Here is a simple and practical example what it means to "lift state up" in React, and how it can help you build your applications.

Lifting state up is a common pattern that is essential for React developers to know. It helps you avoid more complex (and often unnecessary) patterns for managing your state.

How does it do that? Let's see how through a simple example.

Breaking down our Todo App

Let's start with a basic todo application, which consists of three components: TodoCount, TodoList, and AddTodo.

All of these components, as their name suggests, are going to need to share some common state.

If you look at TodoCount, this is where you're going to display, up at the top of your app, how many total to dues you have in your application.

TodoList is going to be where you show all of your todos. It has some initial state with these three items ("item 1", "item 2", "item 3") which you'll display within an unordered list.

And finally, you have AddTodo. This consists of a form, where you want to be able to add a new item to this list. Right now you're just logging the todo that you type into the input to the console:

```
// src/App.js

import React from "react";

export default function App() {
  return (
    <>
      <TodoCount />
      <TodoList />
      <AddTodo />
    </>
  );
}
```

```
function TodoCount() {
  return <div>Total Todos: </div>;
}

function TodoList() {
  const [todos, setTodos] = React.useState(["item 1", "item 2", "item 3"]);

  return (
    <ul>
      {todos.map((todo) => (
        <li key={todo}>{todo}</li>
      ))}
    </ul>
  );
}

function AddTodo() {
  function handleSubmit(event) {
    event.preventDefault();
    const todo = event.target.elements.todo.value;
    console.log(todo);
  }

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" id="todo" />
      <button type="submit">Add Todo</button>
    </form>
  );
}
```

Why should you care about lifting state up?

How can you use the concept of lifting state up to help finish your application?

These components need to share some state, some todo state. You need to share that todo state order to display the number of todos as well as to update your todo list. This is where the concept of lifting state up comes in.

We lift up state to a common ancestor of components that need it, so that they can all share in the state. This allows us to more easily share state among all of these components that need rely upon it.

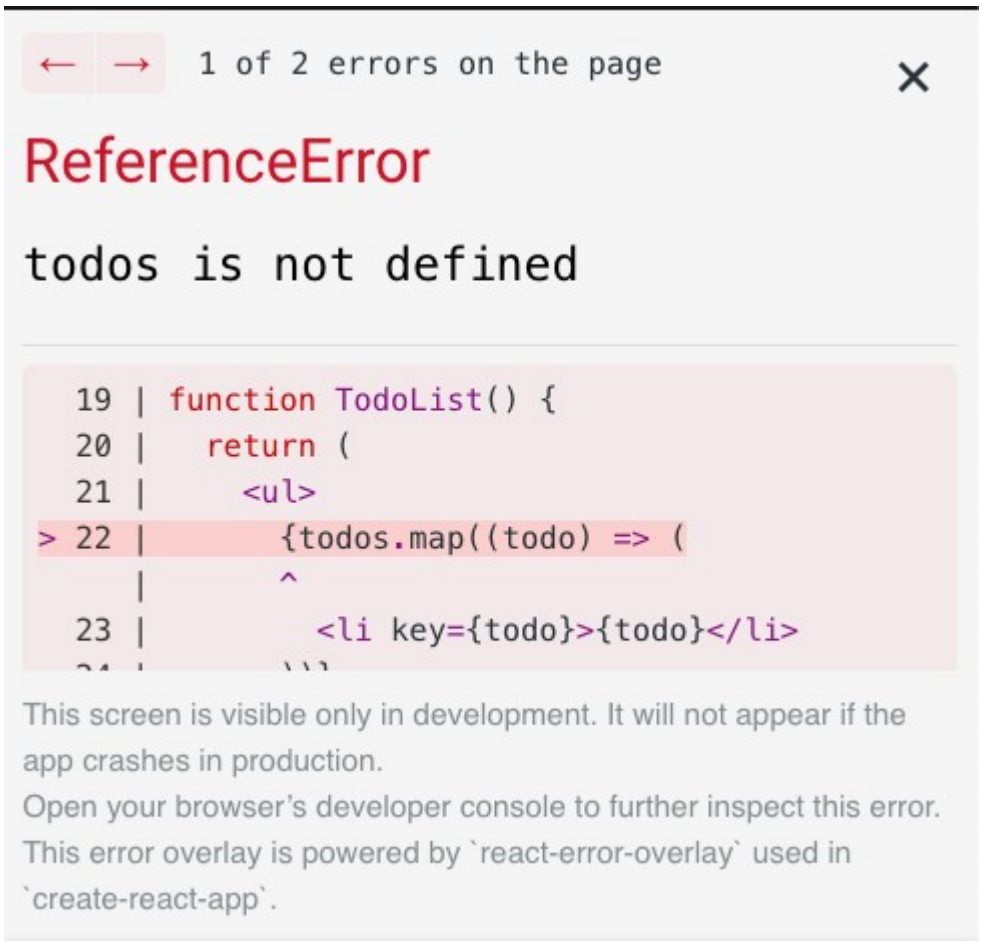
What common ancestor should you lift up your state to so all of the components can read from and update that state? The App component.

