Front End

Class 4

August 4, 2021

Agenda

- Studio Review
- Kahoot
- Code walkthrough with slides
- Studio

React Data Flow

React is light weight. It can work with different libraries

React element is a JS object

Converted to DOM element.

React is a user interface library.

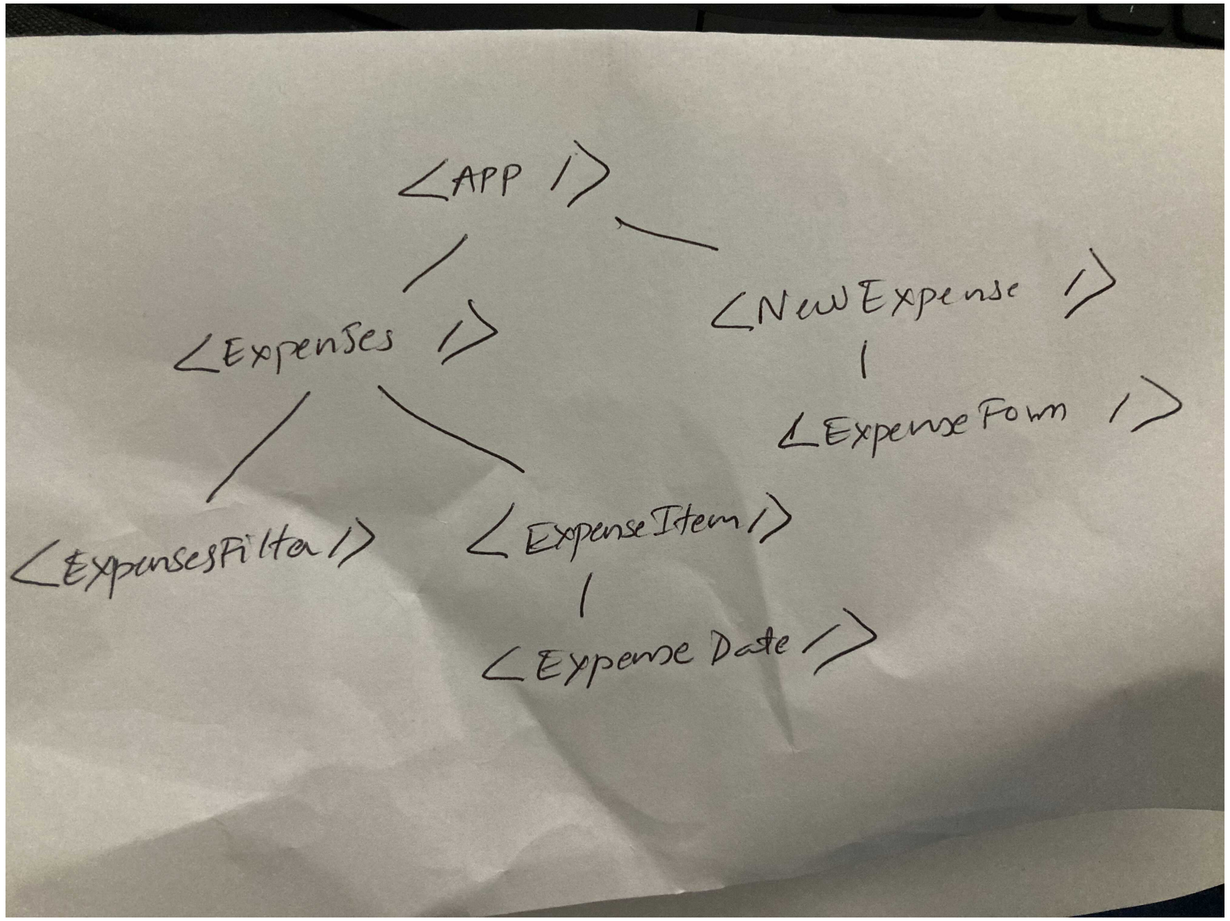
Throughout components – data flowss through a React app.

Adding Two-way Binding

Input can be changed from UI or programmatically.

```
<input type="text" value={enteredTitle} onChange={titleChangeHandler} />
```

Component Tree



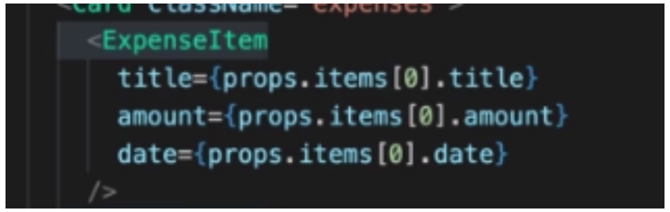
Child-Parent Component Communication

Props can be used to pass data from parent to child.

Can’t skip intermediate components.

Need to pass data collected in ExpenseForm into App component.

So far, we learnt to pass data from parent to child using props. For example, we learnt how to pass data in Expenses into ExpenseItem.



```
  <ExpenseItem
    title={props.items[0].title}
    amount={props.items[0].amount}
    date={props.items[0].date}
  />
```

How can we pass data from ExpenseForm into App?

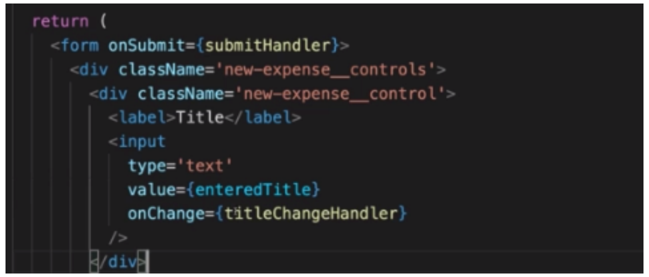
Child-Parent Component Communication

In ExpenseForm we are listening to user input. For example, changes to titleInput.

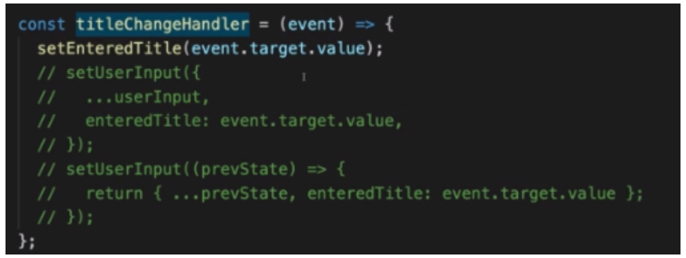
When user types there the function titleChangeHandler executes and we get default event object

We have the input element and that an be thought of as a react built-in component.

