WebDev

Lesson 4

Events

Event handles react on Events

```jsx
function ChatRoom({ roomId }) {
  const [message, setMessage] = useState('');
  // ...
  function handleSendClick() {
    sendMessage(message);
  }
  // ...
  return (
    <>
      <input value={message} onChange={e => setMessage(e.target.value)} />
      <button onClick={handleSendClick}>Send</button>;
    </>
  );
}
```

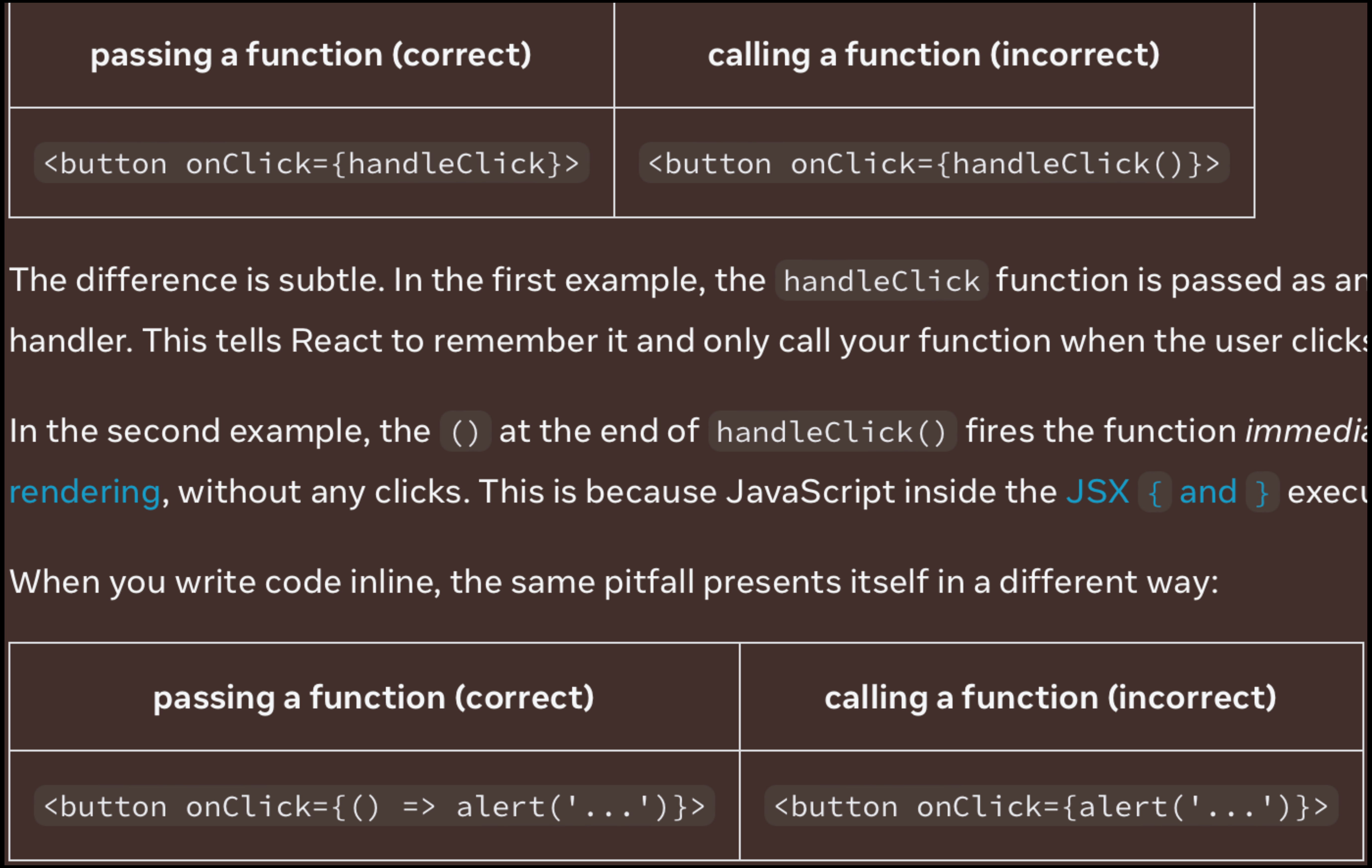
Events

Incorrect usage

Do not call callbacks of events during attribute assignment.

