



Welcome to this session: Local State Management and Events

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

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

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

- **Learn different methods to handle events** in React components, including inline handlers, method binding, and arrow functions.
- **Explore common React event types** like `onClick`, `onChange`, and `onSubmit`, and recognize how to use them effectively.
- **Define the concept of state management** in React and identify how the state is managed within components.
- **Implement the `useState()` hook** to declare and manage local state in React functional components.



Which npm command is used to create a React App with Vite?

- A. `npm create react-app my-app --template vite`
- B. `npm create vite@latest my-app -- --template react`
- C. `npm install vite --template react`
- D. `npm init vite my-app --template react`






What is the purpose of props within a React component?

- A. To store data that can change over time.
- B. To pass data and functions from a parent component to a child component.
- C. To manage the local state of a component.
- D. To trigger re-renders of a component when the data changes.



What do you think is the purpose of a state variable within a React component?

- A. To store and track data that can change over time.
 - B. To handle component styling and animations.
 - C. To manage global state across multiple components.
 - D. To optimise the performance of React components by reducing re-renders.
 - E. Other (please specify in the Question section)
- 

Lecture Overview

- Introduction to events in React
- Explore common React event types
- Introduction to state management
- Implement the `useState()` hook

What are events in React?

- ❖ Events in React are occurrences or actions, such as user inputs (clicking, typing) or system-driven actions, that React components can respond to.
- ❖ **Synthetic Events:** React provides a "synthetic event" system, which standardizes browser events. This ensures consistent behavior across various browsers.

What are events handlers in React?

- ❖ React lets you add event handlers to your JSX. Event handlers are your own functions that will be triggered in response to interactions like clicking, hovering, focusing form inputs, and so on.
- ❖ In React.js, event handling is crucial for building interactive and dynamic user interfaces.
- ❖ Events like `onClick`, `onChange`, `onSubmit`, etc., enable users to interact with components, triggering updates and actions.
- ❖ Understanding event handling enhances the responsiveness and interactivity of React applications.

Adding event handlers

- ❖ To add an event handler, you will first define a function and then pass it as a prop to the appropriate JSX tag.

```
export default function Button() {  
  function handleClick() {  
    alert('You clicked me!');  
  }  
  
  return (  
    <button onClick={handleClick}>  
      Click me  
    </button>  
  );  
}
```

Adding event handlers

- ❖ Alternatively, you can define an event handler inline in the JSX:

```
<button onClick={function handleClick() {  
  alert('You clicked me!');  
}}>
```

- ❖ Or, more concisely, using an arrow function:

```
<button onClick={() => {  
  alert('You clicked me!');  
}}>
```

Passing event handlers as props

- ❖ Often you'll want the parent component to specify a child's event handler.
- ❖ To do this, pass a prop the component receives from its parent as the event handler.

```
import React from 'react';  
function SuperButton({ onClick }) {  
  return <button onClick={onClick}>I am a Super Button!</button>;  
}
```

Event Object

- ❖ When an event occurs, React automatically passes an event object to the event handler function. This object contains a wealth of information about the event, including the following common properties:
 - **target:** The DOM element that triggered the event.
 - **type:** The type of the event (e.g., "click", "change").
 - **preventDefault():** A method to prevent the default behavior of the event.

Event Object

```
import React from 'react';

const MyComponent = () => {
  const handleClick = (event) => {
    console.log('Button clicked!');
    console.log('Event:', event);
    console.log('Target Element:', event.target);
    console.log('Event Type:', event.type);
    console.log('Current Target:', event.currentTarget);
  };

  return <button onClick={handleClick}>Click Me</button>;
};
```

Remember

- ❖ Functions passed to event handlers must be passed, not called.
- ❖ The () at the end of handleClick() fires the function immediately during rendering, without any clicks. This is because JavaScript inside the JSX {} executes right away.

passing a function (correct)	calling a function (incorrect)
<code><button onClick={handleClick}></code>	<code><button onClick={handleClick()}></code>

Passing Arguments to Event Handlers

- ❖ You can pass additional arguments to event handlers using arrow functions.

```
export default function App() {  
  const handleClick = (arg) => {  
    console.log('Clicked with argument:', arg);  
  };  
  
  return <button onClick={() => handleClick('Hello')}>Click Me</button>;  
}
```

State Management

- ❖ **State management** is the process of **handling and updating data** within a React application.
- ❖ It allows components to **maintain their internal state** and **respond to user interactions effectively**.
- ❖ In React, **state** refers to an **object that represents the current condition of a component**.

State Management

- ❖ **Stateful** components have the **ability to hold and modify their state**, which **affects their rendering and behavior**.
- ❖ When a component's state changes, React automatically **re-renders the component** to reflect the updated state.
- ❖ **Changes to state trigger a re-render of the component and its child components**, ensuring that the UI stays in sync with the underlying data.

useState() Hook

- ❖ Before React introduced **hooks**, only **class components** could have state, but with `useState`, **functional components** can now manage and update state without needing to be converted to class components.
- ❖ **React hooks** are powerful functions that can be used in a functional component to manipulate the state of the component.
- ❖ The `useState` hook allows us to **declare state variables** and **update them within the component**.

useState() syntax

- ❖ **Declaration:** You call useState inside a functional component to declare a state variable and a function to update that state.
- ❖ **Syntax:**

```
const [state, setState] = useState(initialState);
```

 - **state** is the variable that holds the current state value.
 - **setState** is the function used to update that state.
 - **initialState** is the initial value of the state.

Why do we declare useState() with const

- ❖ In React, we declare useState as a const because useState returns an array (or a tuple) containing two values:
 - The state variable, which holds the current state value.
 - The state updater function, which is used to update the state.
- ❖ Since the returned values don't change, we use const to ensure that these variables remain immutable (not reassigned) throughout the component lifecycle.

Why do we declare useState() with const

- ❖ Here's why we use const:
 - **Preventing Reassignments:** useState returns a fixed pair of values (the state and the updater function). Declaring them as const ensures that we don't accidentally reassign the variables to something else within the component.
 - **Clarifying Intent:** Using const communicates the intent that the reference to the state and updater function should not change during the component's render cycle. This makes the code more predictable and easier to understand.
 - **Consistency:** By using const, you avoid bugs that could arise if you mistakenly reassign the state variable or the updater function, ensuring that the values remain constant.

Why do we declare useState() with const

❖ Example:

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

- **count** will hold the current state value.
- **setCount** is the function that updates the state.
- ❖ Here, both **count** and **setCount** are declared with **const** because we don't want to change the **reference** to these variables after they are assigned. **The value of count can change**, but the **reference to count itself remains constant**, which aligns with React's rules of state management.

Sharing State

- ❖ Often we want to **share state between multiple components** so when the state changes, **all dependent components re-render**.
- ❖ To do this, we **create a new parent component** to handle the state: creating state variables and functions which are called to change state.
- ❖ The state variables can then be passed to the child components as **props**.
- ❖ This process is often referred to as **“lifting state up”**.



What is the purpose of an event handler within a React component?

- A. To manage the component's internal state.
- B. To handle and respond to user interactions like clicks, inputs, etc.
- C. To manage the data flow between parent and child components.
- D. To optimise the performance of the component by reducing renders.



What is the purpose of the `useState()` hook within React?

- A. To trigger re-renders when a component's props change.
- B. To pass state from a parent component to a child component.
- C. To handle side effects like fetching data or subscribing to external events.
- D. To store and manage the local state within a functional component.

Questions and Answers



Thank you for attending



CoGrammar



Department
for Education