1. What is JSX?

JSX stands for JavaScript XML. It is a React extension which allows writing JavaScript code that looks similar to HTML. It makes HTML file easy to understand. The JSX file makes the React application robust and boosts its performance. JSX provides you to write XML-like syntax in the same file where you write JavaScript code, and then preprocessor (i.e., transpilers like Babel) transform these expressions into actual JavaScript code. Just like XML/HTML, JSX tags have a tag name, attributes, and children.

**Example**

1. **class** App **extends** React.Component {
2. render() {
3. **return**(
4. <div>
5. <h1>Hello JavaTpoint</h1>
6. </div>
7. )
8. }
9. }

In the above example, text inside <h1> tag return as JavaScript function to the render function. After compilation, the JSX expression becomes a normal JavaScript function, as shown below.

1. React.createElement("h1", **null**, "Hello JavaTpoint");
2. **What do you understand by Virtual DOM?**

A Virtual DOM is a lightweight JavaScript object which is an in-memory representation of real DOM. It is an intermediary step between the render function being called and the displaying of elements on the screen. It is similar to a node tree which lists the elements, their attributes, and content as objects and their properties. The render function creates a node tree of the React components and then updates this node tree in response to the mutations in the data model caused by various actions done by the user or by the system.

1. What is the difference between ReactJS and React Native?

The main differences between ReactJS and React Native are given below.

|  |  |  |
| --- | --- | --- |
| **SN** | **ReactJS** | **React Native** |
| **1.** | Initial release in 2013. | Initial release in 2015. |
| **2.** | It is used for developing web applications. | It is used for developing mobile applications. |
| **3.** | It can be executed on all platforms. | It is not platform independent. It takes more effort to be executed on all platforms. |
| **4.** | It uses a JavaScript library and CSS for animations. | It comes with built-in animation libraries. |
| **5.** | It uses React-router for navigating web pages. | It has built-in Navigator library for navigating mobile applications. |
| **6.** | It uses HTML tags. | It does not use HTML tags. |
| **7.** | In this, the Virtual DOM renders the browser code. | In this, Native uses its API to render code for mobile applications. |

1. What **is Props?**

Props stand for "Properties" in React. They are read-only inputs to components. Props are an object which stores the value of attributes of a tag and work similar to the HTML attributes. It gives a way to pass data from the parent to the child components throughout the application.

It is similar to function arguments and passed to the component in the same way as arguments passed in a function.

Props are immutable so we cannot modify the props from inside the component. Inside the components, we can add attributes called props. These attributes are available in the component as this.props and can be used to render dynamic data in our render method.

1. What is an event in React?

An event is an action which triggers as a result of the user action or system generated event like a mouse click, loading of a web page, pressing a key, window resizes, etc. In React, the event handling system is very similar to handling events in DOM elements. The React event handling system is known as Synthetic Event, which is a cross-browser wrapper of the browser's native event.

Handling events with React have some syntactical differences, which are:

* React events are named as camelCase instead of lowercase.
* With JSX, a function is passed as the event handler instead of a string.

1. **what** is the difference between controlled and uncontrolled components?

* The difference between controlled and uncontrolled components are:

|  |  |  |
| --- | --- | --- |
| **SN** | **Controlled** | **Uncontrolled** |
| **1.** | It does not maintain its internal state. | It maintains its internal states. |
| **2.** | Here, data is controlled by the parent component. | Here, data is controlled by the DOM itself. |
| **3.** | It accepts its current value as a prop. | It uses a ref for their current values. |
| **4.** | It allows validation control. | It does not allow validation control. |
| **5.** | It has better control over the form elements and data. | It has limited control over the form elements and data. |

1. **How are forms created in React?**

Forms allow the users to interact with the application as well as gather information from the users. Forms can perform many tasks such as user authentication, adding user, searching, filtering, etc. A form can contain text fields, buttons, checkbox, radio button, etc.

React offers a stateful, reactive approach to build a form. The forms in React are similar to HTML forms. But in React, the state property of the component is only updated via setState(), and a JavaScript function handles their submission. This function has full access to the data which is entered by the user into a form

1. **What** is the difference between Element and Component?

The main differences between Elements and Components are:

|  |  |  |
| --- | --- | --- |
| **SN** | **Element** | **Component** |
| **1.** | An element is a plain JavaScript object which describes the component state and DOM node, and its desired properties. | A component is the core building block of React application. It is a class or function which accepts an input and returns a React element. |
| **2.** | It only holds information about the component type, its properties, and any child elements inside it. | It can contain state and props and has access to the React lifecycle methods. |
| **3.** | It is immutable. | It is mutable. |
| **4.** | We cannot apply any methods on elements. | We can apply methods on components. |
| **5.** | **Example:** const element = React.createElement( 'div', {id: 'login-btn'}, 'Login' ) | **Example:** function Button ({ onLogin }) { return React.createElement( 'div', {id: 'login-btn', onClick: onLogin}, 'Login' ) } |

1. What are fragments?

In was introduced in React 16.2 version. In React, Fragments are used for components to return multiple elements. It allows you to group a list of multiple children without adding an extra node to the DOM.

**Example**

1. render() {
2. **return** (
3. <React.Fragment>
4. <ChildA />
5. <ChildB />
6. <ChildC />
7. </React.Fragment>
8. )
9. }

There is also a shorthand syntax exists for declaring Fragments, but it's not supported in many tools:

1. render() {
2. **return** (
3. <>
4. <ChildA />
5. <ChildB />
6. <ChildC />
7. </>
8. )
9. }
10. **What** are Forward Refs?

Ref forwarding is a feature which is used for passing a ref through a component to one of its child components. It can be performed by making use of the React.forwardRef() method. It is particularly useful with higher-order components and specially used in reusable component libraries.

**Example**

1. **import** React, { Component } from 'react';
2. **import** { render } from 'react-dom';
4. **const** TextInput = React.forwardRef((props, ref) => (
5. <input type="text" placeholder="Hello World" ref={ref} />
6. ));
8. **const** inputRef = React.createRef();
10. **class** CustomTextInput **extends** React.Component {
11. handleSubmit = e => {
12. e.preventDefault();
13. console.log(inputRef.current.value);
14. };
15. render() {
16. **return** (
17. <div>
18. <form onSubmit={e => **this**.handleSubmit(e)}>
19. <TextInput ref={inputRef} />
20. <button>Submit</button>
21. </form>
22. </div>
23. );
24. }
25. }
26. export **default** App;