Below are **very long, clear, exam-oriented answers** for **all 10 React questions**, written in a **Web Technologies / LPU / MCA** style. You can directly use these for **10–15 mark answers**.

---

1. What is a Component? Explain its Types

A **component** is the fundamental building block of a React application. It represents a **reusable piece of the user interface** that controls its own structure, logic, and behavior. Components help divide a large UI into smaller, manageable, and reusable parts.

In React, every UI element such as buttons, forms, headers, and pages can be created as components.

Types of Components in React:

(a) Functional Components

Functional components are simple JavaScript functions that return JSX.

Features:

- Written as functions
- No this keyword
- Use **Hooks** to manage state and lifecycle
- Faster and simpler

Example:

```
function Welcome() {
  return <h1>Welcome to React</h1>;
}
```

(b) Class Components

Class components are ES6 classes that extend React.Component.

Features:

- Use this
- Can use lifecycle methods
- Use state directly

Example:

```
class Welcome extends React.Component {
  render() {
    return <h1>Welcome to React</h1>;
  }
}
```

---

2. Explain how lists work in React

Lists in React are used to **display multiple similar items dynamically**. React uses the JavaScript map() function to render lists.

Each element in a list must have a **unique key** to help React identify which items have changed.

Example:

```
const fruits = ["Apple", "Banana", "Mango"];


function FruitList() {
  return (
    <ul>
      {fruits.map((fruit, index) => (
        <li key={index}>{fruit}</li>
      ))}
    </ul>
  );
}
```

Importance of key:

- Improves performance
- Helps React track element updates
- Prevents rendering issues

---

3. How do you create forms in React? Explain with example

Forms in React are used to **collect user input**. React usually uses **controlled components**, where form data is managed by React state.

Example of a controlled form:

```
function LoginForm() {
  const [name, setName] = React.useState("");


  function handleSubmit(e) {
    e.preventDefault();
    alert("Name: " + name);
  }
```

```
  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        value={name}
        onChange={e => setName(e.target.value)}
      />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Explanation:

- value is controlled by state
- onChange updates state
- onSubmit handles form submission

---

4. How do you implement state in React? How do you update the state of a component?

State is an object that stores data which can change over time and affects the component rendering.

In Functional Components (using Hooks):

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


setCount(count + 1);
```

In Class Components:

```
this.state = { count: 0 };


this.setState({ count: this.state.count + 1 });
```

Key points:

- State changes cause re-rendering
- State should not be modified directly
- Always use setState() or state updater function

5. How can you embed two or more components into one?

Components can be embedded by **calling one component inside another**.

Example:

```
function Header() {
  return <h1>Header</h1>;
}

function Footer() {
  return <h1>Footer</h1>;
}

function App() {
  return (
    <div>
      <Header />
      <Footer />
    </div>
  );
}
```

Explanation:

- App is the parent component
- Header and Footer are child components
- This supports component reuse and modular design

---

6. What do you mean by State Management? Explain its components

State Management is the process of managing data (state) across components in a React application.

Components of State Management:

1. **State** – Application data
2. **Actions** – Events that change state
3. **Reducers** – Logic to update state
4. **Store** – Central place to hold state

Why State Management is needed:

- Avoids prop drilling
- Maintains consistency
- Improves scalability

Examples of state management tools:

- React Context API
- Redux
- Zustand

---

7. How to handle forms? Explain the creation of forms

Handling forms in React involves:

- Capturing user input
- Updating state
- Handling submission

Steps:

1. Create state variables
2. Bind input values to state
3. Handle change events
4. Handle submit event

Example:

```
function FormExample() {
  const [email, setEmail] = React.useState("");

  return (
    <form>
      <input
        type="email"
        value={email}
        onChange={e => setEmail(e.target.value)}
      />
    </form>
  );
```

}

Forms in React provide better control and validation.

---

8. What are the differences between state and props?

State	Props
Local to component	Passed from parent
Mutable	Immutable
Managed within component	Read-only
Can change	Cannot be changed by child

Example:

```
function Child(props) {
  return <h1>{props.name}</h1>;
}
```

State is used for internal data, props for communication.

---

9. What are the differences between class and functional components?

Class Component	Functional Component
Uses ES6 class	Uses functions
Has lifecycle methods	Uses hooks
Uses this	No this
More boilerplate	Cleaner code

Functional components are preferred in modern React.

---

10. Explain the lifecycle methods of components

Lifecycle methods define **different phases of a component's life**.

Phases:

1. Mounting:

- constructor()
- render()
- componentDidMount()

2. Updating:

- shouldComponentUpdate()
- render()
- componentDidUpdate()

3. Unmounting:

- componentWillUnmount()

Functional Component Alternative:

```
useEffect(() => {
  // componentDidMount
  return () => {
    // componentWillUnmount
  };
}, []);
```

Lifecycle methods help control behavior at different stages.

If you want:

- **Short 5-mark answers**
- **MCQs**
- **React Hooks deep dive**
- **Full React revision notes**

just tell me 💪😊