event={handler} - correct

event={() => {}} - correct



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

The difference is subtle. In the first example, the `handleClick` function is passed as an handler. This tells React to remember it and only call your function when the user click

In the second example, the `()` at the end of `handleClick()` fires the function *immedia* rendering, without any clicks. This is because JavaScript inside the JSX `{` and `}` execu

When you write code inline, the same pitfall presents itself in a different way:

passing a function (correct)	calling a function (incorrect)
`<button onClick={() => alert('...')}>`	`<button onClick={alert('...')}>`

Event propagation

Event handlers catch Events of children.

Both buttons trigger <div> onClick handler.

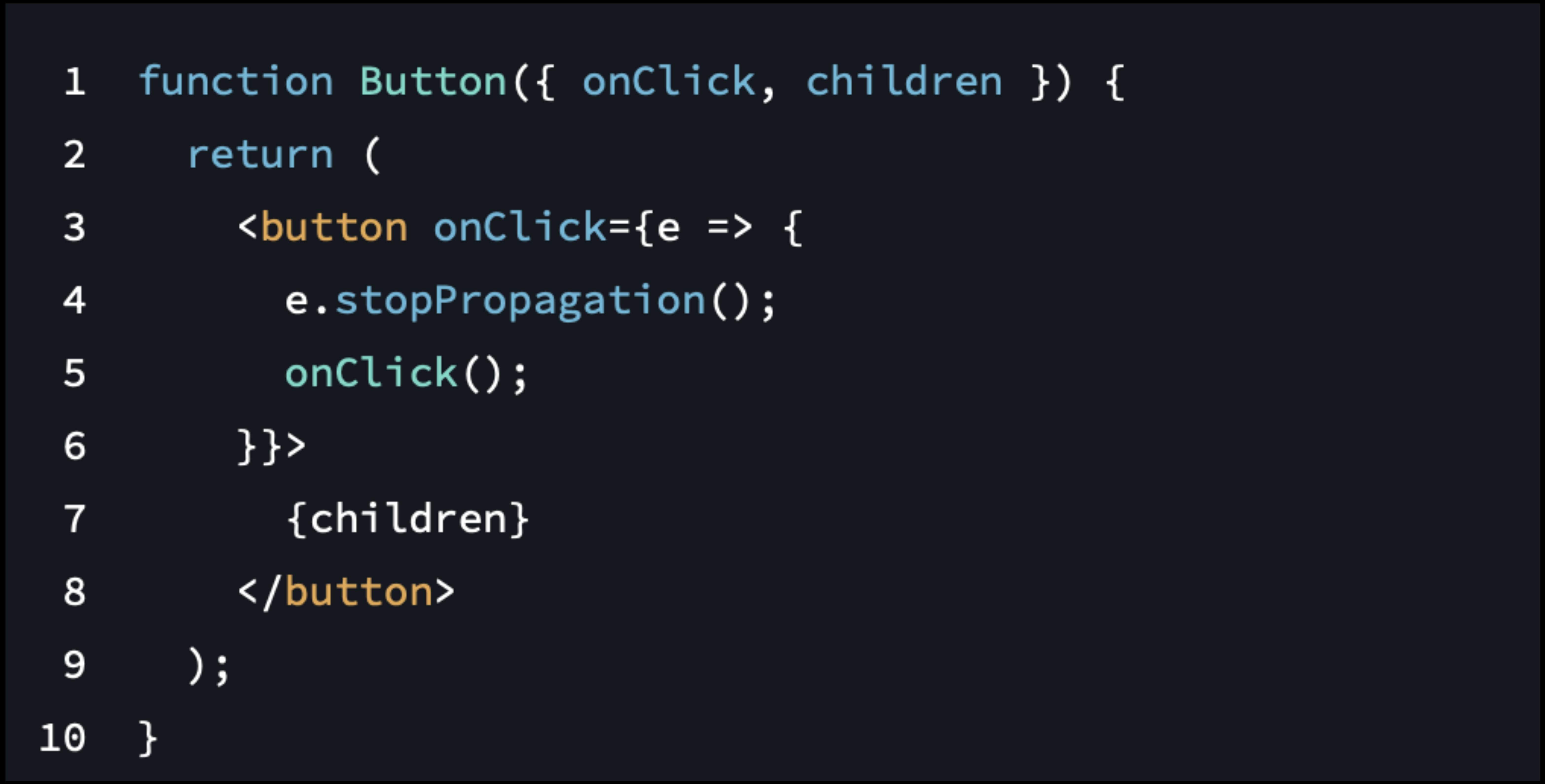
```
export default function Toolbar() {
  return (
    <div className="Toolbar" onClick={() => {
      alert('You clicked on the toolbar!');
    }}>
      <button onClick={() => alert('Playing!')}>
        Play Movie
      </button>
      <button onClick={() => alert('Uploading!')}>
        Upload Image
      </button>
    </div>
  );
}