We set a special onChange prop and that takes a function as a value. It creates a listener for that event.



```
return (
  <form onSubmit={submitHandler}>
    <div className='new-expense__controls'>
      <div className='new-expense__control'>
        <label>Title</label>
        <input
          type='text'
          value={enteredTitle}
          onChange={titleChangeHandler}
        />
      </div>
```



```
const titleChangeHandler = (event) => {
  setEnteredTitle(event.target.value);
  // setUserInput({
  //   ...userInput,
  //   enteredTitle: event.target.value,
  // });
  // setUserInput((prevState) => {
  //   return { ...prevState, enteredTitle: event.target.value };
  // });
};
```

Child-Parent Component Communication

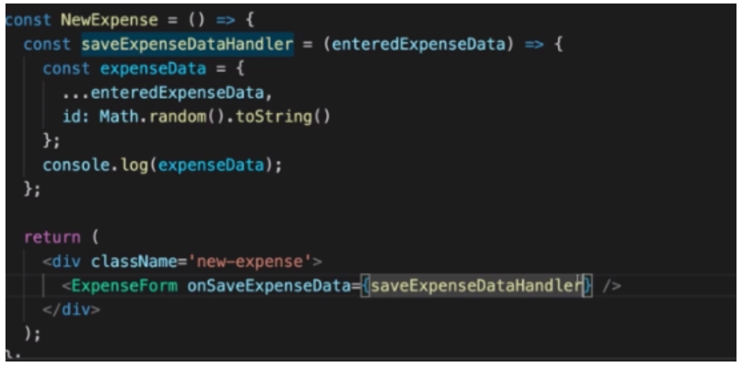
We can create our own event props for our components and expect functions as values and that will allow us to pass functions from parent to child component and then call that function from inside the child component. When we call the function, we can pass data as a parameter to the function. That is how we communication up from child to parent, and pass data from child to parent.

Child-Parent Component Communication

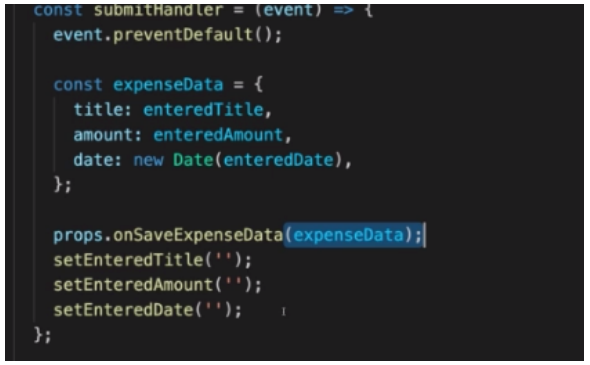
Passing data from ExpenseForm to NewExpense first, then the app component uses NewExpense.

We cannot skip components.

First step -> pass data from ExpenseForm to NewExpense.



```
const NewExpense = () => {
  const saveExpenseDataHandler = (enteredExpenseData) => {
    const expenseData = {
      ...enteredExpenseData,
      id: Math.random().toString()
    };
    console.log(expenseData);
  };

  return (
    <div className='new-expense'>
      <ExpenseForm onSaveExpenseData={saveExpenseDataHandler} />
    </div>
  );
```

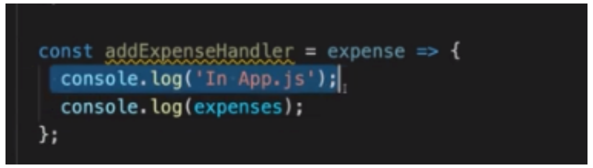


```
  const submitHandler = (event) => {
    event.preventDefault();

    const expenseData = {
      title: enteredTitle,
      amount: enteredAmount,
      date: new Date(enteredDate),
    };

    props.onSaveExpenseData(expenseData);
    setEnteredTitle('');
    setEnteredAmount('');
    setEnteredDate('');
  };
```

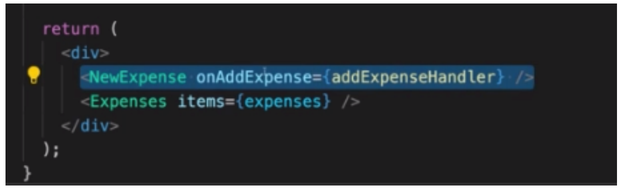
Child-Parent Component Communication

Passing data from NewExpense to App component.

App component

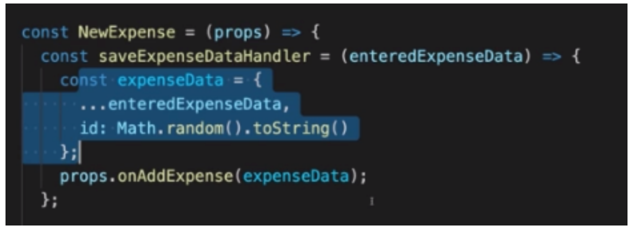


```
const addExpenseHandler = expense => {
  console.log('In App.js');
  console.log(expenses);
};
```

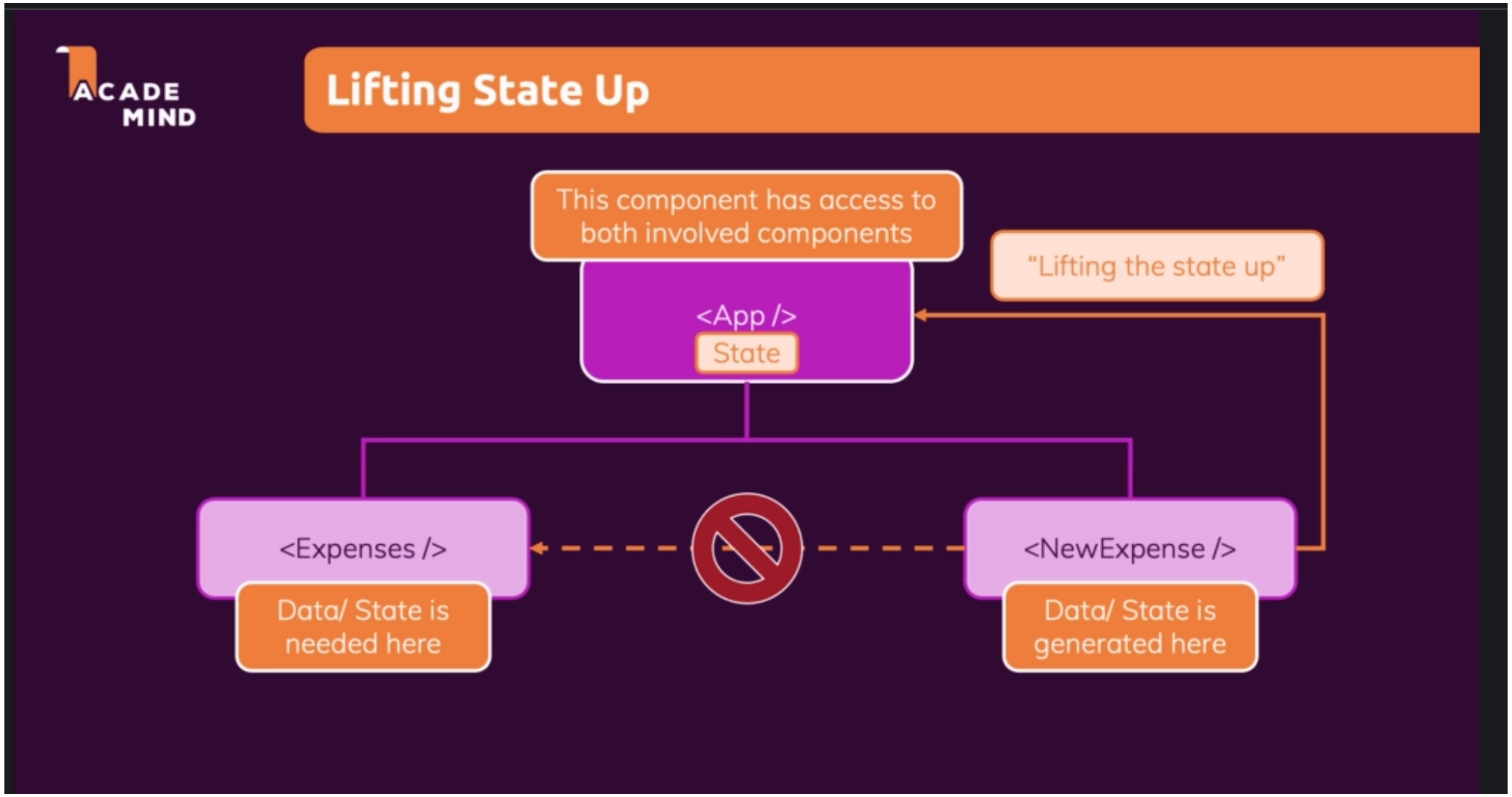


```
  return (
    <div>
      <NewExpense onAddExpense={addExpenseHandler} />
      <Expenses items={expenses} />
    </div>
  );
}
```

