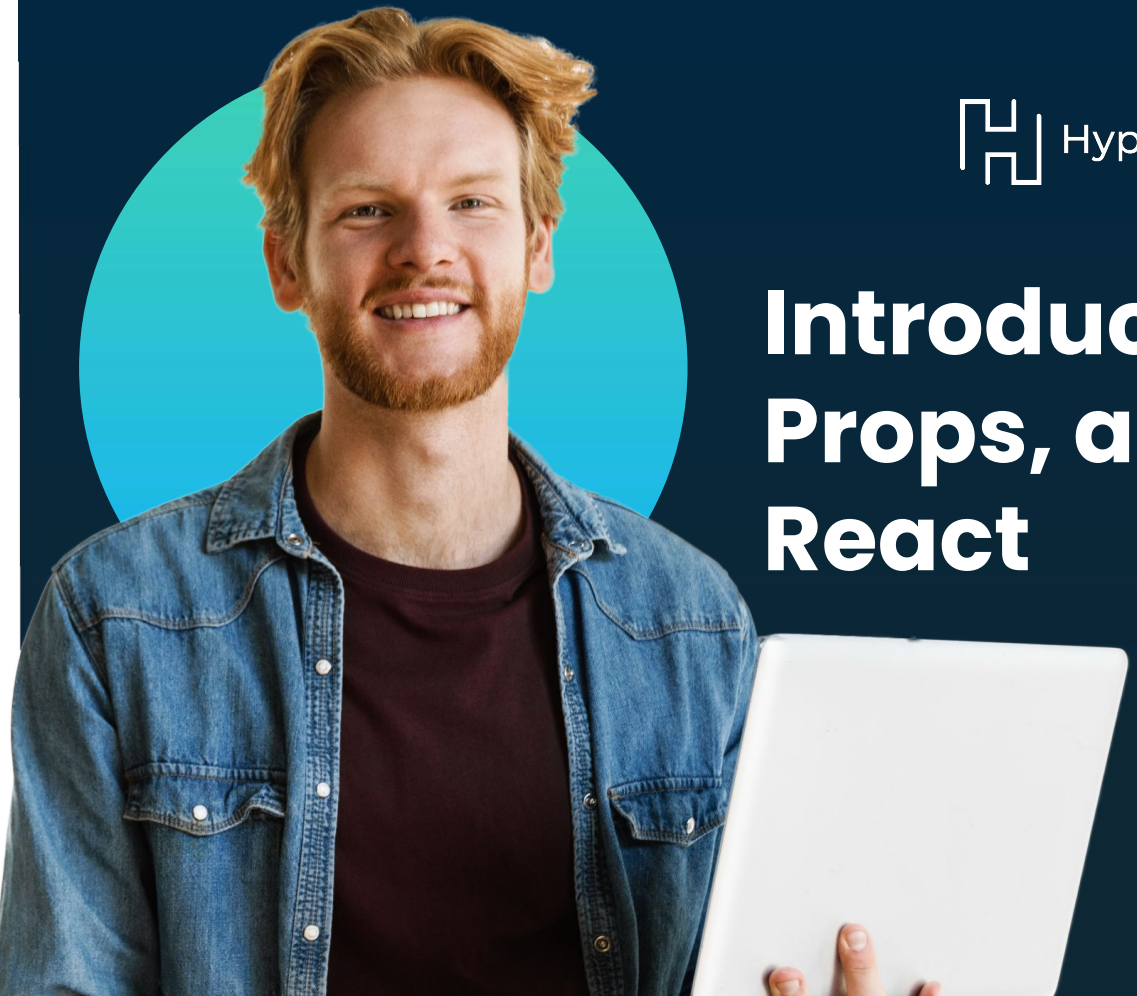
HyperionDev

# Full-Stack Web Development



HyperionDev

# Introduction to Hooks, Props, and State in React

# Full Stack Web Development Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

# **Full Stack Web Development Session Housekeeping** cont.

- For all **non-academic questions**, please submit a query: **www.hyperiondev.com/support**
- Report a **safeguarding** incident: **www.hyperiondev.com/safeguardreporting**
- We would love your **feedback** on lectures: **Feedback on Lectures**

# Objectives

- Explain the concept of Props and State in React.
- Utilize the `useState` and `useEffect` hooks to manage component state and side effects.
- Implement basic state management within a functional component.
- Understand the flow of data in React using Props.

# Introduction

- React is a powerful library for building user interfaces, primarily for single-page applications. At the heart of React are components, which are the building blocks of any React application.
- In this lesson, we focus on two essential concepts: Hooks and Props. By understanding these, you'll be able to manage state, handle side effects, and efficiently pass data between components.

