



## Welcome to this session: Components and Hooks in React

**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](mailto:safeguarding@hyperiondev.com)

# Skills Bootcamp Full-Stack Software 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 Full-Stack Software development

---

- For all **non-academic questions**, please submit a query:  
**[www.hyperiondev.com/support](http://www.hyperiondev.com/support)**
- **Report a safeguarding incident:** **[www.hyperiondev.com/safeguardreporting](http://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

---

By the end of this lesson, learners should be able to:

- Implement state management using the **useState** hook.
- Utilize the **useEffect** hook to manage side effects in functional components.
- Apply the **useRef** hook to access DOM elements and store mutable values.
- Create **functional components** that leverage **multiple hooks** for various functionalities.



## Which of the following correctly describes a React hook?

- A. A function that lets you "hook into" React features
- B. A lifecycle method for class components
- C. A third-party library for managing state
- D. A debugging tool for React



## What is the correct rule about calling hooks in React?

- A. They must be called only inside custom hooks
- B. They must be called in the top-level scope of a functional component
- C. They can be called inside any function
- D. They must only be called in class components



**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.

```
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;
```



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

```
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;
```

# 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.

# useState 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 });
```

# useEffect 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 only 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();
  }, [])

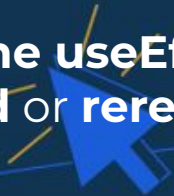
  return (
    <h1>{funFact}</h1>
  )
}

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>
  )
}
```



# 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;
```



## Which of these is true about useState in React?

- A. It can only be used to store primitive data types
- B. It allows you to define and update local state in functional components
- C. It is a lifecycle method replacement for componentDidMount
- D. It is used for optimizing performance



## What will happen if you omit the dependency array in `useEffect`?

- A. The effect will run only once
- B. The effect will not run at all
- C. The effect will run after every render
- D. The effect will run only when unmounting

# Questions and Answers



# Thank you for attending



**CoGrammar**



Department  
for Education