NewExpense component



```
const NewExpense = (props) => {
  const saveExpenseDataHandler = (enteredExpenseData) => {
    const expenseData = {
      ...enteredExpenseData,
      id: Math.random().toString()
    };
    props.onAddExpense(expenseData);
  };
```


ACADE MIND
Lifting State Up
This component has access to both involved components
"Lifting the state up"
<App />
State
<Expenses />
Data/ State is needed here
<NewExpense />
Data/ State is generated here

Related to lifting state up.
Basic component tree. Sibling components.
Parent and child can communicate with data,
App component has access to Expenses and New Expense component,
we can save state in closest involved component. Passing data is lifting the data/state up.

Assignment 2 – Events and State

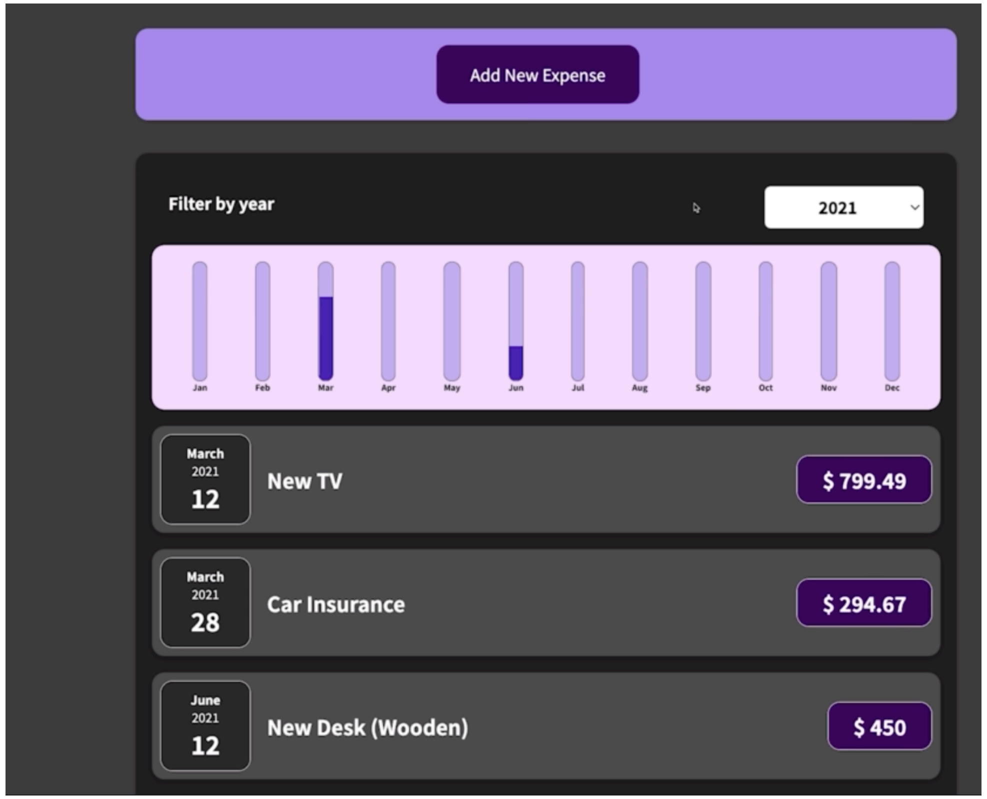
New component – ExpenseFilter (filter component)

Use styling and markup attached with js provided

Listen to changes to drop down.

Picked Year picked by user is passes to expenses component. Store in a state in the expense's component.

Pass the picked year from ExpenseFilter to Expenses. Store in state in Expenses.



March 2021 12 New TV $ 799.49

March 2021 28 Car Insurance $ 294.67

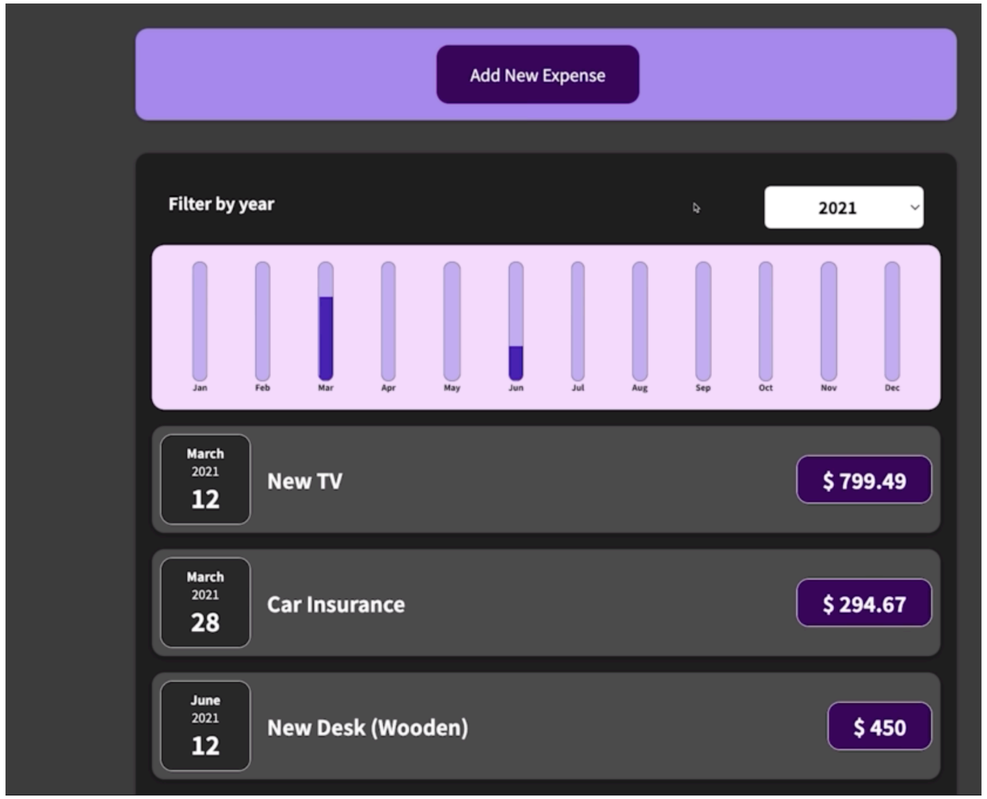
June 2021 12 New Desk (Wooden) $ 450

Assignment 2 – Events and State

Value from ExpensesFilter component passed in via props from parent components

ExpensesFilter presents the dropdown and listens.