# Understanding React Hooks

- React Hooks allow you to use state and other React features without writing a class component. Two fundamental hooks are useState and useEffect.
- Hooks in React are functions that let you “hook into” React state and lifecycle features from function components. Hooks don’t work inside classes, they let you use React without classes.
- Hooks are usually called at the top level of a React component.

# What is useState?

- The useState hook is a way to add local state to a functional component. Prior to hooks, you could only manage state in class components. With useState, managing state in functional components is easier and more concise.
- Hooks are usually called at the top level of a React component.
- useState returns a pair: the current state value and a function that lets you update it. You can call this function from an event handler or somewhere else. It has one argument which is the initial state.
- **Syntax:**

```
const [stateVariable, setStateVariable] = useState(initialValue);
```

  - `stateVariable` is the current state value.
  - `setStateVariable` is a function that updates the state.
  - `initialValue` is the initial state.

# Example: Counter with `useState`

```jsx
import React, { useState } from 'react';

export default function Counter() {
  const [count, setCount] = useState(0); // Initialize state with 0

  return (
    <div>
      <h1>{count}</h1>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};
```

- The useState(0) initializes the count variable with a value of 0.
- The setCount(count + 1) function updates the state by adding 1 to the current value of count.
- When you click the button, the component re-renders with the updated value.

# What is useEffect?

- The useEffect hook allows you to perform side effects in your components. Side effects include operations like data fetching, subscriptions, and directly updating the DOM. It replaces lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount in class components.
- **Syntax**:

```
import React, { useState, useEffect } from 'react';

export default function Counter() {
    const [count, setCount] = useState(0); // Initialize state with 0

    useEffect(() => {
  // Code to run when the component mounts or updates.
    return () => {
        // Cleanup code (optional) when the component unmounts or before it updates.
    };
    }, [dependencies]);
```

- **Dependencies**: The second argument is an array of dependencies that determine when the effect runs. If left empty, the effect runs only once (like componentDidMount). If you pass variables, the effect runs whenever any of them change.

# Example: Fetching Data with useEffect:

src > components > JS DataFetcher.js > ...

```jsx
import React, { useState, useEffect } from 'react';

function DataFetcher() {
  const [data, setData] = useState(null);

  useEffect(() => {
    // Fetch data from an API
    fetch('https://api.example.com/data')
      .then((response) => response.json())
      .then((data) => setData(data));
  }, []); // The empty array means this effect runs only once after the component mounts.

  return (
    <div>
      <h1>Data from API:</h1>
      {data ? <p>{JSON.stringify(data)}</p> : <p>Loading...</p>}
    </div>
  );
}

export default DataFetcher;
```

- The useEffect hook is triggered when the component mounts (because the dependencies array is empty).
- The fetch operation retrieves data and stores it in the state using setData.
- The component re-renders with the fetched data or displays a loading message while waiting.

# Understanding Props in React

- Props (short for "properties") are how you pass data from a parent component to a child component in React. Props are read-only, meaning they cannot be modified by the receiving component. This helps keep components predictable and easier to debug.

# Props vs State

- Props (short for “properties”) and state are both plain JavaScript objects. While both hold information that influences the output of component render, they are different in one important way: props get passed to the component (similar to function parameters) whereas state is managed within the component (similar to variables declared within a function).
- Essentially React component props are used to pass data from component to component. State is used to manage data within a single component.

# How to Pass Props

## Example:

src > components > JS DataFetcher.js > ...

```jsx
import React from 'react';

function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

function App() {
  return <Greeting name="Alice" />;
}

export default App;
```

- The App component passes the name prop with a value of "Alice" to the Greeting component.
- The Greeting component receives the prop via the props object and uses it in the JSX to display the name.

# Prop Destructuring for Clean Code

- Instead of accessing props via props.name, you can destructure them directly in the function signature:

src > components > JS DataFetcher.js > ...

```
1   import React from 'react';
2
3   function Greeting({ name }) {
4     return <h1>Hello, {name}!</h1>;
5   }
6
7   function App() {
8     return <Greeting name="Alice" />;
9   }
10
11  export default App;
12
```

# Practical Activity: Building a Simple Counter App with React

- In this activity, we'll create a counter app that allows users to increment, decrement, and reset the counter value. This exercise will reinforce your understanding of state, props, and handling user interactions in React.

Hyperiondev

# Questions and Answers

HyperionDev

# Thank You for attending!