1. What is React?

React is an open-source JavaScript library developed by Facebook for building user interfaces, especially for single-page applications (SPAs). It allows developers to create reusable UI components, making code more modular and easier to maintain. React follows a declarative programming style and uses a virtual DOM to optimize rendering and improve performance.

2. What are Components in React?

Components are the building blocks of any React application. They allow you to split the UI into independent, reusable pieces. There are two types:

- **Functional Components**: These are plain JavaScript functions that return JSX.

```
function Greeting() {
  return <h1>Hello, world!</h1>;
}
```

- **Class Components**: These use ES6 classes and can hold local state and lifecycle methods.

```
class Greeting extends React.Component {
  render() {
    return <h1>Hello, world!</h1>;
  }
}
```

Vasanta$7

3. What is JSX?

JSX stands for JavaScript XML. It allows developers to write HTML elements in JavaScript and place them in the DOM without using `createElement()` or `appendChild()` methods. JSX is not required to write React, but it makes the code easier to understand and write.

```
const element = <h1>Hello, JSX!</h1>;
```

JSX gets transpiled to React.createElement calls using tools like Babel.

4. What is the Virtual DOM?

The Virtual DOM is a lightweight JavaScript representation of the actual DOM. React keeps this virtual DOM in memory and compares it with a previous snapshot (diffing). When changes are found, React calculates the most efficient way to update the real DOM. This process is known as reconciliation and it improves performance.

5. What are Props and State in React?

- **Props** (short for properties): These are read-only attributes used to pass data from a parent to a child component.
- **State**: This is data that changes over time and determines how the component behaves.

Props:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

State:

```
class Counter extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };
  }

  render() {
    return <p>Count: {this.state.count}</p>;
  }
}
```

Interview Tip: Props are passed from parent to child, while state is local to the component. Props are immutable from the child's perspective.

6. What is `useState()` Hook?

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

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

This returns an array with the current state and a function to update it.

Example with Input:

```
const [name, setName] = useState('');
<input value={name} onChange={(e) => setName(e.target.value)}
/>
```

Interview Tip: State updates via `useState` are asynchronous and trigger re-renders.

7. What is `useEffect()` Hook?

`useEffect` allows you to perform side effects in function components. This includes data fetching, subscriptions, or manually changing the DOM.

```
useEffect(() => {
  console.log('Component mounted');
  return () => console.log('Component unmounted');
```

```
}, []);
```
The second argument (dependency array) controls when the effect runs.

11. What is Context in React?

Context provides a way to pass data through the component tree without having to pass props manually at every level.

```
const ThemeContext = React.createContext('light');
```
You use a Provider to set the context and a Consumer or `useContext` to read it.

12. Why are Keys important in React lists?

Keys help React identify which items have changed, been added, or removed. They should be stable and unique to each item in the list.

```
items.map(item => <li key={item.id}>{item.name}</li>)
```

13. What is React Testing Library (RTL)?

React Testing Library encourages testing components the way users interact with them. Instead of checking internal state or props, it tests behavior.

```
render(<Button />);
expect(screen.getByText(/click me/i)).toBeInTheDocument();
```

14. What is React.StrictMode?

`<React.StrictMode>` is a development tool that highlights potential problems in your application. It does things like:

- Double-invokes certain lifecycle methods
- Warns about deprecated APIs
- Helps prepare your app for future React versions

15. What is `startTransition` in React 18?

`startTransition` is used to mark state updates as non-urgent. It lets React keep the interface responsive while handling heavy updates.

```
startTransition(() => {
  setData(expensiveComputation());
```

```
});
```

16. What's the difference between `ReactDOM.render()` and `createRoot()`?

In React 18, `ReactDOM.render()` is deprecated. Use `createRoot()` to enable concurrent features.

```
const root = createRoot(document.getElementById('root'));
root.render(<App />);
```

17. How does routing work in React using React Router?

React Router is a popular library used for client-side routing in React applications. It allows you to define multiple routes in your application and map them to components.

```
import { BrowserRouter, Routes, Route } from 'react-router-
dom';

<BrowserRouter>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
  </Routes>
</BrowserRouter>
```

It uses the History API to manipulate the URL without a full page reload.

18. How do you fetch data from an API in React?

You can fetch data using the built-in `fetch` API or using libraries like Axios. Usually, this is done inside a `useEffect` hook.

```
useEffect(() => {
  const fetchData = async () => {
    try {
      const response = await fetch('https://api.example.com/
data');
      const json = await response.json();
      setData(json);
    } catch (error) {
      console.error('Error fetching data:', error);
    }
  };

  fetchData();
}, []);
```

This example uses `async/await` for cleaner asynchronous logic and includes error handling.

19. What are custom hooks in React?

Custom hooks are JavaScript functions whose names start with "use" and that may call other hooks. They help in reusing stateful logic across components.

```
function useCounter(initialValue) {
  const [count, setCount] = useState(initialValue);
  const increment = () => setCount(c => c + 1);
  return [count, increment];
}
```

You can use it in a component like:

```
const [count, increment] = useCounter(0);
```

20. How are events handled in React?

React handles events using camelCase syntax and attaches events directly to components, not to DOM elements.

Example:

```
function MyButton() {
  const handleClick = () => {
    alert('Button clicked');
  };

  return <button onClick={handleClick}>Click Me</button>;
}
```

React normalizes events using its synthetic event system, which wraps around the browser's native events for cross-browser compatibility.

Let me know if you'd like to expand this into a mock interview format or get a PDF version!