Value and changes to value are not handled in the Component itself but in the parent (Expenses).



March 2021 12 New TV $ 799.49

March 2021 28 Car Insurance $ 294.67

June 2021 12 New Desk (Wooden) $ 450

Controlled Component

Value used in component is passed on to a parent component using props and is used by the parent component.

Logic is in parent component. Hence, it’s a controlled component. It’s not handled in the component itself but in the parent.

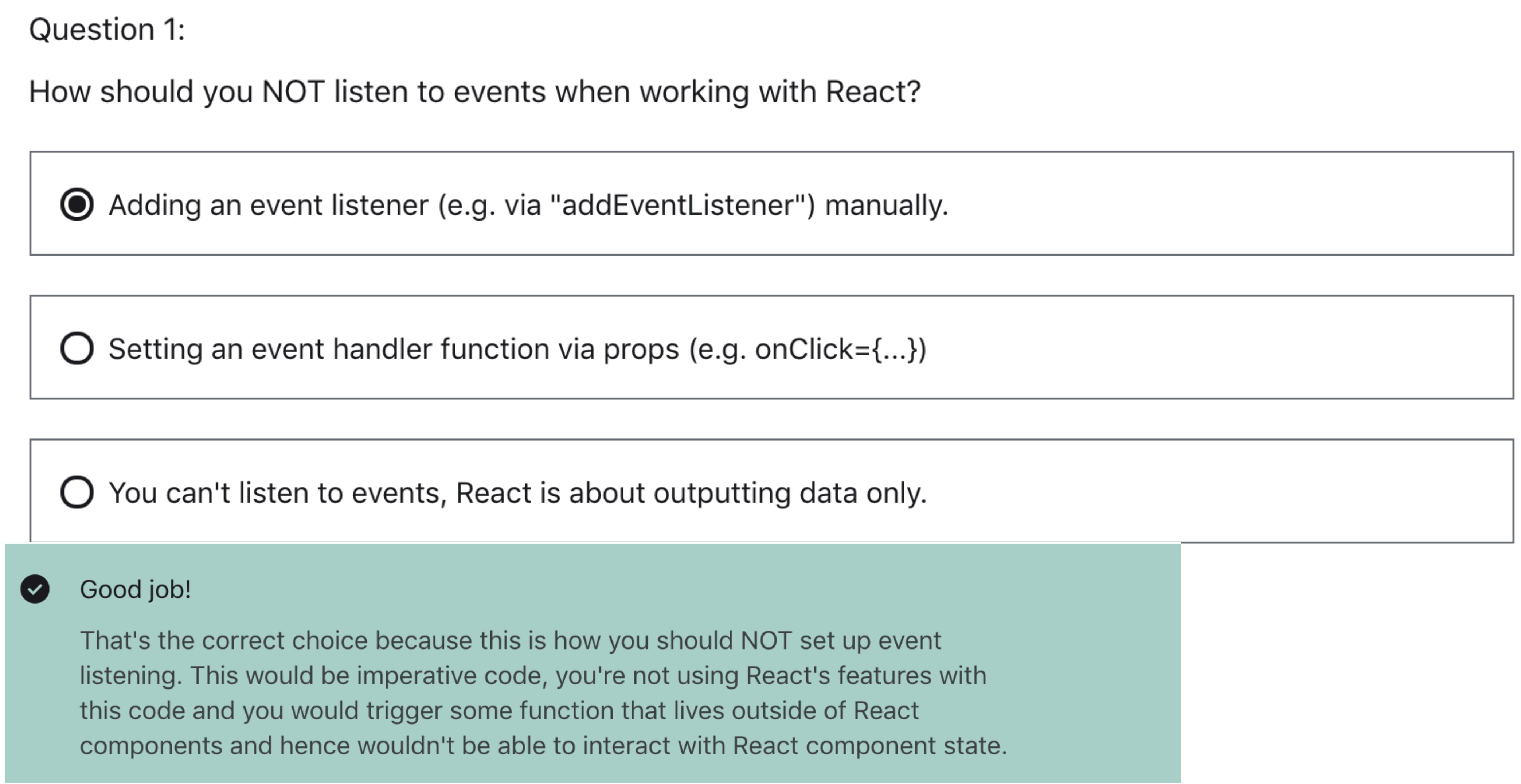
Presentational(stateless) versus stateful.
Dumb versus smart component. ExpenseItem is stateless.

Expense component manages filter state.
Presentational components are just to output some data. Nothing wrong with that.

Apps are split up into small reusable components, and most components will not have state.

Some components have state and is managed and passed around using props. This is a standard pattern.

Quiz



Question 1:

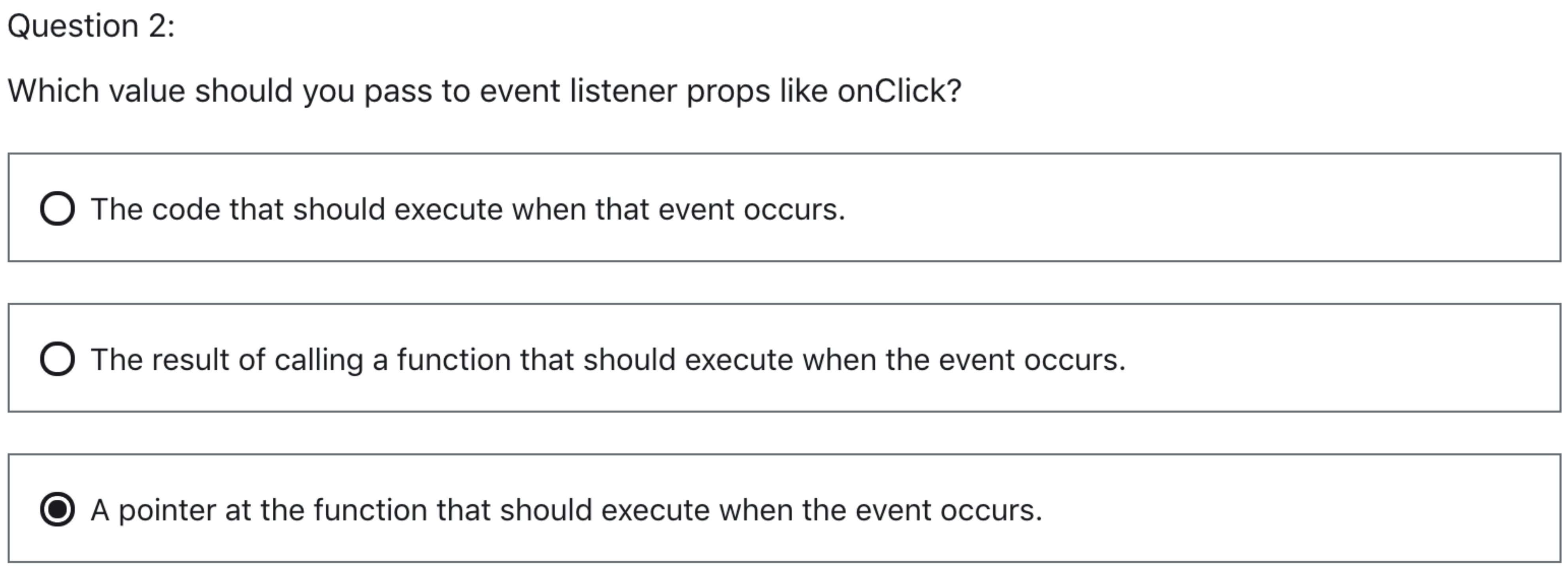
How should you NOT listen to events when working with React?

