1. What is React?

- **React** is a JavaScript library for building user interfaces, primarily for single-page applications (SPAs).
- It lets you build components, which are reusable and independent pieces of code that describe a part of the UI.

2. JSX (JavaScript XML)

- JSX is a syntax extension for JavaScript, allowing you to write HTML-like code in your JavaScript files.
- Example:

```
jsx
Copy code
const element = <h1>Hello, world!</h1>;
```

- JSX must be compiled, and React uses a tool like Babel to convert JSX into standard JavaScript.

3. Components

- Components are the building blocks of React applications. They can be either **functional** or **class** components.
- **Functional Components**:

```
jsx
Copy code
function MyComponent() {
  return <h1>Hello, React!</h1>;
}
```

- **Class Components**:

```
jsx
Copy code
class MyComponent extends React.Component {
  render() {
    return <h1>Hello, React!</h1>;
  }
}
```

4. Props (Properties)

- Props are used to pass data from a parent component to a child component.
- Example:

```
jsx
Copy code
```

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
// Usage:
<Welcome name="John" />
```

5. State

- State is used to manage data that can change within a component.
- State is specific to class components or functional components that use hooks like `useState`.
- **State in Class Components**:

```
jsx
Copy code
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }
  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count
+ 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
```

- **State in Functional Components with Hooks**:

```
jsx
Copy code
import React, { useState } from 'react';
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
```

6. Event Handling

- React uses camelCase for event handlers.
- Example of handling a click event:

```
jsx
Copy code
function MyButton() {
  function handleClick() {
    alert('Button clicked!');
  }
  return <button onClick={handleClick}>Click Me</button>;
}
```

7. Conditional Rendering

- React allows you to conditionally render elements based on state or props.
- Example:

```
jsx
Copy code
function Greeting(props) {
  if (props.isLoggedIn) {
    return <h1>Welcome back!</h1>;
  }
  return <h1>Please sign up.</h1>;
}
```

8. Lists and Keys

- To render a list of elements, use JavaScript's `map()` function.
- Each list item should have a unique **key** to optimize rendering:

```
jsx
Copy code
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>{number}</li>
  );
  return <ul>{listItems}</ul>;
}
```

9. useEffect Hook

- `useEffect` is used for side effects in functional components, like fetching data or subscribing to external events.
- Example:

```
jsx
Copy code
import React, { useState, useEffect } from 'react';
function DataFetcher() {
  const [data, setData] = useState(null);
  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(response => response.json())
```

```
      .then(data => setData(data));
  }, []);  // Empty array means this runs once, like
componentDidMount()
  return <div>{data ? JSON.stringify(data) : 'Loading...'}</div>;
}
```

10. Form Handling

- React makes it easy to handle form elements using state to track their values.
- Example:

```
jsx
Copy code
function NameForm() {
  const [name, setName] = useState('');
  function handleChange(event) {
    setName(event.target.value);
  }
  function handleSubmit(event) {
    alert('A name was submitted: ' + name);
    event.preventDefault();
  }
  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={handleChange} />
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
}
```

11. React Router

- React Router helps with navigation between components in a React application, allowing for the creation of SPAs with different views.

12. Context API

- React's Context API allows you to share state between components without having to pass props manually at every level.
- Example:

```
jsx
Copy code
const ThemeContext = React.createContext('light');
function App() {
  return (
    <ThemeContext.Provider value="dark">
      <Toolbar />
    </ThemeContext.Provider>
```

```
  );
}

function Toolbar() {
  return <ThemedButton />;
}

function ThemedButton() {
  return (
    <ThemeContext.Consumer>
      {theme => <button className={theme}>Button</button>}
    </ThemeContext.Consumer>
  );
}
```