REACT_ASSIGNMENT_4

1. What is JSX?

JSX stands for JavaScript XML. It is an extension of JavaScript syntax that allows you to write HTML-like code within JavaScript.

2. Superpowers of JSX?

JSX allow us to write HTML like code inside js due to which our react code is more readable.

JSX enables seamless integration of JavaScript code within the markup. You can embed JavaScript expressions within curly braces `{}` to dynamically generate content, calculate values, iterate over data, or apply conditional rendering.

when used with a library like React, allows you to create a component-based architecture for building user interfaces. Components are the building blocks of your application's UI, encapsulating their own logic, rendering, and behavior. Here's an example of a component-based architecture using JSX:

```
// Parent Component
function App() {
  return (
    <div>
      <h1>Welcome to My App!</h1>
      <ChildComponent />
    </div>
  );
}

// Child Component
function ChildComponent() {
  return <p>This is a child component.</p>;
}
```

3. Role of Script tag?

Different Scripting Languages: The `type` attribute can be used to specify a scripting language other than JavaScript. For example, if you are using server-side scripting languages like PHP or Perl within your script block, you can indicate the appropriate MIME type using the `type` attribute.

```
<script type="text/php">
  // PHP code here
</script>
```

1. Browser Support: In older versions of HTML, the `type` attribute was required, and it was used to explicitly specify the scripting language. Although it is not necessary in HTML5, some developers still include it for backward compatibility purposes.

It's important to note that for JavaScript code, you can omit the `type` attribute altogether, as the default value of `"text/javascript"` will be assumed. Modern browsers typically treat script blocks without a `type` attribute or with a value of `"text/javascript"` as JavaScript code.

3. diff between html and jsx?

Syntax: HTML uses plain HTML tags and attributes, while JSX is a syntax extension for JavaScript that allows you to write HTML-like code within your JavaScript files.

Embedding JavaScript: JSX allows you to embed JavaScript expressions directly within the markup using curly braces {}. This allows you to dynamically generate content or perform calculations directly within the JSX code.

1. Components: In HTML, you have predefined tags like `<div>`, `<span>`, `<h1>`, etc. In JSX, you can create your own custom components using JavaScript. Components in JSX are functions or classes that return JSX elements. This component-based approach is one of the core features of React.

Styling: In HTML, you typically apply styles using CSS classes and inline styles. In JSX, you can use CSS classes and inline styles as well, but you write them as JavaScript objects within the `style` attribute

Event Handling: In HTML, you attach event handlers using attributes like `onclick` or `onchange`. In JSX, you attach event handlers using camelCase event names, such as `onClick` or `onChange`. The event handler functions are typically defined as methods within a component.

Comments: In HTML, you can add comments using `<!-- ... -->`. In JSX, you can add comments using curly braces and double slashes `{/* ... */}` or multi-line block comments `{/* ... */}`.

Self-Closing Tags: In HTML, certain tags require a closing tag (e.g., `<div></div>`), while others can be self-closed (e.g., `<img src="..." />`). In JSX, you can use self-closing tags for all elements, even those that require a closing tag in HTML.

4. {TitleComponent} vs {<TitleComponent/>} vs {<TitleComponent> </TitleComponent>} in JSX?

1. `{TitleComponent}`: This syntax refers to the component itself without rendering it. It is typically used when passing components as props or assigning them to variables. For example:

```
const MyComponent = () => {
  const title = TitleComponent; // Assigning component to a variable
  return <div>{title}</div>;
};
```

1. `{<TitleComponent/>}`: This syntax directly renders the component using self-closing tags. It is equivalent to `{React.createElement(TitleComponent)}`. This is commonly used when rendering a component directly within JSX. For example:

```
const Heading=()=>{
  return (
    <div>

      {{sum:10}}
      <h1 id="h1" key="h1_el">this is first heading using JSX</h1>
      <h1 id="h1" key="h1_el_s">this is second heading using JSX</h1>
      </div>


  );
}


const root = ReactDOM.createRoot(rootElement);
root.render(<Heading/>);
```

1. `{<TitleComponent></TitleComponent>}`: This syntax also renders the component, but using opening and closing tags. It is equivalent to `{React.createElement(TitleComponent)}`. This form is used when you want to pass child elements or properties to the component. For example:

```
const MyComponent = () => {
  return (
    <div>
      <TitleComponent>
        <span>Hello</span>
      </TitleComponent>
    </div>
  );
};
```