```

Event propagation

e = Event object

stopPropagation() - prevents event propagation into parents.



```jsx
function Button({ onClick, children }) {
  return (
    <button onClick={e => {
      e.stopPropagation();
      onClick();
    }}>
      {children}
    </button>
  );
}
```

Event propagation

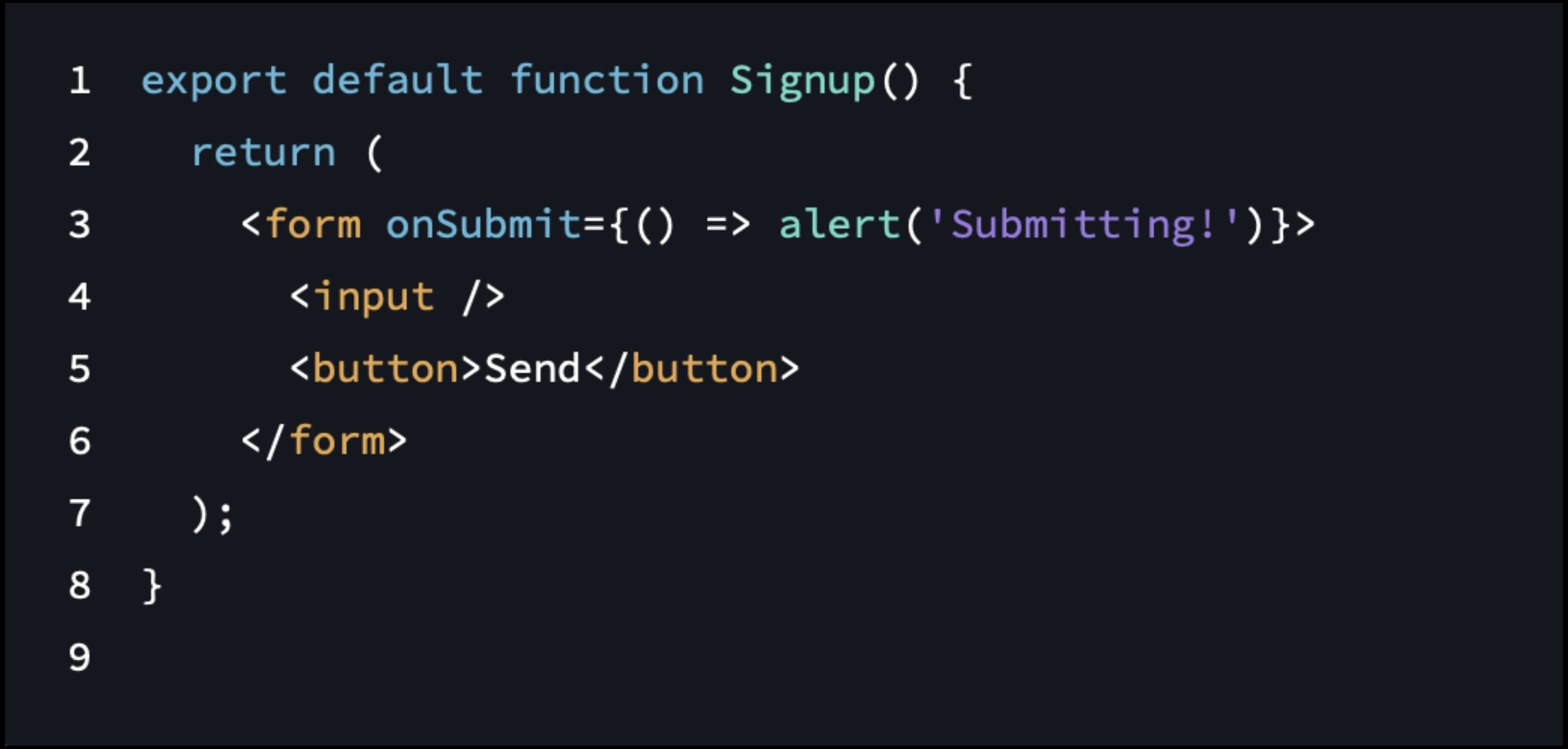
Not <Button> onClick handlers won’t cause <div> onClick handler activation.

```
export default function Toolbar() {
  return (
    <div className="Toolbar" onClick={() => {
      alert('You clicked on the toolbar!');
    }}>
      <Button onClick={() => alert('Playing!')}>
        Play Movie
      </Button>
      <Button onClick={() => alert('Uploading!')}>
        Upload Image
      </Button>
    </div>
  );
}
```

