**Hands-on: 10. ReactJS-HOL**

**Introduction**

React is a powerful JavaScript library used for building user interfaces, especially single-page applications. One of the key concepts in React is JSX, which enhances how developers write components and manage dynamic rendering. This document outlines important React concepts including JSX, React.createElement(), DOM rendering, expressions in JSX, and styling techniques.

1. **Define JSX**

JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like code within JavaScript. It is used extensively in React to describe UI components.

* Example:

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

JSX makes code more readable and declarative. It is not valid JavaScript on its own—Babel or similar tools transpile JSX into standard JavaScript (usually into React.createElement() calls).

1. **Explain ECMA Script**

ECMAScript is the standardized version of JavaScript maintained by ECMA International. Each version (like ES5, ES6/ES2015, ES7, etc.) adds new features to the language.

* ES6 (ECMAScript 2015) introduced major features like let, const, arrow functions, classes, promises, template literals, and destructuring.
* React applications often rely on ES6+ features to create modern, readable, and modular code.

1. **Explain React.createElement()**

Before JSX, the UI elements were created using React.createElement():

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

* This method:
* Takes three arguments: the element type, props, and children.
* Returns a React element (a plain JavaScript object).
* JSX is syntactic sugar for React.createElement().
* Example with JSX:

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

// becomes:

React.createElement('h1', null, 'Hello World');

1. **Creating React Nodes with JSX**

You can create React nodes using JSX by embedding them inside components or returning them from functions:

function Welcome() {

return <h2>Welcome to React!</h2>;

}

const element = <Welcome />;

* React nodes can represent:
* HTML elements
* Custom components
* Fragments and nested components

1. **Rendering JSX to the DOM**

To render JSX into the browser DOM, use ReactDOM.createRoot() and .render() (in React 18+):

import React from 'react';

import ReactDOM from 'react-dom/client';

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

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(element);

This renders the element into the DOM element with id "root".

1. **Using JavaScript Expressions in JSX**

* You can embed any JavaScript expression inside JSX using curly braces {}.

const name = "Anshika";

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

* You can also use functions, operators, and ternary conditions:

const age = 20;

const message = <p>{age >= 18 ? "Adult" : "Minor"}</p>;

Note: You can't use statements (like if or for) directly inside JSX.

1. **Using Inline CSS in JSX**

In JSX, inline styles are written as JavaScript objects, not strings. The keys are written in camelCase instead of kebab-case.

const styleObj = {

color: "blue",

backgroundColor: "lightgray",

fontSize: "20px"

};

const element = <h2 style={styleObj}>Styled Heading</h2>;

Or directly inline:

<h2 style={{ color: "green", fontWeight: "bold" }}>Hello</h2>

**Conclusion**

Understanding JSX and how it interacts with core JavaScript features is crucial when working with React. JSX provides a cleaner, declarative way to build UI elements, while React.createElement() offers a programmatic alternative under the hood. From embedding expressions to rendering components and styling with inline CSS, mastering these foundational concepts enables efficient and readable React development.