React

1. Define JSX

JSX (JavaScript XML) is a syntax extension for JavaScript commonly used with React. It allows developers to write HTML-like code directly within JavaScript, making it easier to create and visualize the structure of the UI components. JSX improves code readability and maintains the logic and markup in one place. Although browsers do not understand JSX natively, it is transpiled into regular JavaScript using tools like Babel before execution.

For example:

const element = <h1>Hello, World!</h1>;

The above JSX code is converted by Babel into:

const element = React.createElement('h1', null, 'Hello, World!');

---

2. Explain about ECMA Script

ECMAScript (ES) is a standardized scripting language specification that JavaScript adheres to. Maintained by ECMA International, the ECMAScript specification (currently at versions like ES6, ES7, etc.) ensures uniformity in scripting languages across different platforms.

ECMAScript 6 (ES6) introduced several features crucial to modern JavaScript development, including:

- let and const for block scoping
- Arrow functions
- Template literals
- Default parameters
- Destructuring assignment
- Promises
- Classes and modules

React heavily relies on ES6+ features for writing clean and efficient code. Understanding ECMAScript is essential for writing robust and maintainable React applications.

---

3. Explain React.createElement()

The React.createElement() method is a core part of React that creates a virtual DOM node. It is used under the hood when JSX is compiled. This function takes three arguments:

```javascript
React.createElement(
  type,          // type of HTML element or component (e.g., 'div', 'h1', MyComponent)
  [props],       // optional properties (attributes)
  [...children]  // optional child elements or text content
)
```

Example:

const element = React.createElement('h1', { className: 'greeting' }, 'Hello, world!');

This code creates a virtual DOM node representing:

<h1 class="greeting">Hello, world!</h1>

---

4. Explain how to create React nodes with JSX

JSX allows developers to create **React nodes** (virtual DOM elements) using a syntax that resembles HTML. These nodes are JavaScript objects that describe what React should render in the UI.

Example:

const element = This is a paragraph node created with JSX. ;

React nodes can be nested, contain child elements, or even reference JavaScript variables and expressions:

```javascript
const name = 'John';
const greeting = <h1>Hello, {name}</h1>;
```

The React compiler (like Babel) translates these JSX statements into calls to React.createElement() during the build process.

---

5. Define how to render JSX to DOM

To render JSX to the actual DOM (the browser's UI), React provides the ReactDOM.render() method. This method mounts the React component or element into a specific DOM container.

Syntax:

ReactDOM.render(element, container);

Where:

- element is a JSX expression or React component
- container is a DOM element (typically obtained via document.getElementById())

Example:

```javascript
const element = <h1>Welcome to React</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

This renders the <h1> element into the DOM node with the id root.

---

6. Explain how to use JavaScript expressions in JSX

JSX supports embedding JavaScript expressions within curly braces {}. This feature allows dynamic content rendering inside JSX.

Valid expressions include:

- Variables
- Function calls
- Arithmetic operations
- Ternary operators
- Object properties

Example:

```javascript
const user = { name: 'Alice', age: 25 };
const element = <p>{user.name} is {user.age} years old.</p>;
```

Conditional rendering can also be achieved:

```javascript
const isLoggedIn = true;
const message = <h2>{isLoggedIn ? 'Welcome back!' : 'Please log in.'}</h2>;
```

Note: Statements like if or for are not valid inside JSX directly; use them outside the return or wrap logic within helper functions.

---

7. Explain how to use inline CSS in JSX

In JSX, inline styles are defined as JavaScript objects. Each CSS property is written in camelCase rather than the traditional kebab-case. The style object is passed to the style attribute.

Example:

```javascript
const styleObject = {
  backgroundColor: 'lightblue',
  padding: '10px',
  fontSize: '16px'
};

const element = <div style={styleObject}>Styled with inline CSS</div>;
```

Alternatively, you can define the styles inline:

```javascript
const element = (
  <div style={{ color: 'red', fontWeight: 'bold' }}>
    This text is red and bold.
  </div>
);
```