Default behavior

Some Events have default behavior during event trigger.

Submit event of Form tag reloads the page.



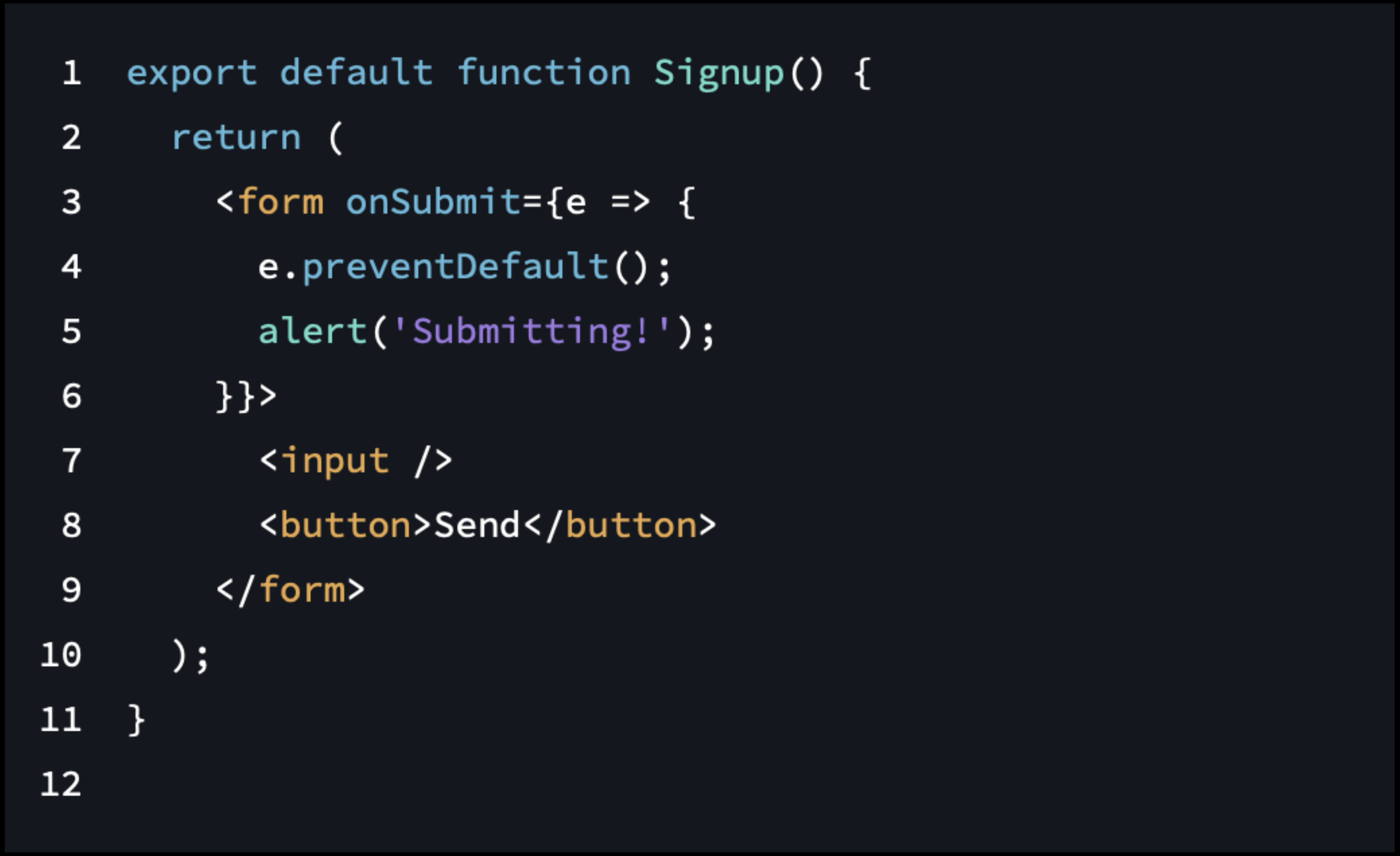
```
export default function Signup() {
  return (
    <form onSubmit={() => alert('Submitting!')}>
      <input />
      <button>Send</button>
    </form>
  );
}
```

