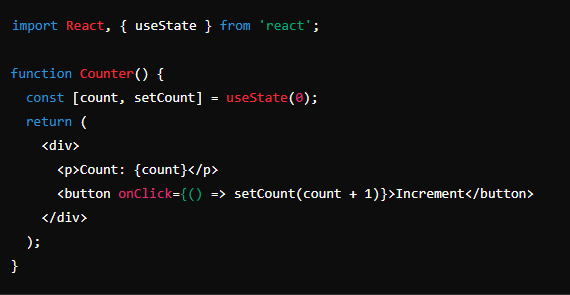


**Class Components**: Defined as ES6 classes that extend React.Component. They require more boilerplate code.

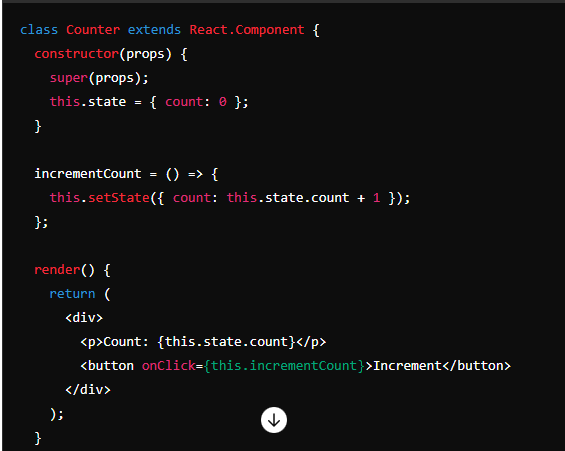


**State Management**:

* **Functional Components**: Use React hooks (e.g., useState, useEffect) for managing state and side effects.



**Class Components**: Manage state using this.state and this.setState.



**Lifecycle Methods**:

* **Functional Components**: Use hooks like useEffect to handle lifecycle events

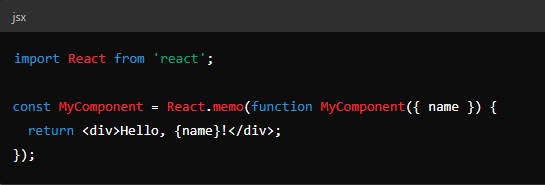


**Class Components**: Use built-in lifecycle methods such as componentDidMount, componentDidUpdate, and componentWillUnmount.

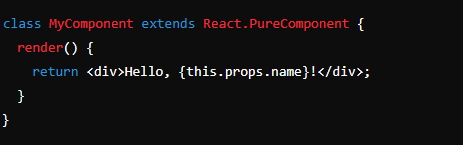


**Performance and Optimization**:

* **Functional Components**: Can benefit from hooks like React.memo for memoization to prevent unnecessary re-renders.



**Class Components**: Use shouldComponentUpdate or PureComponent for optimization



**Ease of Testing**:

* 1. **Functional Components**: Generally easier to test due to their simplicity and lack of this context.
  2. **Class Components**: Require more setup for testing, especially when dealing with state and lifecycle methods.

### Summary:

1. **Syntax and Structure**: Functions vs. ES6 classes.
2. **State Management**: Hooks (useState) vs. this.state.
3. **Lifecycle Methods**: Hooks (useEffect) vs. built-in methods.
4. **Performance and Optimization**: React.memo vs. PureComponent or shouldComponentUpdate.
5. **Ease of Testing**: Functional components are typically easier to test.