- [✓ Adding an event listener (e.g. via "addEventListener") manually.]
- ○ Setting an event handler function via props (e.g. onClick={...})
- ○ You can't listen to events, React is about outputting data only.

✓ Good job!

That's the correct choice because this is how you should NOT set up event listening. This would be imperative code, you're not using React's features with this code and you would trigger some function that lives outside of React components and hence wouldn't be able to interact with React component state.

Quiz

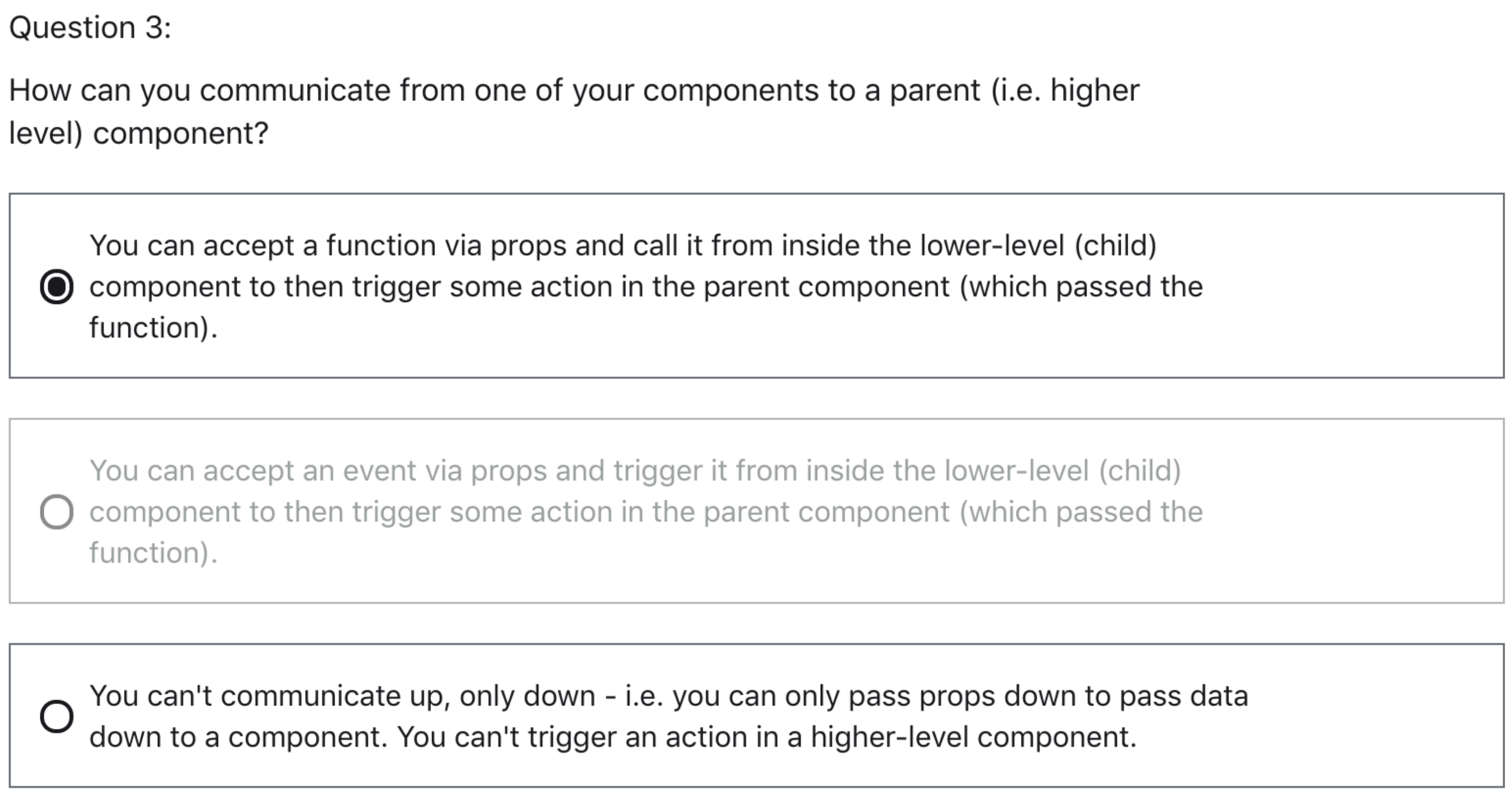


Question 2:

Which value should you pass to event listener props like onClick?

- ○ The code that should execute when that event occurs.
- ○ The result of calling a function that should execute when the event occurs.
- [✓ A pointer at the function that should execute when the event occurs.]

