CoGrammar

Welcome to this session:
Task Walkthrough - React - Hooks

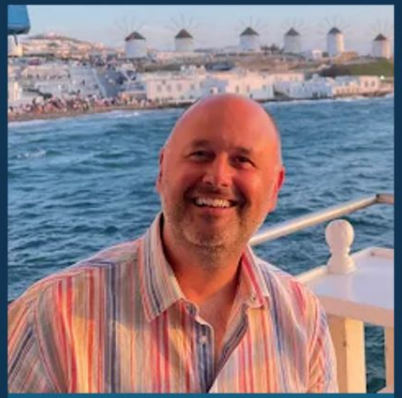
The session will start shortly...

Questions? Drop them in the chat.
We'll have dedicated moderators answering questions.

Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:



Ian Wyles
Designated Safeguarding Lead

Simone Botes

Nurhaan Snyman

Rafiq Manan

Ronald Munodawafa

Tevin Pitts

Scan to report a safeguarding concern

or email the Designated Safeguarding Lead:
Ian Wyles
safeguarding@hyperiondev.com

CoGrammar HyperionDev

Skills Bootcamp Cloud Web Development

- 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**

Skills Bootcamp Cloud Web Development

- 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***
- If you are hearing impaired, please kindly use your computer's function through Google chrome to enable captions.

Learning Outcomes

- **Set up a React project using Vite** and configure it for API integration.
- **Use useState and useEffect hooks** to manage data and side effects in React applications.
- **Implement useRef** for managing focus and DOM interactions.
- **Fetch and display data from external APIs** dynamically based on user input.
- **Design user-friendly interfaces** to present fetched data in an intuitive way.

Lecture Overview

- Presentation of the Task
- Introduction to Hooks
- State Hook
- Effect Hook
- Cleanup Function
- Ref Hook
- Task Walkthrough

Hooks Task

Imagine creating an app that's like a digital fortune cookie—except it's powered by numbers! In this task, you'll build a Fun Trivia Fetcher App that takes your favorite number and delivers a fascinating fact about it. 🎲

Want to know what makes the number 42 so special? Or why the number 7 might be lucky? With just a few clicks, your app will pull exciting trivia straight from the Numbers API and display it instantly.

- Feature a sleek, auto-focused input field where users can type in their number.
- Fetch a quirky, fun fact about the number when the user clicks a button.
- Display the trivia in a visually appealing way, with smooth and responsive interactions.

Which HTTP method is commonly used to fetch data from an API in React?

A. POST
B. DELETE
C. PUT
D. GET

CoGrammar

How can you avoid unnecessary API calls in a React component?

A. Use useState without rendering
B. Use useEffect with proper dependency arrays
C. Use useRef to skip rendering
D. Use inline functions in JSX

React Hooks

JavaScript functions that allow functional components to access React features, like state and side effects.

- Before Hooks, **class components** were used, which allowed internal state to be managed and lifecycle events to be handled directly.
- React Hooks allow us to work with React components in a **simpler and more concise** way, without having to write classes.
- Hooks also make our code more **readable** and **maintainable**.
- There are **many types of hooks**, and **custom hooks** can be defined as well.
- This lecture will be covering state, effect and ref hooks.

State Hook

Hook used for state management, allowing components to store and retrieve information.

- The **useState** hook declares a **state variable**, which is **preserved between function calls** and whose **change triggers a rerender**.
- The function **accepts** the **initial state** of the variable as input.
- The function **returns** a pair of values: the **state variable** and the **function that updates it**.

```
const [number, setNumber] = useState(10);
const [string, setString] = useState("");
const [object, setObject] = useState({
    attribute1: "Name",
    attribute2: 23,
    attribute3: false });
```

CoGrammar

Function Components Recap: JavaScript functions which accept a single prop object as input and use hooks to create reusable pieces of UI by returning React elements.

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

function Counter () {
    let [count, setCount] = useState(0);

    function inc () {
        setCount(count + 1);
    }

    return (
        <div>
            <p>Count: {count}</p>
            <button onClick={inc} >Increment</button>
        </div>
    )
}

export default Counter;
```

CoGrammar

This is how we would implement the counter with a class component.