Default behavior

e - Event

preventDefault - disables default behavior of given Event.

Thus, no reload for this Submit event.



```
export default function Signup() {
  return (
    <form onSubmit={e => {
      e.preventDefault();
      alert('Submitting!');
    }}>
      <input />
      <button>Send</button>
    </form>
  );
}
```

Memoization

useMemo, useCallback

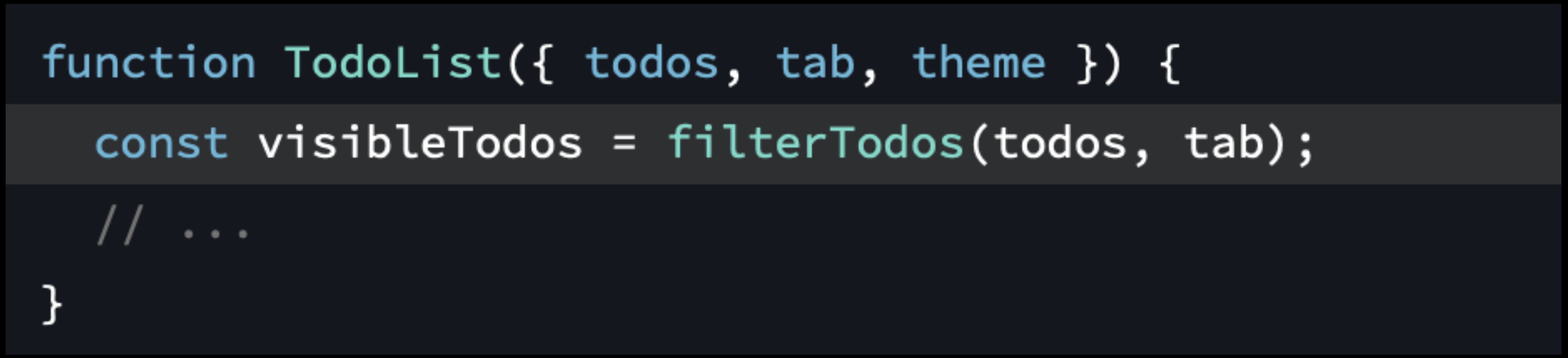
Computationally heave code left unmanaged may hinder performance.