Quiz

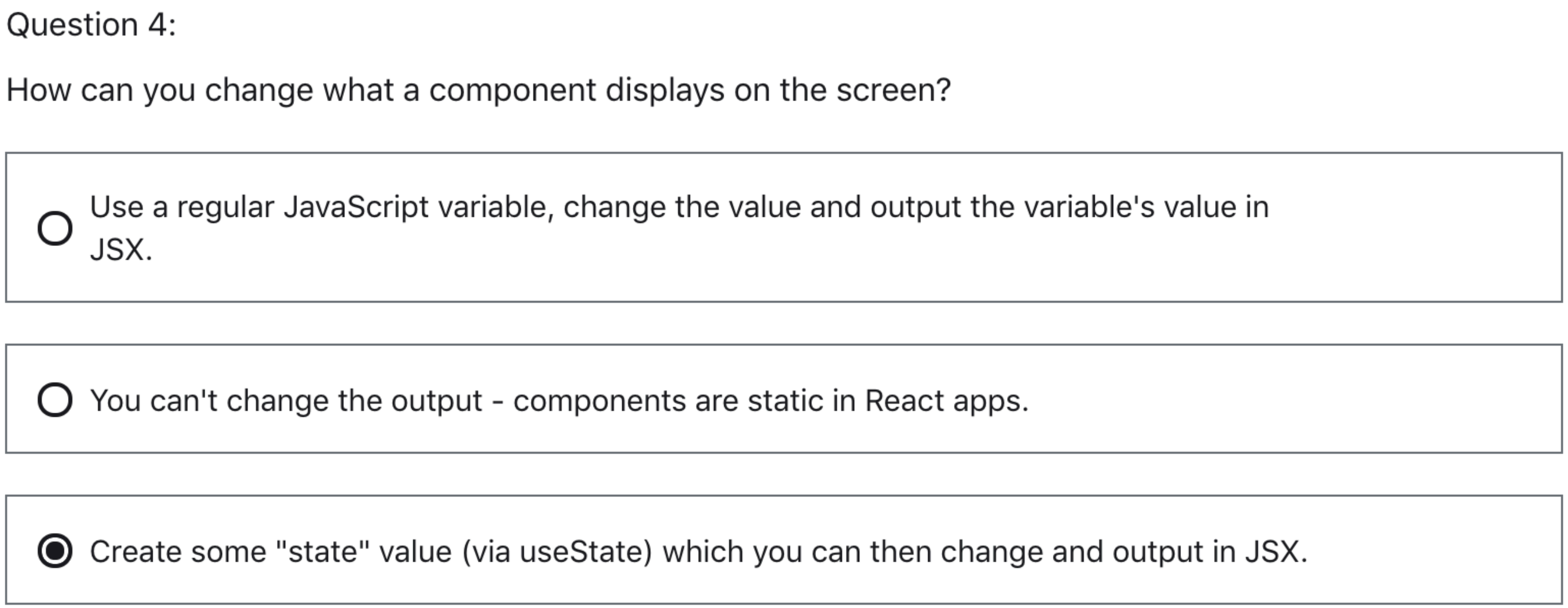


Question 3:

How can you communicate from one of your components to a parent (i.e. higher level) component?

- [✓ You can accept a function via props and call it from inside the lower-level (child) component to then trigger some action in the parent component (which passed the function).]
- You can accept an event via props and trigger it from inside the lower-level (child) component to then trigger some action in the parent component (which passed the function).
- You can't communicate up, only down - i.e. you can only pass props down to pass data down to a component. You can't trigger an action in a higher-level component.

Quiz

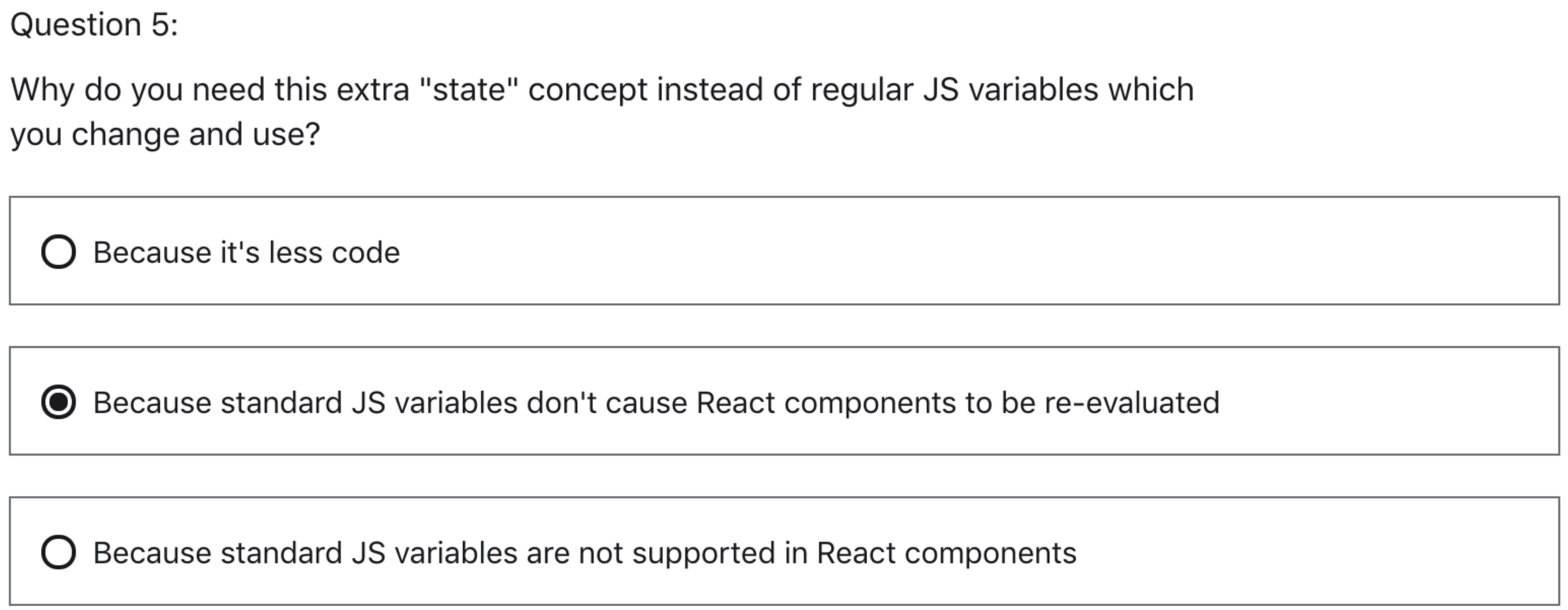


Question 4:

How can you change what a component displays on the screen?

- Use a regular JavaScript variable, change the value and output the variable's value in JSX.
- You can't change the output - components are static in React apps.
- [✓ Create some "state" value (via useState) which you can then change and output in JSX.]

