



Welcome to this session: State and Events 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

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
- **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:

- **Define the core concepts of state and events in React.**
- **Explain how React's state and event system works within a component.**
- **Demonstrate how to implement state and event handling in a React component.**
- **Identify issues related to state management and event handling in React applications.**



What do you think “state” might refer to in a programming context?

- A. The styling of elements on a webpage
- B. Data or information that changes over time (e.g user input)
- C. The layout of a webpage
- D. A location where code is executed





Which of the following best describes JavaScript event?

- A. A user interaction, like a mouse click or a keyboard input, that triggers some action on a webpage
- B. A request to the server to fetch data
- C. A delay before the page loads
- D. A specific type of error that occurs in JavaScript



Lecture Overview

- Introduction to State
- Introduction to hooks (useState hook)
- Prop drilling
- Event handling



What is State in React?

- ❖ In React, state refers to an object that represents the current condition of a component.
- ❖ Stateful components have the ability to hold and modify their state, which affects their rendering and behavior.



How Does State Work?

- ❖ 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

- ❖ In functional components, we use the useState hook to introduce stateful behavior.
- ❖ The useState hook allows us to declare state variables and update them within the component.



useState Hook

- ❖ state: Represents the current value of the state variable.
- ❖ setState: A function used to update the state variable and trigger re-rendering.

```
let [fullName, setFullName] = useState('Clark Kent');
```

Prop drilling

- ❖ Prop drilling is a common challenge in React applications where data needs to be passed through multiple layers of components.
- ❖ It arises when passing props down several levels in the component tree, leading to code complexity and maintenance issues.

Prop drilling: Examples

```
import React from 'react';
import ParentComponent from './Parent';

function GrandParentComponent() {
  const userData = { name: 'John', age: 30 };

  return <ParentComponent userData={userData} />;
}

export default GrandParentComponent;
```

Prop drilling: Examples

```
import React from 'react';
import ChildComponent from './Child';

function ParentComponent({ userData }) {
  |   return <ChildComponent userData={userData} />;
  }

export default ParentComponent;
```


Prop drilling: Examples

```
import React from 'react';
import User from './User';

function ChildComponent({ userData }) {
  |   return <User userData={userData} />;
  }

export default ChildComponent;
|
```

Prop drilling: Examples

```
import React from 'react';

function User({ userData }) {
  return (
    <div>
      <h2>User Details</h2>
      <p>Name: {userData.name}</p>
      <p>Age: {userData.age}</p>
    </div>
  );
}

export default User;
```



Challenges of Prop Drilling


- ❖ Prop drilling can make code harder to maintain and refactor, especially in large component hierarchies.
- ❖ It increases coupling between components and makes it difficult to track data flow.

Let's take a
break





Events 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 (onClick event)

- ❖ 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>  
  );  
}
```


Remember

- ❖ Functions assigned 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 `{}` and `}` executes right away.

passing a function (correct)

```
<button onClick={handleClick}>
```

calling a function (incorrect)

```
<button onClick={handleClick()}>
```


onChange Event

- ❖ The onChange event is triggered when the value of an input element changes. It's commonly used for handling user input in forms.

```
const MyComponent = () => {  
  const [value, setValue] = useState('');  
  
  const handleChange = (event) => {  
    console.log('Input changed!');  
    console.log('New value:', event.target.value);  
    setValue(event.target.value);  
  };  
  
  return <input type="text" value={value} onChange={handleChange} />;  
};
```

onSubmit Event

- ❖ The onSubmit event is triggered when a form is submitted. It's essential for form validation and submission.

```
const MyComponent = () => {  
  const handleSubmit = (event) => {  
    console.log('Form submitted!');  
  };  
  
  return (  
    <form onSubmit={handleSubmit}>  
      <button type="submit">Submit</button>  
    </form>  
  );  
};
```

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>;  
}
```

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>;
};
```

preventDefault

- ❖ The `preventDefault` method is called within event handlers to prevent the default behavior of an event. It's commonly used to prevent form submission or link navigation.

```
export default function App() {  
  const handleClick = (e) => {  
    e.preventDefault();  
    console.log('Hayyyyyyyy!!!');  
  };  
  
  return (  
    <div>  
      <a href="/items" onClick={handleClick}>  
        Click me  
      </a>  
    </div>  
  );  
}
```


Practical Example: A simple Counter App

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

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

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
    </div>
  );
}

export default Counter;
```




What is the correct way to initialize state in a functional component using useState?

- A. `const [state, setState] = useState(initialValue)`
- B. `const [state] = useState(initialValue)`
- C. `const state = useState(initialValue)`
- D. `const setState = useState(initialValue)`





Which of the following is a valid React event handler for a form input change?

- A. `onChange={handleClick}`
- B. `onClick={handleChange}`
- C. `onSubmit={handleInputChange}`
- D. `onChange={handleInputChange}`

Questions and Answers



Thank you for attending



CoGrammar



Department
for Education