Here's what your app should now look like:

```
// src/App.js

import React from "react";

export default function App() {
  const [todos, setTodos] = React.useState(["item 1", "item 2", "item 3"]);

  return (
    <>
      <TodoCount />
      <TodoList />
      <AddTodo />
    </>
  );
}

function TodoCount() {
  return <div>Total Todos: </div>;
}

function TodoList() {
  return (
    <ul>
      {todos.map((todo) => (
        <li key={todo}>{todo}</li>
      ))}
    </ul>
  );
}

function AddTodo() {
  function handleSubmit(event) {
    event.preventDefault();
    const todo = event.target.elements.todo.value;
    console.log(todo);
  }

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" id="todo" />
      <button type="submit">Add Todo</button>
    </form>
  );
}
```

Yow to Pass State Down

However, there's a small problem.
TodoList doesn't have access to the todos state variable, so you need to pass that down from App:



You can do that with components in React using props. On TodoList, let's add a prop named todos. You can destructure todos from the props object. This allows you to see your todo items once again.

Now what about displaying the total number of todos within the TodoCount component?

This is another instance in which you can pass down the data as a prop, since TodoCount relies upon that state. Once again, we'll provide the same prop, todos, destructure it from the props object and display the total number of todos using todos.length:

```
import React from "react";

export default function App() {
  const [todos, setTodos] = React.useState(["item 1", "item 2", "item 3"]);

  return (
    <>
      <TodoCount todos={todos} />
      <TodoList todos={todos} />
      <AddTodo />
```

```
    </>
  );
}

function TodoCount({ todos }) {
  return <div>Total Todos: {todos.length}</div>;
}

function TodoList({ todos }) {
  return (
    <ul>
      {todos.map((todo) => (
        <li key={todo}>{todo}</li>
      ))}
    </ul>
  );
}
```

How to Pass Callbacks Down

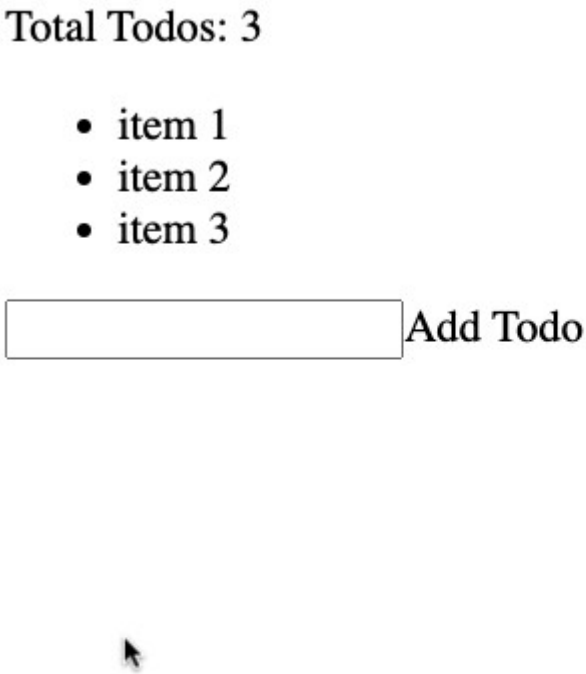
Now the last step is to be able to add a new todo.
This is where your setter function, setTodos, comes in. To update your todo state, you don't need to pass down both values, the variable and the setter function – all you need to do is pass down setTodos.
You'll pass it down to addTodo as a prop of the same name (setTodos) and destructure it from props.
As you can see, you're using your form on submit to get access to the input's value – whatever was typed into it, which you're putting within a local variable named todo.
Instead of needing to pass down the current todos array, you can just use an inner function to get the previous todo's value. This allows you to get previous todos and just return what you want the new state to be.
This new state will be an array, in which you will spread all of the previous todos and add your new todo as the last element in that array:

```
import React from "react";

export default function App() {
  const [todos, setTodos] = React.useState(["item 1", "item 2", "item 3"]);

  return (
    <>
      <TodoCount todos={todos} />
```

```
      <TodoList todos={todos} />
      <AddTodo setTodos={setTodos} />
    </>
  );
}

function AddTodo({ setTodos }) {
  function handleSubmit(event) {
    event.preventDefault();
    const todo = event.target.elements.todo.value;
    setTodos(prevTodos => [...prevTodos, todo]);
  }

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" id="todo" />
      <button type="submit">Add Todo</button>
    </form>
  );
}
```

Not only by lifting state up and passing its state variable down to the components that need to read from it can we use this pattern – we can also use it for callbacks to be able to update state.

Once you add a new item to your todo list, it's immediately added to state. Then you see your TodoList component re-render to display that new item, as well as TodoCount to show the total number of todos which is now 4:



Conclusion

Lifting state up is an important pattern for React developers because sometimes we have state that's located within a particular component that also needs to be shared with sibling components.
Instead of using an entire state management library like Redux or React Context, we can just lift the state up to the closest common ancestor and pass both the state variables the state values down as well as any callbacks to update that state.