Quiz

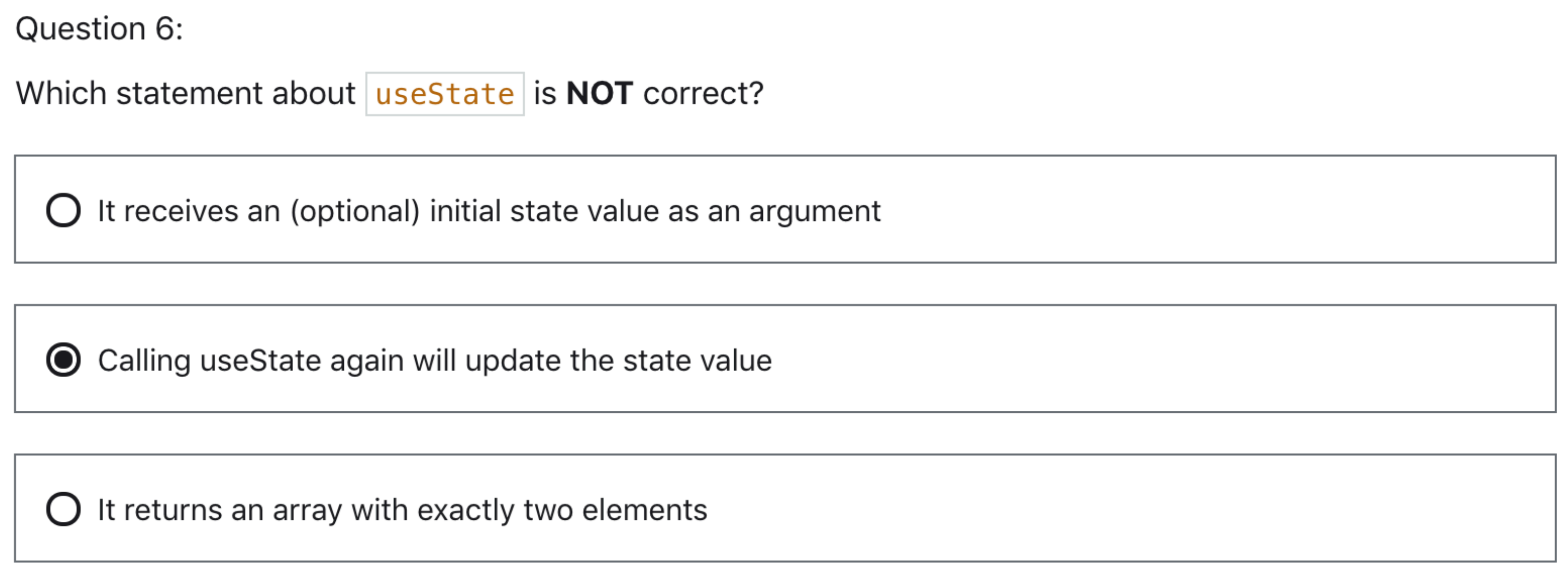


Question 5:

Why do you need this extra "state" concept instead of regular JS variables which you change and use?

- ○ Because it's less code
- [✓ Because standard JS variables don't cause React components to be re-evaluated]
- ○ Because standard JS variables are not supported in React components

Quiz

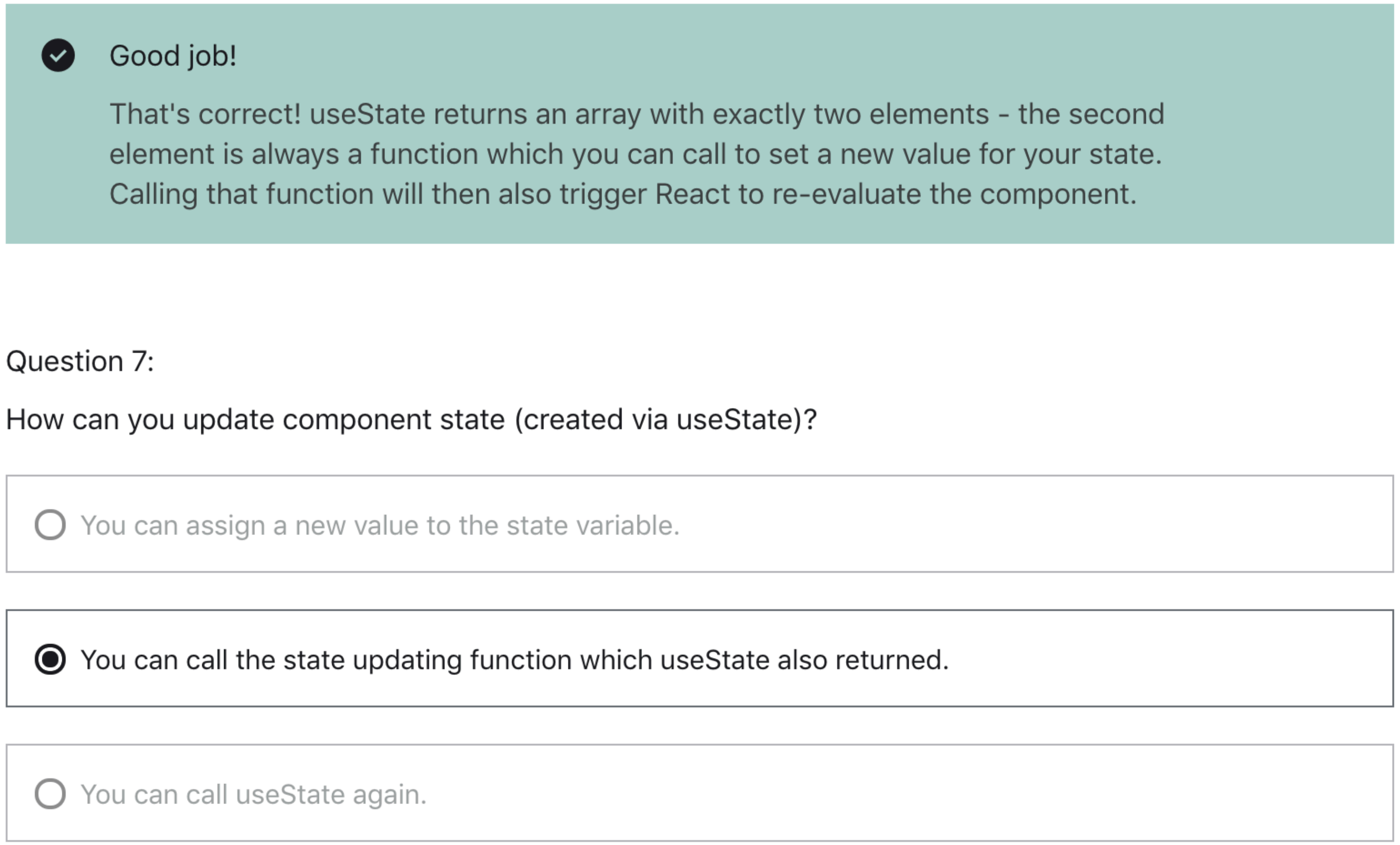


Question 6:

Which statement about `useState` is **NOT** correct?

- It receives an (optional) initial state value as an argument
- [✓ Calling useState again will update the state value]
- It returns an array with exactly two elements

Quiz



Good job!

