**Hands-on: 2. ReactJS-HOL**

**React Components: Building Blocks of User Interfaces**

In React, components are the fundamental building blocks for creating dynamic and reusable user interfaces. A clear understanding of how components operate, their various types, and their distinctions from regular JavaScript functions is crucial for developing scalable React applications.

**What are React Components?**

A React component is essentially a self-contained, reusable block of code that represents a specific part of the user interface. These components receive input, known as props, and then return React elements that dictate what should be displayed on the screen. They are instrumental in breaking down complex UIs into smaller, more manageable pieces.

**Components vs. JavaScript Functions: Key Distinctions**

While both React components and JavaScript functions involve writing code, their purposes and functionalities differ significantly:

| Aspect | React Component | JavaScript Function |
| --- | --- | --- |
| Purpose | Builds UI elements in React | Performs logic or calculations |
| Return Value | Returns JSX (React elements) | Returns any value (number, string, object, etc.) |
| Reusability | Can be reused in React UI | Reused for logic, not UI |
| State Handling | Can manage state (especially in class/function with hooks) | Does not manage UI state |
| React Features | Uses props, state, lifecycle (render, useEffect, etc.) | Purely logic-based without React context |

**Types of React Components**

React primarily offers two types of components:

* **Class Components:** These are ES6 classes that extend React.Component.
* **Function Components:** These are simpler functions that return JSX and can utilize hooks.

**Class Components**

A Class Component is a JavaScript class that must extend React.Component and define a render() method to return JSX.

**Example:**

JavaScript

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

**Features of Class Components:**

* Possess lifecycle methods like componentDidMount().
* Capable of managing state using this.state.
* Tend to be more verbose than function components.

**Understanding Function Components**

A Function Component provides a simpler way to write components using plain JavaScript functions.

**Example (Basic):**

JavaScript

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

**Example (With Hooks):**

JavaScript

import { useState } from 'react';

function Counter() {

const [count, setCount] = useState(0);

return <button onClick={() => setCount(count + 1)}>Click {count}</button>;

}

**Features of Function Components:**

* Require less boilerplate code.
* Support hooks such as useState and useEffect.
* Are the preferred choice for modern React development.

**The Component Constructor**

The constructor() method is a specialized function used within class components. Its primary purposes are to initialize state and bind methods.

**Syntax:**

JavaScript

constructor(props) {

super(props);

this.state = { count: 0 };

}

**Purpose of the Constructor:**

* To set up the initial state.
* To bind class methods to this.
* To access props before the render() method executes.

**The render() Function**

The render() function is a mandatory method in all class components. It is responsible for returning the JSX that defines the user interface for that specific component.

**Example:**

JavaScript

render() {

return <div>Hello, World!</div>;

}

**Role of the render() Function:**

* Describes what should be displayed.
* Is automatically invoked by React when a component's state or props undergo changes.

In summary, React components, encompassing both class-based and functional approaches, form the core of any React application. A thorough understanding of their distinctions, structure, and essential methods like

constructor() and render() empowers developers to construct scalable and efficient user interfaces. With the widespread adoption of hooks, functional components have emerged as the standard, offering simplicity without compromising on power. Mastering components is the foundational step towards becoming proficient in React development.