Body of functions will re-run each render

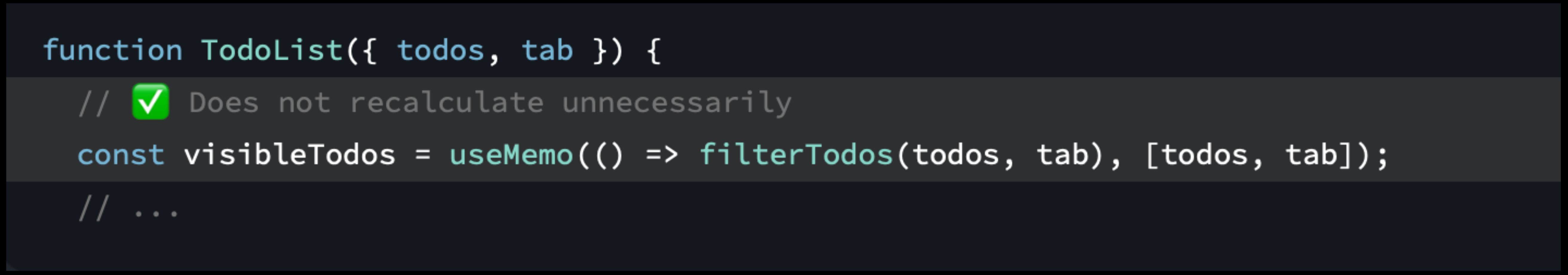
Memoization

useMemo

useMemo is a great hook for creating cache of our re-calculating variables



```
function TodoList({ todos, tab, theme }) {
  const visibleTodos = filterTodos(todos, tab);
  // ...
}
```



```
function TodoList({ todos, tab }) {
  // ✅ Does not recalculate unnecessarily
  const visibleTodos = useMemo(() => filterTodos(todos, tab), [todos, tab]);
  // ...
```

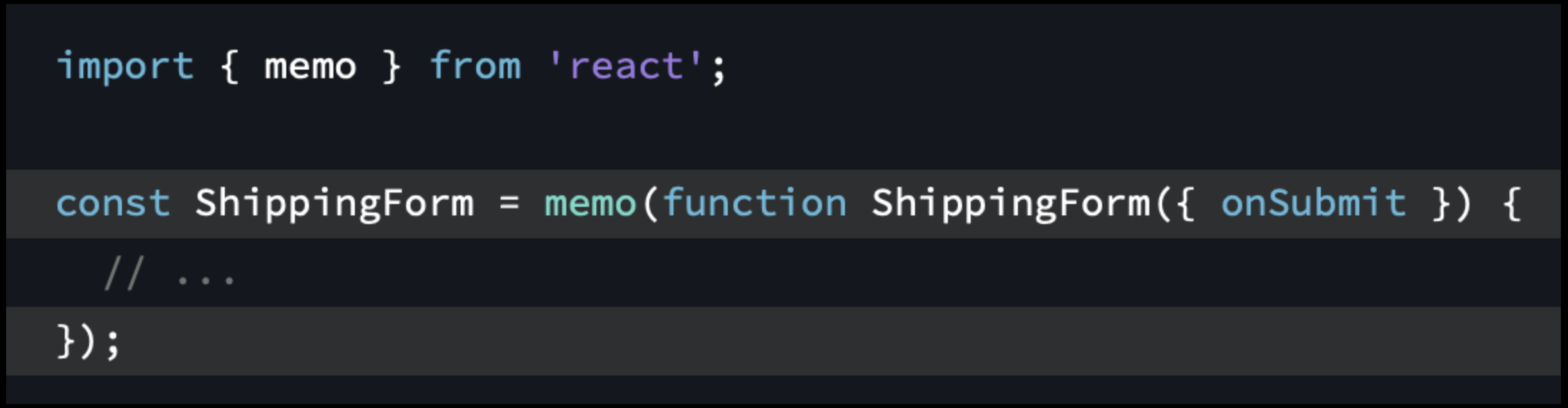
Memoization

useCallback

useMemo is a great hook for
creating cache of our re-calculating
functions

memo() - components

useMemo is a great hook for creating cache of our re-calculating functions



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

const ShippingForm = memo(function ShippingForm({ onSubmit }) {
  // ...
});
```