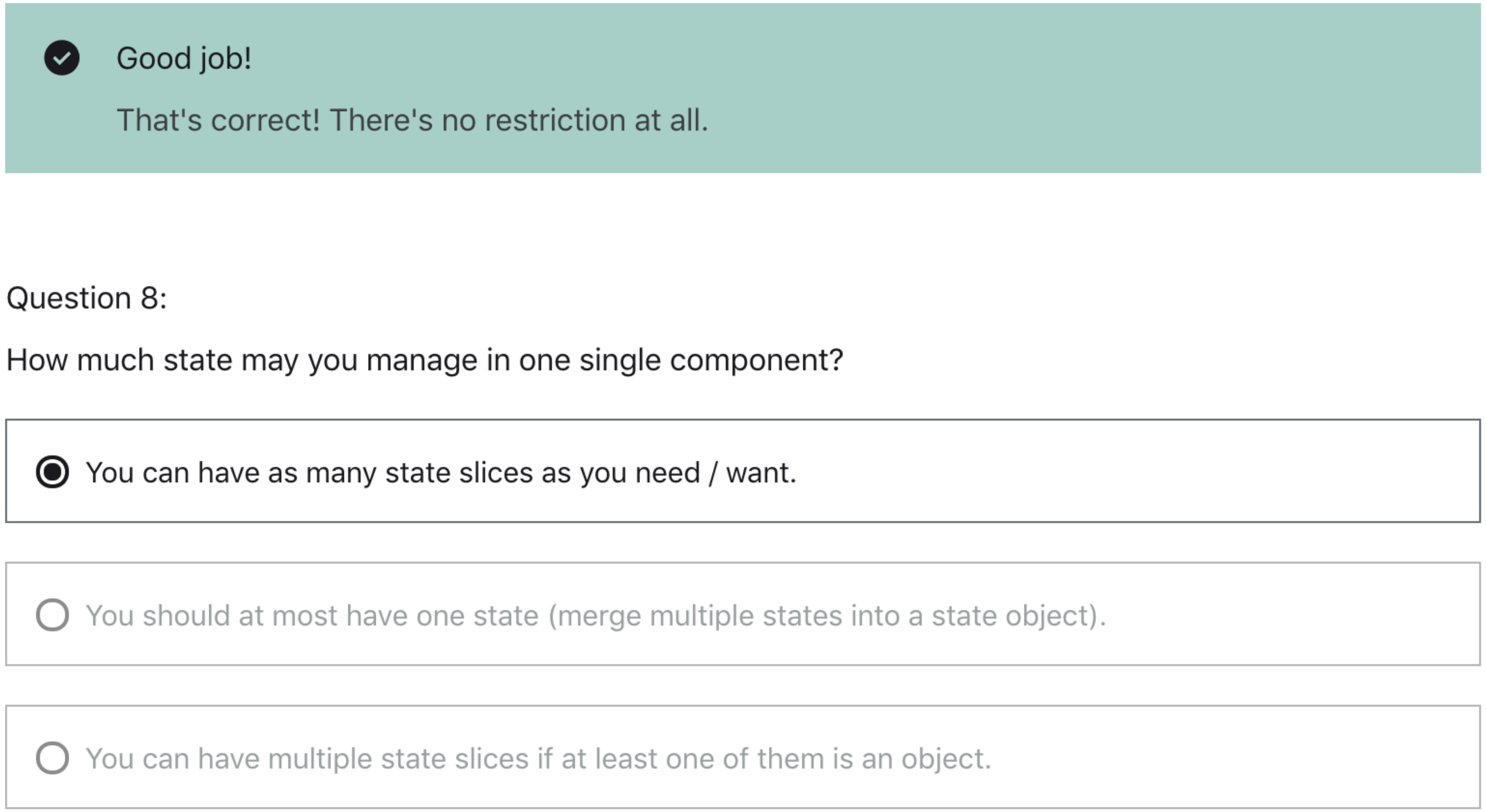
That's correct! useState returns an array with exactly two elements - the second element is always a function which you can call to set a new value for your state. Calling that function will then also trigger React to re-evaluate the component.

Question 7:

How can you update component state (created via useState)?

- You can assign a new value to the state variable.
- [✓ You can call the state updating function which useState also returned.]
- You can call useState again.

Quiz



Good job!

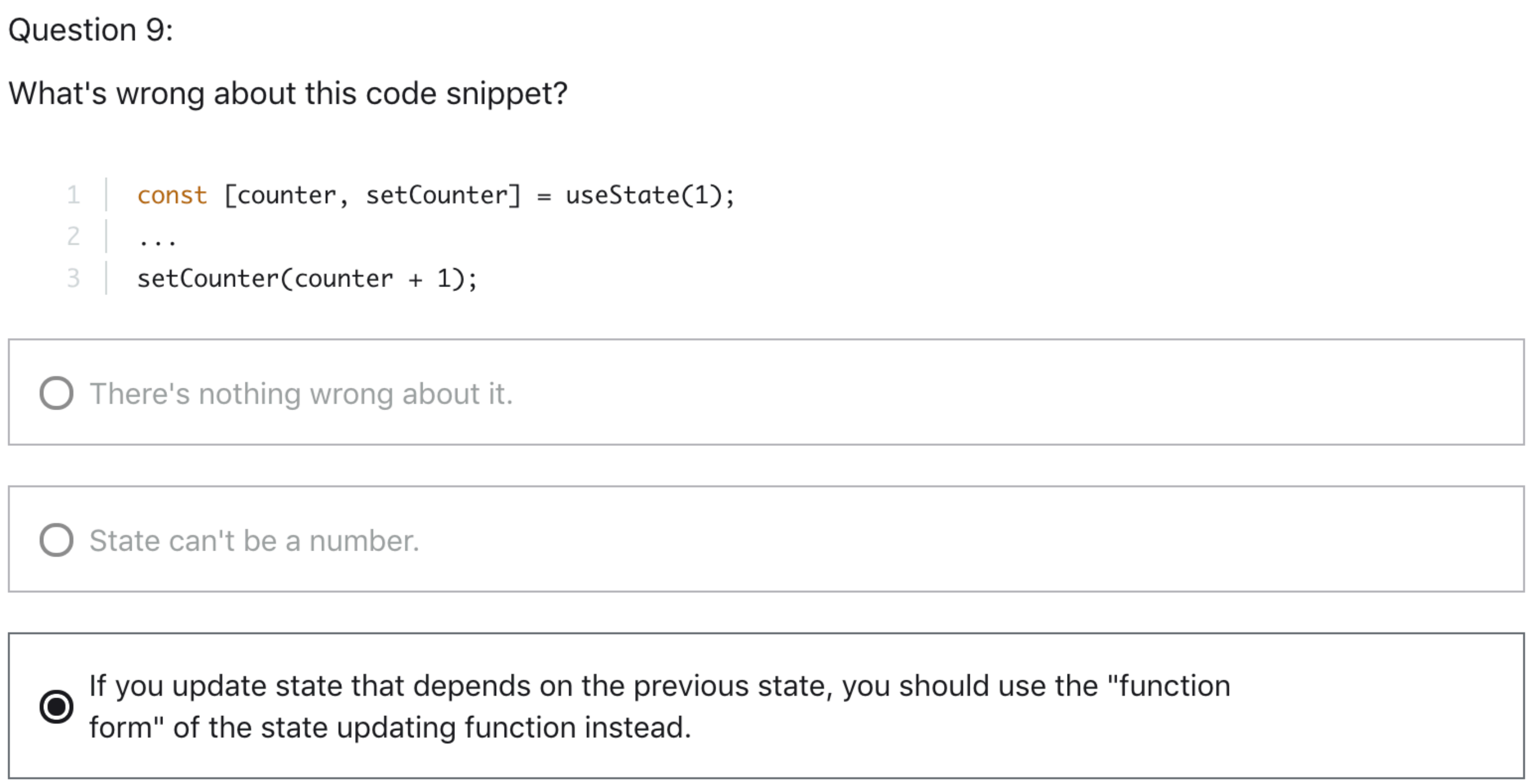
That's correct! There's no restriction at all.

Question 8:

How much state may you manage in one single component?

- [✓ You can have as many state slices as you need / want.]
- You should at most have one state (merge multiple states into a state object).
- You can have multiple state slices if at least one of them is an object.

Quiz



Question 9:

What's wrong about this code snippet?

```
const [counter, setCounter] = useState(1);
...
setCounter(counter + 1);
```

- ○ There's nothing wrong about it.
- ○ State can't be a number.
- [✓ If you update state that depends on the previous state, you should use the "function form" of the state updating function instead.]

Slides

https://github.com/academind/react-complete-guide-code/blob/04-react-state-events/slides/slides.pdf