```jsx
import React, { Component } from "react";

class Counter extends Component {
    constructor() {
        super();
        this.state = {
            count: 0
        };
        this.inc = this.inc.bind(this);
    }

    inc () {
        this.setState({ count: this.state.count + 1 });
    }

    render() {
        return (
            <div>
                <p>Count: {this.state.count}</p>
                <button onClick={this.inc} >Increment</button>
            </div>
        )
    }
}
export default Counter;
```

Effect Hook

Hook used for connecting to and synchronizing external systems after your components are rendered, known as performing side effects.

- The **useEffect** hook is used for tasks like **fetching data**, directly **updating the DOM** and setting up **event listeners**.
- The function takes in two arguments: a **block of code** which will be executed when the component is loaded, and a **dependencies list**, which is a list of variables whose change will trigger the first argument to be rerun.
- If **no dependency argument** is passed, the first argument will run on **every render**.
- If an **empty dependency argument** is passed, the first argument will on be run on the first render of the component.

Fetch Data from API

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

function API() {
  let [funFact, setFunFact] = useState(null);

  useEffect(() => {
    async function fetchData() {
      let response = await fetch("https://catfact.ninja/fact/");
      let data = await response.json();
      console.log(data.fact)
      setFunFact(data.fact);
    }
    fetchData();
  },[])

  return (
    <h1>{funFact}</h1>
  )
}
export default API;
```

Cleanup Function

Function returned by the useEffect hook which gets executed before every rerun of the component and after the component is removed.

- Tasks that can be performed in the useEffect hook, may need to be **aborted or stopped** when the **component is removed** or when **state changes**.
- For example, API calls may need to be aborted, timers stopped and connections removed.
- If this is not handled properly, your code may attempt to update a state variable which no longer exists, resulting in a **memory leak**.
- This is done with a **cleanup function**, which is **returned by the useEffect hook**. This function will run when the component is **removed** or **rerendered**.

Cleanup Function

```
import { useEffect } from 'react';

function SweepAway () {
  useEffect(() => {
    const clicked = () => console.log('window clicked')
    window.addEventListener('click', clicked)

    // return a clean-up function
    return () => {
      window.removeEventListener('click', clicked)
    }
  }, [])

  return (
    <div>When you click the window you'll find a
         message logged to the console</div>
  )
}
```

CoGrammar

Ref Hook

Hook used to store mutable values which do not trigger re-renders and update DOM elements directly.

- The **useRef** hook is store values which **persist between re-renders**, but **do not cause the component to re-render** when changed.
- We can also access DOM elements using useRef by passing the returned object to elements in the **ref** attribute.
- The function accepts an **initial value** as an **input**.
- The function returns an **object** with the property **current** initialised to the value passed as input to the function.

Ref Hook

```
import { useRef } from 'react';

function PetCat () {

    let pet = useRef(0);

    function handleClick() {
        pet.current = pet.current + 1;
        alert('You clicked ' + pet.current + ' times!');
      }

  return (
    <div>
        <button onClick={handleClick}> Pet the virtual cat! </button>
    </div>
  )
}

export default PetCat;
```

CoGrammar

Hooks Task

Imagine creating an app that's like a digital fortune cookie—except it's powered by numbers! In this task, you'll build a Fun Trivia Fetcher App that takes your favorite number and delivers a fascinating fact about it. 🎲

Want to know what makes the number 42 so special? Or why the number 7 might be lucky? With just a few clicks, your app will pull exciting trivia straight from the Numbers API and display it instantly.

- Feature a sleek, auto-focused input field where users can type in their number.
- Fetch a quirky, fun fact about the number when the user clicks a button.
- Display the trivia in a visually appealing way, with smooth and responsive interactions.

What is the primary purpose of useEffect in React?

A. To handle component styling
B. To fetch data or perform side effects after rendering
C. To manage component-level state
D. To update DOM elements directly

What is useRef commonly used for in React?

A. Styling components dynamically
B. Referencing DOM elements directly
C. Managing component state
D. Storing API responses

CoGrammar

CoGrammar

Q & A SECTION

Please use this time to ask any questions relating to the topic, should you have any.

Thank you for attending

CoGrammar

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education