Answer the following

1. **How does JSX differ from HTML?**

Ans –

JSX (JavaScript XML) and HTML (Hypertext Markup Language) share similar syntax, but JSX has several key differences:

- **JavaScript Integration:** JSX allows JavaScript expressions to be embedded directly within the HTML-like code. For example, you can use variables, functions, or JavaScript expressions inside JSX by wrapping them in curly braces {}.

Example:

const name = "John";

const element = <h1>Hello, {name}!</h1>;

In HTML, this kind of dynamic behavior isn't possible without JavaScript.

- **Attributes:** JSX attributes use camelCase instead of the typical lowercase in HTML. For instance, class in HTML becomes className in JSX, for becomes htmlFor, etc.

Example:

<input type="text" className="input-field" />

- **Self-Closing Tags:** In JSX, all tags must be closed, including self-closing tags, which in HTML might be written without a closing slash (e.g., <img>). In JSX, you write them as <img />.
- **Event Handlers:** Event handlers in JSX are written in camelCase. For instance, onClick instead of onclick.

Example:

<button onClick={handleClick}>Click Me</button>

2. **Can browsers understand JSX directly? How is JSX processed?**

Ans –

Browsers cannot understand JSX directly because it is not valid JavaScript. JSX needs to be transformed into regular JavaScript before the browser can execute it. This transformation is typically done by a build tool such as **Babel**, which converts JSX into React.createElement() calls.

For example, JSX:

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

Gets transpiled into JavaScript like:

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

This code is then run by the browser, and the virtual DOM in React is updated accordingly.

3. **What are fragments in JSX and how do you use them?**

Ans –

Fragments are a feature in JSX that allow you to group multiple elements without adding extra nodes to the DOM. They are useful when you need to return multiple elements from a component but don't want to wrap them in a or any other unnecessary container.

There are two ways to use fragments in JSX:

1. **Using the React.Fragment component:**

```
return (
  <React.Fragment>
    <h1>Title</h1>
    <p>Content</p>
  </React.Fragment>
);
```

2. **Using the shorthand syntax (<> and </>):**

```
return (
  <>
    <h1>Title</h1>
    <p>Content</p>
  </>
);
```

Both methods achieve the same result, and the JSX elements inside will not result in any additional wrapping elements in the DOM.

4. **How do you handle inline styles in JSX? Provide an example.**

Ans –

Inline styles in JSX are specified as an object, where the keys are camelCase versions of the CSS properties, and the values are strings. This is different from HTML, where you typically use a string of CSS styles.

Example:

```
const style = {
  color: 'blue',
  backgroundColor: 'lightgrey',
  padding: '10px',
};

const element = <div style={style}>Hello, Styled World!</div>;
```

Alternatively, you can inline the styles directly:

Jsx :

```
const element = <div style={{ color: 'blue', backgroundColor: 'lightgrey', padding: '10px' }}>Hello, Styled World!</div>;
```

5. **What are the rules for writing JSX tags?**

Ans –

One Root Element: JSX must have one root element. If you return multiple elements, they must be wrapped in a single parent, such as a div, a React.Fragment, or the shorthand <> and </>.

Example:

```
return (
  <div>
    <h1>Hello</h1>
    <p>World</p>
  </div>
);
```

- **Self-Closing Tags:** Tags like <img>, <input>,
, and others must be self-closed by adding a / before the closing angle bracket, even if they don't have content.

```
<img src="image.jpg" alt="example" />
```

- **Expressions inside Curly Braces:** JavaScript expressions, such as variables, functions, and operators, should be wrapped in curly braces {} inside JSX.

```
const name = "Alice";
const greeting = <h1>Hello, {name}!</h1>;
```

- **Attributes are CamelCased:** Attributes like class and for are replaced with className and htmlFor respectively.

```
<label htmlFor="input-id">Label</label>
<div className="container"></div>
```

6. **What is the purpose of the key attribute in JSX?**

Ans –

The key attribute is used in React (JSX) when rendering lists of elements. It helps React identify which items in the list have changed, been added, or been removed. This improves the performance of updates because React can avoid re-rendering the entire list and instead apply changes to the specific items that have changed.

Example:

```
const list = ['Apple', 'Banana', 'Cherry'];


const items = list.map((item, index) =>
  <li key={index}>{item}</li>
);


return <ul>{items}</ul>;
```

In this example, each li element has a unique key based on its index, which allows React to efficiently manage updates when the list changes.