Class 06 – React JS

COMPONENTS IN REACT

MERN STACK DEVELOPMENT - MODULE B

Component

In React, a component is a reusable, self-contained piece of code that defines a specific part of a user interface. Components are the building blocks of React applications and allow developers to break down the UI into smaller, manageable pieces. Each component can have its own logic, state, and properties.

There are two main types of components in React:

1. **Functional Components:**
   Also known as stateless components, these are simple functions that take in props (properties) as arguments and return React elements. They don't have their own internal state.

2. **Class Components:**
   Also known as stateful components, these are ES6 classes that extend from React.Component. They can have their own internal state and lifecycle methods.

Define a Component

React component can be defined by using an arrow function or with function keyword for a functional component. Here's an example:

example

```
import React from 'react';
const MyComponent = () => {
    return (
        <div>
            <h1>Hello, World!</h1>
            <p>This is a functional component using an arrow function.</p>
        </div>
)}

export default MyComponent;
```

Props in React Components:

Passing and Utilizing Data in Components

In React, props (is short for "properties,“) are a mechanism for passing data from a parent component to a child component. They are **immutable** and are used to configure and customize a component when it is created.

In other words, it allows user to pass data from one component to another. Props are a way to communicate and share information between React components.

Passing Data: Props enable you to pass data from a parent component to a child component. This data can include various types, such as strings, numbers, objects, or even functions.

Immutable: Props are immutable, meaning they cannot be modified by the child component that receives them. They are read-only and serve as a way to transmit information from one component to another in a unidirectional flow.

Accessing Props: In a functional component, you can access props as a parameter of the function.

Passing Props (parent component):

In the parent component, passing data to a child component by including attributes in the JSX tag corresponding to the child component.

example

```
import React from 'react';
import Child_Component from './Child_Component';   // assuming file path

const App = () => {
    const message = "Hello from Parent Component!";
    return (
        <div>
            <Child_Component  message_from_parent={message} />
        </div>
)}

export default ParentComponent;
```

Receiving Props (child component):

In the child component, receiving data from parent component and use the passed data through the props parameter.

example

```
import React from 'react';
const Child_Component = (props) => {
    return (
        <div>
            <p> Render message: {props.message_from_parent} </p>
        </div>
    )
}

export default Child_Component
```

State: A Component's Memory

In React.js, "state" refers to an object that represents the current condition or state of a component. It is used to store and manage dynamic data within a component and allows the component to re-render when the state changes. State is a crucial concept in React, enabling the creation of dynamic and interactive user interfaces.

Components often need to change what’s on the screen as a result of an interaction. Typing into the form should update the input field, clicking “next” on an image carousel should change which image is displayed, clicking “buy” should put a product in the shopping cart. Components need to “remember” things: the current input value, the current image, the shopping cart. In React, this kind of component-specific memory is called state.

Reference docs: https://react.dev/learn/state-a-components-memory

useState Hook in React

In React, the useState hook is part of the Hooks API, introduced in React 16.8. It allows functional components to manage state without the need for a class. The useState hook returns an array with two elements: the current state value and a function that allows you to update that state.

syntax

```
const [state, setState] = useState(initialState);
```

state:

The current state value.

setState:

A function that allows you to update the state.

Mutable & Immutable

Immutable (Primitive Values)	Mutable (Everything Else)
undefined	Object
Boolean	Array
Number	Map
String	Set
BigInt	Date
Symbol	Function
null	Almost everything made with 'new' keyword

Mutable / Mutability

the concept of mutability refers to the ability of an object to be altered after its creation. Mutable objects, such as arrays and objects, allow modifications to their state, enabling operations like adding or removing elements from an array or changing properties in an object. For example, you can push a new element into an array or update a property in an object directly.

Immutable / Immutability

immutability in JavaScript signifies that an object cannot be changed once it's been created. Immutable objects, like strings and numbers, maintain their original state throughout their lifecycle. Operations that seem to modify immutable objects actually create new instances with the desired changes, leaving the original object unchanged. This characteristic ensures predictability and facilitates reasoning about code, particularly in functional programming, where immutability is often favored for its advantages in avoiding side effects and promoting a more straightforward programming model.

Hooks in React JS

React Hooks: Functions introduced in React 16.8 to provide state and lifecycle features in functional components, eliminating the need for class components. Key hooks include **useState** for managing state and **useEffect** for handling side effects. Custom hooks allow for the creation of reusable logic. Follows "Rules of Hooks" to ensure correct usage.

Some more frequently used hooks in React are as follows:

1. useRef
2. useMemo
3. useCallback
4. useContext

That's it for today!

Keep coding, stay curious, and enjoy your JavaScript journey!